**ТУРБО-ПАСКАЛЬ СПРАВОЧНОЕ РУКОВОДСТВО**

**ВЕРСИЯ 3.0**

**Orionsoft (C) 1996**

**В. Пушков**

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| ВВЕДЕНИЕ | 8 |
| Язык Паскаль | 8 |
| Турбо-Паскаль | 8 |
| ГЛАВА 1 КАК ПОЛЬЗОВАТЬСЯ ТУРБО-СИСТЕМОЙ | 10 |
| Приступая к работе | 10 |
| Важное замечание | 10 |
| Файлы Турбо-системы | 10 |
| Как запустить Турбо-Паскаль | 11 |
| Инсталляция | 12 |
| Инсталляция экрана | 13 |
| Инсталляция команд встроенного редактора | 13 |
| Меню турбо-системы | 13 |
| Выбор текущего дисковода | 14 |
| Выбор рабочего файла | 14 |
| Выбор главного файла | 15 |
| Команда запуска редактора | 16 |
| Команда запуска компилятора | 16 |
| Команда запуска на выполнение | 16 |
| Команда сохранения рабочего файла | 16 |
| Команда "Выполнить" | 16 |
| Команда вывода оглавления | 17 |
| Команда окончания работы | 17 |
| Опции компилятора | 17 |
| Команды M, C, H | 17 |
| Начальный адрес | 18 |
| Конечный адрес | 18 |
| Поиск ошибок периода выполнения | 18 |
| Выход из меню опций | 19 |
| Редактор системы Турбо | 19 |
| Строка состояния | 19 |
| Команды редактирования | 20 |
| Заметка об управляющих символах | 21 |
| Начиная работу, нужно знать, как ее закончить | 22 |
| Основные команды перемещения курсора | 22 |
| Расширенные команды перемещения курсора | 23 |
| Команды вставки и удаления | 24 |
| Блочные команды | 26 |
| Прочие команды редактирования | 27 |
| Сравнение редакторов Турбо и WordStar | 31 |
| Перемещение курсора | 31 |
| Пометить одиночное слово | 31 |
| Закончить редактирование | 31 |
| Восстановить строку | 31 |
| Табуляция | 31 |
| Автоматический отступ | 31 |
| ГЛАВА 2 ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА | 32 |
| Основные символы | 32 |
| Зарезервированные слова | 32 |
| Стандартные идентификаторы | 32 |
| Разделители | 33 |
| Строки программы | 33 |
| ГЛАВА 3 СТАНДАРТНЫЕ ПРОСТЫЕ ТИПЫ | 34 |
| INTEGER - целый | 34 |
| BYTE - байт | 34 |
| REAL - вещественный | 34 |
| BOOLEAN - логический | 35 |
| CHAR - символьный | 35 |
| ГЛАВА 4 ЭЛЕМЕНТЫ ЯЗЫКА, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ | 36 |
| Идентификаторы | 36 |
| Числа | 36 |
| Строки | 36 |
| Управляющие символы | 38 |
| Комментарии | 38 |
| Директивы компилятора | 39 |
| ГЛАВА 5 ЗАГОЛОВОК ПРОГРАММЫ И ПРОГРАММНЫЙ БЛОК | 40 |
| Заголовок программы | 40 |
| Разделы описаний | 40 |
| Раздел описаний меток | 40 |
| Раздел определений констант | 41 |
| Раздел определений типов | 41 |
| Раздел описаний переменных | 42 |
| Абсолютные переменные | 42 |
| Раздел описаний процедур и функций | 43 |
| Операторная часть | 43 |
| ГЛАВА 6 ВЫРАЖЕНИЯ | 44 |
| Операции | 44 |
| Унарный минус | 44 |
| Операция NOT | 44 |
| Операции группы умножения | 45 |
| Операции группы сложения | 45 |
| Операции отношений | 46 |
| Вызовы функций | 46 |
| ГЛАВА 7 ОПЕРАТОРЫ | 47 |
| Простые операторы | 47 |
| Оператор присваивания | 47 |
| Вызов процедуры | 47 |
| Оператор перехода | 48 |
| Пустой оператор | 48 |
| Структурные операторы | 48 |
| Составной оператор | 48 |
| Условные операторы | 49 |
| Оператор IF | 49 |
| Оператор CASE | 50 |
| Операторы цикла | 51 |
| Оператор FOR | 51 |
| Оператор WHILE | 52 |
| Оператор REPEAT | 52 |
| ГЛАВА 8 ПРОСТЫЕ И ОГРАНИЧЕННЫЕ ТИПЫ | 53 |
| Простой тип | 53 |
| Ограниченный тип | 54 |
| Преобразование типов | 54 |
| Проверка границ | 55 |
| ГЛАВА 9 СТРОКОВЫЙ ТИП | 56 |
| Определение строкового типа | 56 |
| Строковые выражения | 56 |
| Строковое присваивание | 57 |
| Процедуры обработки строк | 57 |
| Процедура DELETE | 57 |
| Процедура INSERT | 57 |
| Процедура STR | 58 |
| Процедура VAL | 58 |
| Функции обработки строк | 59 |
| Функция COPY | 59 |
| Функция CONCAT | 59 |
| Функция LENGTH | 59 |
| Функция POS | 59 |
| ГЛАВА 10 МАССИВЫ | 61 |
| Определение массива | 61 |
| Многомерные массивы | 62 |
| Массивы символов | 63 |
| Стандартные массивы | 63 |
| Массив MEM | 63 |
| Массив PORT | 64 |
| Оптимизация индексов массивов | 64 |
| ГЛАВА 11 ЗАПИСИ | 65 |
| Определение записи | 65 |
| Оператор WITH | 67 |
| Записи с вариантами | 68 |
| ГЛАВА 12 МНОЖЕСТВА | 70 |
| Определение типа множества | 70 |
| Выражения с множествами | 71 |
| Множественные конструкции | 71 |
| Операции над множествами | 71 |
| Присваивание значений множествам | 72 |
| ГЛАВА 13 КОНСТАНТЫ С ЗАДАННЫМ ТИПОМ | 73 |
| Неструктурные константы с заданным типом | 73 |
| Структурные константы с заданным типом | 74 |
| Константы-массивы | 74 |
| Многомерные константы-массивы | 75 |
| Константы-записи | 75 |
| Константы-множества | 76 |
| ГЛАВА 14 ФАЙЛЫ | 77 |
| Определение типа файла | 77 |
| Операции над файлами | 78 |
| Процедура ASSIGN | 78 |
| Процедура REWRITE | 78 |
| Процедура RESET | 78 |
| Процедура READ | 78 |
| Процедура WRITE | 78 |
| Процедура SEEK | 79 |
| Процедура FLUSH | 79 |
| Процедура CLOSE | 79 |
| Процедура ERASE | 79 |
| Процедура RENAME | 79 |
| Файловые стандартные функции | 80 |
| Функция EOF | 80 |
| Функция FILEPOS | 80 |
| Функция FILESIZE | 81 |
| Использование файлов | 81 |
| Текстовые файлы | 84 |
| Операции над текстовыми файлами | 84 |
| Процедура READLN | 84 |
| Процедура WRITELN | 84 |
| Функция EOLN | 84 |
| Функция SEEKEOLN | 85 |
| Функция SEEKEOF | 85 |
| Логические устройства | 86 |
| Пользовательские драйверы ввода/вывода | 88 |
| Стандартные файлы | 89 |
| Ввод и вывод текста | 91 |
| Процедура READ | 91 |
| Процедура READLN | 93 |
| Процедура WRITE | 93 |
| Процедура WRITELN | 95 |
| Файлы, не имеющие типа | 95 |
| Процедуры BLOCKREAD и BLOCKWRITE | 96 |
| Контроль ввода/вывода | 97 |
| ГЛАВА 15 УКАЗАТЕЛИ | 99 |
| Определение переменной-указателя | 99 |
| Размещение переменных (процедура NEW) | 99 |
| Процедуры MARK и RELEASE | 100 |
| Функция MEMAVAIL | 101 |
| Использование указателей | 101 |
| Процедура DISPOSE | 103 |
| Процедура GETMEM | 103 |
| Процедура FREEMEM | 104 |
| Функция MAXAVAIL | 104 |
| Указатели и целые | 104 |
| Полезные советы | 104 |
| ГЛАВА 16 ПРОЦЕДУРЫ И ФУНКЦИИ | 105 |
| Параметры | 105 |
| Ослабление ограничений на типы параметров | 106 |
| Параметры-переменные, не имеющие типа | 107 |
| Процедуры | 108 |
| Описание процедуры | 108 |
| Стандартные процедуры | 110 |
| Экранные процедуры | 110 |
| Процедура CLREOL | 110 |
| Процедура CLRSCR | 110 |
| Процедура CRTINIT | 111 |
| Процедура CRTEXIT | 111 |
| Процедура DELLINE | 111 |
| Процедура INSLINE | 111 |
| Процедура GOTOXY | 111 |
| Процедура LOWVIDEO | 111 |
| Процедура NORMVIDEO | 111 |
| Специальные процедуры | 112 |
| Процедура DELAY | 112 |
| Процедура EXIT | 112 |
| Процедура HALT | 112 |
| Процедура MOVE | 112 |
| Процедура FILLCHAR | 112 |
| Процедура RANDOMIZE | 113 |
| Функции | 113 |
| Описание функции | 113 |
| Стандартные функции | 114 |
| Арифметические функции | 115 |
| Функция ABS | 115 |
| Функция ARCTAN | 115 |
| Функция COS | 115 |
| Функция EXP | 115 |
| Функция FRAC | 115 |
| Функция INT | 115 |
| Функция LN | 116 |
| Функция SIN | 116 |
| Функция SQR | 116 |
| Функция SQRT | 116 |
| Функции для работы с простыми типами | 116 |
| Функция PRED | 116 |
| Функция SUCC | 116 |
| Функция ODD | 117 |
| Функции преобразования типов | 117 |
| Функция CHR | 117 |
| Функция ORD | 117 |
| Функция ROUND | 117 |
| Функция TRUNC | 117 |
| Прочие стандартные функции | 118 |
| Функция ADDR | 118 |
| Функция HI | 118 |
| Функция KEYPRESSED | 118 |
| Функция LO | 118 |
| Функция RANDOM | 118 |
| Функция PARAMCOUNT | 118 |
| Функция PARAMSTR | 119 |
| Функция SIZEOF | 119 |
| Функция SWAP | 119 |
| Функция UPCASE | 119 |
| Вызовы функций операционной системы | 119 |
| Процедура и функция BDOS | 119 |
| Функция BDOSHL | 120 |
| Процедура и функция BIOS | 120 |
| Функция BIOSHL | 120 |
| Предварительные объявления | 120 |
| Внешние подпрограммы | 122 |
| ГЛАВА 17 ВКЛЮЧЕНИЕ ФАЙЛОВ В КОМПИЛЯЦИЮ | 123 |
| ГЛАВА 18 ОВЕРЛЕЙНЫЕ СТРУКТУРЫ | 125 |
| Создание оверлеев | 126 |
| Вложенные оверлейные структуры | 128 |
| Автоматическое управление оверлеями | 130 |
| Размещение оверлейных файлов | 130 |
| Эффективное использование оверлеев | 131 |
| Ограничения на оверлейные структуры | 132 |
| Область данных | 132 |
| Предварительные объявления | 132 |
| Рекурсия | 132 |
| Ошибки периода выполнения | 132 |
| ГЛАВА 19 ПРОЦЕДУРЫ CHAIN И EXECUTE | 133 |
| ГЛАВА 20 ВКЛЮЧЕНИЕ МАШИННОГО КОДА В ИСХОДНЫЙ ТЕКСТ | 135 |
| ГЛАВА 21 ОБРАБОТКА ПРЕРЫВАНИЙ | 137 |
| ГЛАВА 22 ВНУТРЕННИЕ ФОРМАТЫ ДАННЫХ | 138 |
| Базовые типы данных | 138 |
| Простые типы | 138 |
| Данные типа REAL | 138 |
| Строки | 139 |
| Множества | 139 |
| Интерфейсные блоки файлов | 140 |
| Указатели | 141 |
| Структуры данных | 141 |
| Массивы | 141 |
| Записи | 142 |
| Дисковые файлы | 142 |
| Файлы произвольного доступа | 142 |
| Текстовые файлы | 142 |
| Параметры | 142 |
| Параметры-переменные | 143 |
| Параметры-значения | 143 |
| Простые типы | 143 |
| Данные типа REAL | 143 |
| Строки - данные типа STRING | 143 |
| Множества | 144 |
| Указатели | 144 |
| Массивы и записи | 144 |
| Результаты функций | 144 |
| Хип и стеки | 145 |
| ГЛАВА 23 РАСПРЕДЕЛЕНИЕ ПАМЯТИ | 147 |
| Компиляция в памяти | 147 |
| Компиляция на диск | 148 |
| Выполнение программы, хранимой в памяти | 148 |
| Выполнение программы, хранимой на диске | 149 |
| ПРИЛОЖЕНИЕ A СТАНДАРТНЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ | 151 |
| ПРИЛОЖЕНИЕ B ОПЕРАЦИИ | 156 |
| ПРИЛОЖЕНИЕ C ДИРЕКТИВЫ КОМПИЛЯТОРА | 157 |
| ПРИЛОЖЕНИЕ D СРАВНЕНИЕ ТУРБО-ПАСКАЛЯ СО СТАНДАРТНЫМ | 161 |
| ПРИЛОЖЕНИЕ E СООБЩЕНИЯ КОМПИЛЯТОРА ОБ ОШИБКАХ | 163 |
| ПРИЛОЖЕНИЕ F СООБЩЕНИЯ ОБ ОШИБКАХ ПЕРИОДА ВЫПОЛНЕНИЯ | 167 |
| ПРИЛОЖЕНИЕ G СООБЩЕНИЯ ОБ ОШИБКАХ ВВОДА/ВЫВОДА | 169 |
| ПРИЛОЖЕНИЕ H ПЕРЕВОД СООБЩЕНИЙ ОБ ОШИБКАХ КОМПИЛЯЦИИ | 171 |
| ПРИЛОЖЕНИЕ I ТАБЛИЦА КОДА ASCII | 175 |
| ПРИЛОЖЕНИЕ J ИНСТАЛЛЯЦИЯ | 176 |
| ПРИЛОЖЕНИЕ K СОВЕТЫ И ОТВЕТЫ НА ВОПРОСЫ | 183 |

**ВВЕДЕНИЕ**

Эта книга представляет собой справочное руководство по работе с системой Турбо Паскаль в среде операционной системы CP/M-80 (OSDOS). Несмотря на широкое использование примеров, эта книга не является учебником по Паскалю, и для работы с ней требуется знание по крайней мере основ этого языка.

За основу руководства взят перевод "TURBO PASCAL. REFERENCE MANUAL. VERSION 3.0", Всесоюзный центр переводов, 1986.

**Язык Паскаль**

Паскаль - это язык программирования высокого уровня, не имеющий узкой специализации, т.е. язык общего назначения. Создателем этого языка был профессор Цюрихского технического университета (Швейцария) Никлаус Вирт, который назвал свой язык в честь Блеза Паскаля, выдающегося французского философа и математика 17-го столетия.

Публикуя в 1971 году свое определение языка Паскаль, профессор Вирт стремился сделать вклад в методику обучения систематическому подходу к программированию, и в особенности в методику обучения структурному программированию. С той поры на Паскале программировались практически все мыслимые задачи почти на всех существующих машинах, и в настоящее время он считается одним из наиболее передовых языков высокого уровня, применяющимся как для целей обучения, так и в профессиональном программировании.

**Турбо-Паскаль**

Турбо-Паскаль спроектирован так, чтобы удовлетворить потребности всех категорий пользователей: студенту он предлагает дружескую диалоговую среду, значительно ускоряющую процесс обучения; в руках же профессионального программиста он становится чрезвычайно эффективным инструментом, обеспечивающим "чемпионские" показатели времени компиляции и выполнения программ.

Турбо-Паскаль близко следует определению стандартного Паскаля, данному в книге Иенсена и Вирта "Паскаль. Сообщение и руководство пользователя". Те небольшие и второстепенные отличия, которые все же имеются, изложены в приложении D. В дополнение к стандарту Турбо-Паскаль имеет ряд расширений, таких как:

- переменные, имеющие абсолютные адреса,

- манипуляции с битами и байтами,

- прямой доступ к памяти и портам ввода/вывода,

- динамические строки,

- свободный порядок разделов описаний,

- полная поддержка средств операционной системы,

- включение машинного кода в исходный текст программы,

- включение файлов в компиляцию,

- логические операции над целыми,

- оверлейная система,

- связывание программ в цепочку с общими переменными,

- произвольный доступ к файлам данных,

- структурные константы,

- функции преобразования типов.

Кроме того, для увеличения гибкости Турбо-Паскаля в него включено много дополнительных стандартных процедур и функций.

**ГЛАВА 1.**

**КАК ПОЛЬЗОВАТЬСЯ ТУРБО-СИСТЕМОЙ**

В этой главе рассказано о том, как настроить систему Турбо-Паскаль и о том, как с этой системой работать; в частности, описана работа встроенного редактора.

**Приступая к работе**

Перед началом работы с системой Турбо-Паскаль Вам следует, в Ваших собственных интересах, сделать рабочую копию эталонной дискеты и хранить оригинал в безопасном месте. Для получения копии воспользуйтесь программой копирования файлов; убедитесь при этом, что все файлы успешно скопировались.

Приложение K содержит ответы на наиболее часто возникающие вопросы, поэтому его полезно прочитать, если перед Вами встают какие-либо проблемы.

**Важное замечание**

Турбо-Паскаль позволяет задавать ряд директив компилятора для управления специальными средствами, работающими во время выполнения программы, такими как контроль границ индексов массивов, рекурсия и т.д. СЛЕДУЕТ УЧИТЫВАТЬ, что подразумеваемые по умолчанию установки этих директив оптимизируют время выполнения и минимизируют объем занимаемой памяти. Поэтому ряд таких средств остается выключенным до явной установки их программистом в ТЕКСТЕ ПРОГРАММЫ. Все директивы компилятора и их значения по умолчанию приведены в приложении С.

**Файлы Турбо-системы**

TURBO.COM - Программа Турбо-Паскаль: компилятор, редактор и проч. При вводе с клавиатуры команды TURBO происходит загрузка этого файла, после чего Турбо-система готова к работе.

TURBO.OVR - Оверлейный файл для TURBO.COM. Должен присутствовать на рабочей дискете, если Вы хотите исполнять. COM-файлы из TURBO.

TURBO.MSG - Текстовый файл, содержащий сообщения об ошибках. Нет необходимости в его присутствии на рабочей дискете, если Вас устраивает отсутствие диагностических сообщений об ошибках времени компиляции. В этом случае ошибки будут печататься только в виде номеров ошибок, но Вы можете узнать в приложении E их смысл. Это не только сохраняет пространство на диске, но, что более важно, дает дополнительно порядка 1.5 Кбайт оперативной памяти. В любом случае система отмечает все ошибки. Существует возможность отредактировать этот файл сообщений, если Вам понадобится перевести сообщения на другой язык или представить в другой кодировке; эта процедура описана в приложении H.

TINST.COM - программа инсталляции (настройки) Турбо-системы. Достаточно ввести с клавиатуры TINST, и эта программа проведет Вас через процедуру настройки, которая полностью управляется с помощью меню. Этот и следующие файлы не обязаны присутствовать на рабочем диске.

TINST.DTA - Данные инсталляции.

TINST.MSG - Сообщения для программы инсталляции. Этот файл также можно перевести на любой необходимый Вам язык.

TLIST.COM - Программа распечатки исходных текстов программ

Файлы.PAS - примеры Паскаль-программ.

READ.ME - Если этот файл присутствует, то в нем содержатся последние исправления, внесенные в систему, или предложения по использованию системы.

Только файл TURBO.COM обязательно должен находиться на Вашем рабочем диске. Таким образом, полностью работоспособный Турбо-Паскаль требует всего лишь 30 Кбайт дискового пространства. Файл TURBO.OVR нужен только в том случае, если Вы собираетесь исполнять программы непосредственно из меню Турбо-системы.

TURBO.MSG нужен только при необходимости формирования текстовых

сообщений об ошибках периода компиляции. Файлы TINST используются только при инсталляции (настройке).

**Как запустить Турбо-Паскаль**

Когда Вы скопируете систему на Ваш рабочий диск, введите с клавиатуры команду:

TURBO

После нажатия клавиши RETURN система начнет работу с вывода на экран следующего сообщения:

***----------------------------------------***

***TURBO Pascal System Version 3.02A***

***CP/M-80, Z80***

***Copyright (C) 1983,84,85 BORLAND Inc.***

***----------------------------------------***

***No terminal selected***

***Include error messages (Y/N)?***

В первой строке содержится номер Вашей версии и код операционной системы. Предпоследняя строка показывает, на какой терминал (дисплей) настроена система. Последняя строка содержит сообщение "Включить сообщения об ошибках?".

Если Вы введете "Y" в ответ на вопрос, файл сообщений об ошибках будет загружен в память (если он есть на диске), и появится сообщение:

Loading TURBO.MSG

Вы можете, однако, ответить "N" и сохранить тем самым около 1.5 Кбайт оперативной памяти. После этого появится главное меню Турбо-системы:

***Logged drive: A***

***Work file:***

***Main file:***

***Edit Compile Run Save***

***eXecute Dir Quit compiler Options***

***Text: 0 bytes (82A6-82A6)***

***Free: 17759 bytes (82A7-C806)***

***>\_***

Меню показывает Вам допустимые команды, каждая из которых детально описана в последующих разделах. Каждая команда выполняется путем ввода соответствующей ей заглавной буквы, при этом клавишу RETURN нажимать не нужно - команда выполняется немедленно. Значения текущего дисковода и памяти показаны только для примера.

Если Ваша Турбо-система не инсталлирована (процедура начальной установки не производилась), и Вы собираетесь использовать встроенный текстовый редактор, введите "Q", чтобы выйти из TURBO, и выполните инсталляцию.

**Инсталляция**

Наберите на клавиатуре TINST для запуска программы инсталляции. При этом все файлы TINST, а также файл TURBO.COM должны находиться в текущем дисководе. На экране появится основное меню программы TINST:

***Turbo Pascal installation menu***

***Choose installation item from the following:***

***[S]creen installation | [C]ommand installation | [Q]uit***

***Enter S,C, or Q: \_***

**Инсталляция экрана**

Нажмите "S" для запуска инсталляции экрана. Вы увидите меню, содержащее названия наиболее часто используемых терминалов, и сможете выбрать тот, который Вам подходит, введя соответствующий номер. Если Ваш терминал отсутствует в меню и не является совместимым ни с одним из указанных, то Вам придется выполнить инсталляцию самостоятельно. Это довольно прямолинейная процедура, но Вам придется обратиться к руководству по Вашему терминалу для ответа на вопросы инсталляции. Подробности смотрите в приложении J.

После выбора терминала Вам нужно решить, не хотите ли Вы модифицировать его перед инсталляцией. Это может быть нужно, если Вы имеете такой же терминал, но с некоторыми дополнительными возможностями. В этом случае добавьте требуемые команды для активизации Ваших дополнительных возможностей. Ответив "Y", Вы получите ряд дополнительных вопросов, которые описаны в приложении J.

Обычно на этот вопрос дается ответ "N" (нет), что означает Вашу удовлетворенность тем, как стандартно инсталлирован терминал. Далее Вам нужно ответить на вопрос о рабочей частоте Вашего микропроцессора, введя соответствующее значение в МГц (допустимыми являются 2,4,6 и 8).

После этого вновь появляется основное меню, и Вы можете перейти к инсталляции команд, которая описана в следующем разделе, либо закончить инсталляцию, введя "Q".

**Инсталляция команд встроенного редактора**

Встроенный редактор системы Турбо-Паскаль реагирует на ряд команд, которые перемещают курсор по экрану, вставляют, удаляют и перемещают текст и т.д. Команды настроены фирмой BORLAND под "стандарт", установленный редактором WordStar. С помощью программы инсталляции их можно легко переопределить в соответствии с Вашими вкусами и клавиатурой.

Подробное описание процесса инсталляции команд встроенного редактора Вы можете найти в приложении J.

**Меню Турбо-системы**

После инсталляции вновь вызовите систему Турбо-Паскаль, набрав на клавиатуре команду TURBO. Экран очистится, и на нем появится главное меню уже с подсвеченными буквами команд. Если это не так, то Вам следует проверить данные инсталляции.

***(L)ogged drive: A***

***(W)ork file:***

***(M)ain file:***

***(E)dit (C)ompile (R)un (S)ave***

***e(X)ecute (D)ir (Q)uit compiler (O)ptions***

***Text: 0 bytes (82A6-82A6)***

***Free: 17759 bytes (82A7-C806)***

***>\_***

(здесь для обозначения подсветки букв использованы скобки)

Заметим кстати, что всякий раз, когда упоминается "подсвечивание", подразумевается, что Ваш дисплей может оперировать различными видео атрибутами для вывода текста различной интенсивностью, в инверсном изображении, с подчеркиванием или с каким-либо другим выделением. Если это не так, то Вам следует просто игнорировать всякое упоминание о "подсвечивании".

Главное меню показывает команды, доступные при работе с Турбо-Паскалем. Команды выполняются нажатием соответствующей (подсвеченной) буквы. Клавишу RETURN нажимать не нужно: команда выполняется немедленно. При работе с системой меню может исчезать с экрана, однако оно легко восстанавливается путем ввода

любой "недопустимой" команды, т.е. нажатием любой клавиши, которой не соответствует ни одна команда (например, RETURN или ПРОБЕЛ).

Последующие разделы посвящены подробному описанию всех команд.

**Выбор текущего дисковода**

Команда "L" используется для смены текущего дисковода. Если Вы нажмете "L", появится запрос:

New drive: \_

приглашающий ввести имя нового устройства от A до P, за которым может следовать двоеточие. После этого нажмите RETURN. Если Вы не хотите менять текущий дисковод, просто нажмите RETURN.

Команда L выполняет программный сброс дисковой системы, даже если Вы не изменяли дисковод, и должна использоваться всегда, когда Вы меняете дискету во избежание фатальных ошибок записи. Новое выбранное устройство не сразу показывается в меню (меню автоматически не обновляется). Нажмите, например, ПРОБЕЛ для вывода меню, где будет показан новый текущий дисковод.

**Выбор рабочего файла**

Команда "W" используется для выбора рабочего файла, т.е. файла, с которым работают команды "E", "C", "R", "X" и "S" (редактировать, компилировать, выполнить и сохранить). После ввода команды "W" на экране появится:

Work file name: \_

Здесь Вы можете ввести любое допустимое название файла, состоящее из имени (1..8 знаков), необязательной точки и необязательного типа файла (1..3 знака): FILENAME.TYP .

Если ввести имя файла без точки и типа, то по умолчанию подразумевается тип .PAS, который и добавляется к имени. Можно явно указать имя файла без типа, если после имени поставить точку, а тип опустить.

Примеры:

--------

PROGRAM станет PROGRAM.PAS

PROGRAM. не изменится

PROGRAM.FIL не изменится

Следует избегать употребления типов файлов BAK, CHN, COM, поскольку TURBO использует их для специальных целей.

Когда рабочий файл определен, он считывается с дискеты, если он там имеется. Если он отсутствует на дискете, появляется сообщение:

New file

Если Вы редактировали другой файл, но не сохранили его (не записали на диск), то появится сообщение:

Workfile X:FILENAME.TYP not saved. Save (Y/N)? \_

Оно предупредит Вас о том, что Вы собираетесь загрузить новый файл на место старого и уничтожить его. Ответьте "Y" для сохранения старого файла или "N", если запись не нужна.

Новый рабочий файл будет показан в меню, когда оно будет обновлено, например, нажатием клавиши ПРОБЕЛ.

**Выбор главного файла**

Команда "M" может быть использована для определения главного файла при работе с программами, содержащими директиву компилятора $I для включения файлов в исходный текст. Главным файлом должен быть файл, с которого начинается компиляция, то есть файл, содержащий директивы включения $I. Рабочий файл Вы можете определить отличным от главного файла и таким образом редактировать включаемые файлы, оставляя главный файл неизменным.

Когда начинается компиляция и рабочий файл отличается от главного файла, текущий файл автоматически сохраняется, а главный файл загружается в память. Если во время компиляции обнаружена ошибка, файл, содержащий ошибку (это или главный, или включаемый файл) автоматически становится рабочим файлом, который затем может быть отредактирован. После того, как ошибка будет устранена, и компиляция начнется вновь, исправленный рабочий файл автоматически записывается на диск, а главный файл снова загружается в память.

Имя главного файла вводится так же, как и имя рабочего файла.

**Команда запуска редактора**

Команда "E" используется для запуска встроенного редактора и редактирования рабочего файла. Если рабочий файл не указан, то система сначала требует указать его. Затем меню исчезнет, и начнет работать редактор. Более подробно о работе редактора можно узнать в последующих разделах этой главы, посвященных этой теме.

В то время, как для компиляции и выполнения программ систему Турбо-Паскаль можно использовать без настройки (инсталляции), использование редактора требует, чтобы терминал был инсталлирован.

**Команда запуска компилятора**

Команда "C" используется для запуска компилятора. Если не определен главный файл, будет компилироваться рабочий, иначе будет компилироваться главный файл. В этом последнем случае, если рабочий файл редактировался, система спрашивает, нужно ли записать его на диск перед тем, как загрузить и начать компиляцию главного файла. Компиляция может быть прервана в любой момент нажатием любой клавиши.

Результатом компиляции может быть либо программа, расположенная в оперативной памяти, либо программа в COM-файле, либо программа в CHN-файле. Соответствующий выбор делается в меню "Опции компилятора" (см. Рис.1-5 ниже). По умолчанию программа будет размещена в памяти.

**Команда запуска на выполнение**

Команда запуска "R" используется для запуска программ, размещенных в памяти или в COM-файле (если действует ключ "C" из меню опций компилятора). Если скомпилированная программа уже находится в памяти, она сразу начнет выполняться, если нет, сначала автоматически начнется компиляция.

**Команда сохранения рабочего файла**

Команда "S" используется для сохранения текущего рабочего файла на диске. Старая версия этого файла, если она имеется, переименовывается, получая тип .BAK, а новая версия записывается под старым именем.

**Команда "выполнить"**

Команда "X" (eXecute) дает возможность выполнять другие COM-программы из Турбо-системы, например, программы копирования и т.д. Если Вы введете "X", система ответит:

Program: \_

Вы можете ввести имя любой программы, которая будет загружена и выполнена. По окончании ее выполнения управление будет возвращено Турбо-системе, которая ответит:

> .

**Команда вывода оглавления**

Команда "D" выводит оглавление текущего диска и информацию о количестве оставшегося на нем свободного пространства. После ввода "D" система выдает запрос:

Dir mask: \_

Вы можете ввести имя диска (за которым должно следовать двоеточие), а также маску, определяющую имена файлов, в которой содержатся обобщающие символы "\*" и "?". Если же просто нажать RETURN, то будет выведено полное оглавление текущего диска.

**Команда окончания работы**

Команда "Q" используется для выхода из Турбо-системы. Если рабочий файл подвергался редактированию, система спросит, желаете ли Вы сохранить его до выхода.

**Опции компилятора**

По команде "O" на экран выводится меню, которое содержит набор подразумеваемых по умолчанию значений и режимов компилятора и позволяет изменить эти значения и режимы. Предусмотрена также вспомогательная функция поиска ошибок времени выполнения в программах, скомпилированных в файлы объектных кодов (.COM и .CHN).

***compile -> (M)emory***

***(C)om-file***

***c(H)n-file***

***(F)ind run-time error (Q)uit***

Аналогично командам из главного меню каждой команде в меню опций компилятора соответствует одна из "подсвеченных" клавиш.

**Команды M,C,H**

Команды "M","C","H" позволяют выбрать режим работы компилятора, т.е. то место, куда он должен поместить код, являющийся результатом компиляции.

Режим "M" (Memory - память) подразумевается по умолчанию. Сформированный код записывается в оперативную память и остается там готовым к исполнению по команде из главного меню "R" (выполнить).

По команде "C" устанавливается режим генерации COM-файла, а стрелка в меню перемещается на соответствующую строку. Готовый код записывается в файл с тем же именем, что и рабочий файл (или главный, если он указан), но с типом .COM. Этот файл содержит код программы и библиотеку периода выполнения Паскаля и может быть впоследствии запущен посредством ввода его имени с клавиатуры.

По команде "H" устанавливается режим генерации CHN-файла, а стрелка в меню также перемещается на соответствующую строку. Готовый код записывается в файл с тем же именем, но с типом .CHN . Этот файл содержит код программы, но не имеет библиотеки периода выполнения Паскаля и должен активизироваться из другой Паскаль-программы с помощью процедуры CHAIN.

При выборе режимов COM и CHN на экране появляются две дополнительные строки:

***(S)tart address: 20E2 (min 20E2)***

***(E)nd address: C942 (max CC06)***

**Начальный адрес**

Команда "S" задает адрес первого байта кода программы в шестнадцатеричном формате. Обычно этот адрес равен конечному адресу библиотеки Паскаля плюс 1, однако его можно увеличить, если, например, Вы хотите оставить место для абсолютных переменных, которые должны разделяться несколькими CHAIN-программами.

После ввода команды "S" система запрашивает новый адрес. Если при этом нажать просто клавишу RETURN, то будет принято минимальное значение.

**Конечный адрес**

Команда "E" задает максимальный доступный программе адрес в шестнадцатеричном формате. Значение в круглых скобках указывает вершину свободной оперативной памяти (TPA), т.е. адрес BDOS минус 1. По умолчанию подразумевается несколько меньшее значение для размещения загрузчика, который располагается как раз под BDOS при исполнении программ из Турбо-системы.

Введя команду "E", Вы получаете от системы запрос на ввод конечного адреса. Если при этом просто нажать клавишу RETURN, то принимается значение, подразумеваемое по умолчанию. Если установить конечный адрес больше этой величины, то получающиеся в результате такой установки программы не смогут запускаться из Турбо-системы, поскольку они будут затирать Турбо-загрузчик.

**Поиск ошибок периода выполнения**

Если при прогоне программы, записанной после компиляции в память, возникает ошибка периода выполнения, то включается редактор, и ошибка автоматически предъявляется пользователю. Очевидно, что такое невозможно для программы, записанной в COM-или CHN-файл. В этом случае ошибка периода выполнения порождает вывод сообщения, в котором указывается код ошибки и значение счетчика адреса (PC) в момент возникновения ошибки:

***Run-time error 1, PC=1B56***

***Program aborted***

Чтобы отыскать то место в исходном тексте, где возникла ошибка (точнее, то место в тексте, которое соответствует ошибке в исполняемом коде), нужно ввести команду "F" в меню опций. После этого система выведет подсказку для ввода адреса и тогда нужно будет ввести адрес из сообщения об ошибке:

***Enter PC: 1B56***

Теперь система уже в состоянии найти нужное место в исходном тексте, и ошибка предъявляется пользователю точно также, как если бы она возникла при исполнении программы, записанной после компиляции непосредственно в память.

Коды ошибок периода выполнения и ошибок ввода/вывода, а также соответствующая им расшифровка приведены в приложениях F и G.

**Выход из меню опций**

Команда "Q" служит для выхода из меню опций компилятора и возврата в главное меню Турбо-системы.

**Редактор системы Турбо**

Встроенный редактор - это экранный редактор, специально предназначенный для создания текстов программ. Если Вы знакомы с редактором WordStar, Вам не нужны дальнейшие пояснения по работе с Турбо-редактором, поскольку стандартные команды полностью идентичны известным Вам командам WordStar. Имеются некоторые незначительные отличия; кроме того, Турбо-редактор имеет ряд расширений: они обсуждаются в разделе этой главы "Сравнение редакторов Турбо и WordStar".

Работать с редактором системы Турбо предельно просто: когда Вы определите рабочий файл и нажали клавишу "E", меню исчезает, и редактор запускается. Если рабочий файл уже существует на текущем диске, то он загружается, и на экране появляется первая страница текста. Если речь идет о новом файле, то экран остается пустым, за исключением СТРОКИ СОСТОЯНИЯ в верхней его части.

Чтобы выйти из редактора и вернуться в меню, нужно нажать комбинацию клавиш CTRL-K-D; ниже мы рассмотрим это подробнее.

Текст вводится с клавиатуры точно так, как на пишущей машинке. Чтобы окончить строку, нажмите клавишу RETURN. Когда Вы заполните весь экран, верхние строки будут исчезать, освобождая место для новых. Вы можете вернуться на страницу как вперед, так и назад с помощью команд редактирования, описанных ниже.

Рассмотрим сначала структуру строки состояния, расположенную в верхней части экрана.

**Строка состояния**

Первая строка экрана - это строка состояния, содержащая следующую информацию:

Line N Col N Insert Indent X:FILENAME.TYP

Здесь:

Line N Показывает номер строки, в которой находится курсор, начиная от начала файла.

Col N Показывает номер столбца, в котором находится курсор, считая от начала строки.

Insert Показывает, что включен режим ВСТАВКИ, т.е. вводимые с клавиатуры символы будут вставлены в позицию курсора, а имеющийся справа от курсора текст будет сдвигаться вправо по мере ввода нового текста. Использование команды "Вкл/выкл режим вставки" (CTRL-V) приводит к появлению вместо "Insert" сообщения "Overwrite" (ЗАМЕНА), при этом вводимый с клавиатуры текст будет замещать символы под курсором, а не отодвигать их.

Indent Показывает, что включен режим автоматического отступа. Он может быть выключен командой "Вкл/выкл автоотступ" (CTRL-Q-I).

X:FILENAME.TYP Дисковод, имя и тип редактируемого файла.

**Команды редактирования**

Как отмечалось раньше, текст вводится так же, как с помощью пишущей машинки, однако, это компьютерный редактор текстов, который предлагает Вам средства редактирования для выполнения манипуляций над текстом намного проще, чем на бумаге.

Турбо-редактор содержит 45 команд редактирования для перемещения курсора, поиска и замены строк и т.д. Эти команды могут быть разделены на 4 группы:

- команды перемещения курсора,

- команды вставки и удаления,

- блочные команды,

- прочие команды.

В каждую из этих групп входят логически взаимосвязанные команды, которые описываются в последующих разделах. В таблице 1-1 содержится обзор всех допустимых команд.

В ситуации, подобной нашей, наилучшим способом обучения было бы как можно быстрее приступить к работе. Поэтому запускайте Турбо, укажите одну из демонстрационных Паскаль-программ в качестве рабочего файла и введите команду "E", чтобы начать редактирование. Затем, по мере дальнейшего чтения, пробуйте различные команды.

Не отступайте, даже если поначалу Вам это покажется несколько трудным. То, что фирма BORLAND приняла решение сделать редактор системы Турбо совместимым с WordStar, не является простым совпадением, ибо логика команд, однажды изученная, столь глубоко проникает в Вас, что редактор практически превращается в продолжение Вашего интеллекта. Это Вам может подтвердить всякий, кто написал мегабайты текста с помощью этого редактора.

**Обзор команд редактирования**

КОМАНДЫ ПЕРЕМЕЩЕНИЯ КУРСОРА

|  |  |
| --- | --- |
| На символ влево | В начало экрана |
| На символ вправо | В конец экрана |
| На слово влево | В начало текста |
| На слово вправо | В конец текста |
| На строку вверх | В начало строки |
| На строку вниз | В конец строки |
| Скроллинг вверх | В начало блока |
| Скроллинг вниз | В конец блока |
| На страницу вверх | В последнюю позицию курсора |
| На страницу вниз |  |

КОМАНДЫ ВСТАВКИ И УДАЛЕНИЯ

|  |  |
| --- | --- |
| Вкл/выкл режим вставки | Уничтожить слово справа |
| Вставить строку | Уничтожить символ под курсором |
| Удалить строку | Уничтожить символ слева |
| Удалить символы до конца строки |  |

БЛОЧНЫЕ КОМАНДЫ ПРОЧИЕ КОМАНДЫ

|  |  |
| --- | --- |
| Пометить начало блока | Окончить редактирование |
| Пометить конец блока | Табуляция |
| Пометить слово | Вкл/выкл автоотступ |
| Скопировать блок | Восстановить строку |
| Переместить блок | Найти |
| Уничтожить блок | Найти и заменить |
| Прочитать блок с диска | Повторить последний поиск |
| Записать блок на диск | Префикс управляющих символов |
| Скрыть/показать блок |  |

Каждое из последующих описаний содержит заголовок, определяющий команду, и подразумеваемые по умолчанию нажатия клавиш, которые активизируют данную команду.

В средней части каждого описания оставлено место, где можно пометить, какие клавиши используются на Вашем терминале, если они отличаются от указанных. Если у Вас на клавиатуре есть клавиши перемещения курсора и специализированные клавиши текстообработки (вставить, удалить и т.д.), то было бы удобно использовать именно их.

**Заметка об управляющих символах**

Все команды предполагают использование управляющих символов. Управляющие символы - это специальные символы, генерируемые Вашей клавиатурой, если Вы будете удерживать в нажатом состоянии клавишу CTRL (УПР) и нажмете любую клавишу от A до Z,

(более того, даже клавиши [ ] ^ \_ @ генерируют управляющие символы таким образом).

Клавиша CTRL работает подобно клавише SHIFT: если при нажатой клавише SHIFT нажать A, то клавиатура сгенерирует прописную букву A; если же при нажатой клавише CTRL нажать A, то клавиатура сгенерирует CTRL-A - управляющий символ.

**Начиная работу, нужно знать, как правильно ее закончить**

Команда выхода из редактора будет описана далее, однако Вам может оказаться полезным узнать уже сейчас, что команда CTRL-K-D реализует выход из редактора из возвращает Вас в главное меню. Эта команда не выполняет автоматическую запись файла на диск; запись файла на диск следует делать явным образом по команде "S" из главного меню (Save - сохранить).

**Основные команды перемещения курсора**

Самое важное при изучении редактора - это научиться перемещать курсор по экрану. Редактор системы Турбо использует для этой цели специальную группу управляющих символов, а именно управляющие символы CTRL-A, CTRL-S, CTRL-D, CTRL-F, CTRL-E, CTRL-R, CTRL-X и CTRL-C. Эти символы используются потому, что на клавиатуре системы "QWERTY" они находятся вблизи от клавиши CTRL и их расположение как бы логически соответствует их назначению. Для клавиатуры с русскими буквами системы "ЙЦУКЕНГ" пользоваться такими комбинациями неудобно, поэтому, если клавиши управления курсором генерируют другие управляющие символы, имеет смысл запустить программу TINST для перенастройки команд встроенного редактора под Вашу клавиатуру.

**На символ влево CTRL-S**

Перемещает курсор на один символ влево без уничтожения находящегося там символа. Клавиша BACKSPACE может быть инсталлирована для получения того же действия. Если курсор находится в начале строки, он не перемещается.

**На символ вправо CTRL-D**

Перемещает курсор на один символ вправо, не уничтожая находящийся там символ. Если курсор находится в крайней правой позиции экрана, содержимое экрана смещается влево, пока курсор не окажется в крайней правой позиции строки, а именно в столбце 128, где и остановится.

**На слово влево CTRL-A**

Перемещает курсор влево до начала слова. Слово определяется как последовательность символов, разделенных одним из следующих символов: ПРОБЕЛ < > , ; . ( ) [ ] ^ ' \* + - / $. Эта команда может переводить курсор на другую строку.

**На слово вправо CTRL-F**

Перемещает курсор вправо до конца слова. Может передвинуть кур-

сор в другую строку.

**На строку вверх CTRL-E**

Перемещает курсор на строку вверх. Если курсор находится в верхней строке экрана, содержимое экрана перемещается (скроллируется) на одну строку вниз.

**На строку вниз CTRL-X**

Перемещает курсор на одну строку вниз. Если курсор находился в одной из двух последних строк экрана, содержимое экрана перемещается (скроллируется) на одну строку вверх.

**Скроллинг экрана вверх CTRL-W**

Вызывает перемещение на одну строку текста по направлению к его началу; содержимое экрана сдвигается вниз.

**Скроллинг экрана вниз CTRL-Z**

Вызывает перемещение на одну строку текста по направлению к его концу; содержимое экрана сдвигается вверх.

**На предыдущую страницу CTRL-R**

Перемещает курсор на одну страницу вверх с перекрытием на одну строку, т.е. курсор перемещается на полный экран без одной строки.

**На следующую страницу CTRL-C**

Перемещает курсор на одну страницу вниз с перекрытием на одну строку.

**Расширенные команды перемещения курсора**

Команды, которые мы обсудили выше, позволят Вам перемещаться по тексту Вашей программы. После того, как Вы овладеете ими, у Вас может появиться желание перемещать курсор по тексту немного быстрее. Редактор системы Турбо поддерживает шесть команд для быстрого (Quickly) перемещения в начало или конец строки, в начало или конец текста, а также команды перехода в последнюю позицию курсора.

Для запуска этих команд требуется вводить два знака: сначала префикс CTRL-Q, а затем один из следующих управляющих символов: S, D, E, X, R и C (они повторяют уже знакомые нам управляющие символы перемещения курсора).

**В начало строки CTRL-Q-S**

Перемещает курсор к левому краю экрана, т.е. в столбец 1.

**В конец строки CTRL-Q-D**

Перемещает курсор в конец строки, т.е. в позицию, следующую за последним символом в строке. Хвостовые пробелы для экономии памяти удаляются всегда во всех строках.

**В начало экрана CTRL-Q-E**

Перемещает курсор в начало экрана.

**В конец экрана CTRL-Q-X**

Перемещает курсор в конец экрана.

**В начало файла CTRL-Q-R**

Перемещает курсор на первый символ текста.

**В конец файла CTRL-Q-C**

Устанавливает курсор за последним символом текста.

**В начало блока CTRL-Q-B**

Перемещает курсор в позицию, помеченную маркером начала блока, который устанавливается командой CTRL-K-B (отсюда и Q-B). Эта команда срабатывает, даже если блок не отображается (см. далее команду "Скрыть/показать блок"), или если не установлен маркер конца блока.

**В конец блока CTRL-Q-K**

Перемещает курсор в позицию, помеченную маркером конца блока, который устанавливается командой CTRL-K-K (отсюда и Q-K). Команда срабатывает, даже если блок не отображается (см. далее команду "Скрыть/показать блок"), или не установлен маркер начала блока.

**В последнюю позицию курсора CTRL-Q-P**

Перемещает курсор в последнее его положение после операций сохранения файла, поиска, замены.

**Команды вставки и удаления**

Эти команды позволяют вставлять и уничтожать символы, слова и строки. Они подразделяются на три группы: одна команда управления режимом ввода текста, несколько простых команд и одна расширенная команда.

Отметим, что редактор системы Турбо позволяет вам "пойти на попятную" при корректировке текста: отменить все исправления текущей строки, пока Вы не вышли за ее пределы. Соответствующая команда (CTRL-Q-L) описана далее.

**Вкл/выкл режим вставки CTRL-V**

При вводе текста может быть установлен один из двух режимов: ВСТАВКА или ЗАМЕНА. При запуске редактора по умолчанию устанавливается режим вставки; он позволяет вставлять новый текст в уже существующий. Имеющийся справа от курсора текст смещается вправо по мере ввода нового текста.

Режим замены можно включить при необходимости заменить старый текст новым. Вводимые символы замещают символы, которые были расположены в позиции курсора.

Эти режимы переключаются командой "Вкл/выкл режим вставки" CTRL-V, а текущий режим отображается в строке состояния в верхней части экрана.

**Удалить символ слева DEL**

Уничтожает символ слева от курсора. Все символы справа от курсора и сам курсор сдвигаются на одну позицию влево. Клавиша BACKSPACE, которая обычно продвигает курсор назад (влево), не стирая текст, подобно CTRL-S, может быть инсталлирована для выполнения этой функции, если эта клавиша более удобно расположена на Вашей клавиатуре, или если на Вашей клавиатуре вообще отсутствует клавиша DEL. Эта команда позволяет перейти на предыдущую строку, если курсор находился в начале строки, и может использоваться для уничтожения разделителя строк.

**Удалить символ в позиции курсора CTRL-G**

Удаляет символ в позиции курсора и сдвигает все символы справа от курсора на одну позицию влево. Действие этой команды не распространяется далее признака конца строки.

**Удалить слово справа CTRL-T**

Уничтожает слово справа от курсора. Слово определяется как последовательность символов, ограниченная одним из следующих разделителей: ПРОБЕЛ < > , ; . [ ] ( ) ' " + - / $ . Эта команда может действовать на следующую строку.

**Вставить строку CTRL-N**

Вставляет разделитель строк в позицию курсора. Курсор не перемещается.

**Удалить строку CTRL-Y**

Удаляет строку, содержащую курсор и передвигает все нижележащие строки на одну строку вверх. Курсор перемещается к левому краю экрана. Удаленную строку восстановить невозможно, поэтому будьте внимательны.

**Уничтожить до конца строки CTRL-Q-Y**

Уничтожает весь текст от позиции курсора до конца текущей строки.

**Блочные команды**

Все блочные команды - это расширенные команды (т.е. в стандартном определении каждой из них по два символа), и Вы можете пропустить их при первом знакомстве с редактором. В дальнейшем, когда у Вас возникнет потребность перемещать, удалять или копировать целые куски текста, Вы сможете вернуться к этому разделу.

А пока проявим упорство и обсудим использование блоков текста.

Блоком текста считается произвольный кусок текста - от одиночного знака до нескольких страниц. Блок помечается установкой МАРКЕРА НАЧАЛА БЛОКА на первом знаке и МАРКЕРА КОНЦА БЛОКА за последним знаком требуемого участка текста. Помеченный таким образом блок может быть скопирован, перемещен, удален и записан в файл. Имеется команда чтения файла с диска в текст как блока, а также специальная команда, помечающая одиночное слово как блок.

**Пометить начало блока CTRL-K-B**

Эта команда отмечает начало блока. Маркер сам по себе невидим на экране, и блок становится визуально отмеченным лишь тогда, когда устанавливается маркер конца блока, причем только при том условии, что экран инсталлирован для того или иного вида подсвечивания. Но даже если блок не отмечен визуально, его внутренняя маркировка сохраняется и позволяет выполнять над ним манипуляции.

**Пометить конец блока CTRL-K-K**

Эта команда отмечает конец блока. Как и в предыдущем случае, сам маркер на экране невидим, а маркировка блока становится видимой лишь в том случае, если также установлен маркер начала блока.

**Пометить одиночное слово CTRL-K-T**

Эта команда отмечает одиночное слово как блок, размещая соответствующим образом маркеры начала и конца блока. Если курсор расположен внутри слова, будет отмечено это слово, иначе будет помечено слово слева от курсора. Определение слова такое же, как и ранее.

**Скрыть/показать блок CTRL-K-H**

Эта команда вызывает альтернативное включение и выключение визуальной маркировки блока (бледный текст, если Ваш терминал это позволяет). Команды манипуляции с блоками работают только при показанном блоке. Команды перемещения курсора в начало или конец блока работают в любом случае.

**Скопировать блок CTRL-K-C**

Эта команда размещает копию предварительно помеченного блока, начиная с текущего положения курсора. Сам блок остается неизменным, однако маркеры блока перемещаются на новую копию блока. Если помеченного блока нет, команда игнорируется и выдается сообщение об ошибке.

**Переместить блок CTRL-K-V**

Эта команда перемещает предварительно помеченный блок с его места в позицию курсора. Блок исчезает со старого места, и маркеры блока отмечают его новое положение. Если отмеченного блока нет, команда игнорируется, и выдается сообщение об ошибке.

**Удалить блок CTRL-K-Y**

Эта команда удаляет предварительно помеченный блок. Невозможно восстановить удаленный блок, поэтому будьте внимательны.

**Прочитать блок с диска CTRL-K-R**

Эта команда используется для чтения файла в текущий текст в позицию курсора, как если бы это был перемещаемый или копируемый блок. Считанный блок помечается маркерами начала и конца блока. После ввода этой команды редактор запрашивает имя файла. Им может быть любое допустимое имя. Если не введен тип файла, по умолчанию предполагается .PAS. Файл без типа вводится как имя, за которым следует точка.

**Записать блок на диск CTRL-K-W**

Эта команда используется для записи предварительно помеченного блока в файл. Блок не изменяется и маркеры остаются на местах. После ввода этой команды редактор запрашивает имя файла для записи. Если файл с введенным именем уже существует, выводится предупреждение о том, что файл будет перезаписан. Если нет помеченного блока, команда игнорируется, и выдается сообщение об ошибке. Именем файла для записи может быть любое допустимое имя файла. Если тип файла не указан, то система автоматически подставляет тип .PAS . Файл без типа вводится как имя, за которым следует точка. Избегайте пользоваться типами .BAK, .CHN, .COM, поскольку эти типы используются системой Турбо для своих нужд.

**Прочие команды редактирования**

В этом разделе собраны команды, логически не входящие ни в одну из ранее описанных групп, однако, они очень важны, в особенности первая из них.

**Закончить редактирование CTRL-K-D**

Эта команда вызывает окончание редактирования и возврат к главному меню. Редактирование выполняется исключительно в памяти и никак не затрагивает файлы на диске. Сохранить отредактированный файл можно командой "S" главного меню или автоматически в

связи с компиляцией или определением нового рабочего файла.

**Табуляция CTRL-I**

Турбо-редактор не имеет фиксированных позиций табуляции. Вместо этого позиции табуляции автоматически устанавливаются на начало каждого слова в строке, содержащей курсор. Это дает возможность очень удобной автоматической табуляции, особенно полезной при вводе текстов программ, если Вы часто используете выделение отступом объявлений переменных, циклов и т.д. Помните, что Паскаль позволяет получать очень красивые распечатки исходных текстов - пользуйтесь этим не только для красоты, но, что более важно, для получения удобочитаемых программ, особенно если их придется модифицировать спустя длительное время.

**Вкл/выкл автоотступа CTRL-Q-I**

Если этот режим включен, отступ текущей строки будет повторен в каждой следующей строке, т.е. когда Вы нажмете RETURN, курсор не вернется в первую позицию следующей строки, а сразу установится в позицию под началом только что оконченной строки. Если Вы захотите изменить отступ, используйте команды перемещения курсора вправо или влево для выбора новой позиции. Когда этот режим включен, строка состояния содержит слово "Indent". По умолчанию режим включен.

**Восстановить строку CTRL-Q-L**

Эта команда дат возможность аннулировать все изменения, сделанные в строке, до тех пор, пока Вы ее не покинули. Когда курсор попадает на строку, она запоминается, и по этой команде может быть восстановлена. Но как только курсор покидает строку, отредактированная строка записывается в буфер текста и уже не может быть восстановлена. По этой причине не может быть восстановлена строка, удаленная по команде "Удалить строку".

**Поиск CTRL-Q-F**

Эта команда дает возможность найти любую строку длиной до 30 символов. При вводе этой команды строка состояния очищается, и редактор запрашивает образец для поиска. Введите образец, завершив ввод клавишей RETURN. Образец может содержать любые символы, включая управляющие. Управляющие символы вводятся в образец с помощью префикса CTRL-P, за которым следует нужный служебный символ. Например, чтобы ввести в образец CTRL-A, нужно при нажатой клавише CTRL нажать клавиши P и затем A. Так можно включить в образец признак конца строки, введя комбинацию CTRL-M CTRL-J. Отметим, что CTRL-A имеет особое значение: этот символ совпадает с любым знаком и может использоваться в образцах в качестве символа группирования имен.

Образец для поиска может быть отредактирован командами "На символ влево", "На символ вправо", "На слово влево", "На слово вправо". Последняя команда вызывает предыдущий образец, который затем может быть отредактирован. Операция поиска может быть прервана командой прерывания (CTRL-U).

После ввода образца для поиска редактор запрашивает опции поиска. Допустимы следующие опции:

B - поиск назад, т.е. от текущего положения курсора к началу текста;

G - глобальный поиск, т.е. поиск по всему тексту независимо от положения курсора;

n - любое число. Поиск n-го появления образца, считая от текущего положения курсора;

U - игнорировать различие между заглавными и строчными буквами (лат);

W - поиск только целых слов, т.е. пропускать образец, входящий в другие слова.

**Примеры:**

"W" - ищет только целые слова, т.е. образец "термин" не будет найдена в строке "терминал".

"BU" - ищет в обратном направлении, игнорируя разницу между заглавными и строчными буквами. По образцу "Block" будут найдены "blockread", "BLOCKADE" и т.д.

"125" - ищет 125-е появление образца.

Список опций (если он имеется), следует оканчивать нажатием клавиши RETURN, после чего поиск начинается. Если в тексте имеется подстрока, подходящая по образцу, то курсор устанавливается в конец этой строки. Операцию поиска можно повторить с помощью команды "Повторить последнюю операцию поиска" (CTRL-L).

**Поиск и замена CTRL-Q-A**

Эта команда позволяет найти строку длиной до 30 символов и заменить ее другой строкой длиной не более 30 символов. В ответна эту команду строка состояния очищается, и редактор запрашивает образец для поиска. Введите образец, завершив ввод клавишей RETURN. Образец может содержать любые символы, включая управляющие. Управляющие символы вводятся в образец с помощью префикса CTRL-P, за которым следует нужный служебный символ.

Например, чтобы ввести в образец CTRL-A, нужно при нажатой клавише CTRL нажать клавиши P и затем A. Так можно включить в образец признак конца строки, введя комбинацию CTRL-M CTRL-J. Отметим, что CTRL-A имеет особое значение: этот символ совпадает с любым знаком и может использоваться в образцах в качестве символа группирования имен.

Образец поиска может быть отредактирован командами "На символ влево", "На символ вправо", "На слово влево", "На слово вправо". Последняя команда вызывает предыдущий образец, который затем может быть отредактирован. Операция поиска с заменой может быть прервана командой прерывания (CTRL-U).

После того, как образец для поиска введен, редактор запрашивает строку для замены. Вводить можно не более 30 знаков; ввод управляющих символов и редактирование выполняется также, как и для образца поиска, однако CTRL-A здесь уже не имеет никакого специального значения. Если же просто нажать клавишу RETURN, то в процессе замены образец будет заменен на пустую строку, т.е. просто удален.

Далее, наконец, редактор запросит опции. Для поиска и замены имеются следующие опции:

B - поиск и замена в обратном направлении, т.е. поиск и замена от текущего положения курсора до начала текста;

G - глобальный поиск и замена, т.е. поиск и замена по всему тексту независимо от положения курсора;

n - любое число. Поиск и замена n-го вхождения заданной строки, считая от текущего положения курсора;

N - замена без дополнительного запроса, т.е. без остановки и запроса "Replace (Y/N)?" для каждого обнаруженного вхождения строки поиска;

U - игнорировать различие между заглавными и строчными буквами (лат);

W - поиск и замена только целых слов, т.е. пропускать подходящие по образцу последовательности, входящие в другие слова.

**Примеры:**

"N10" - ищет следующие 10 вхождений строки поиска и заменяет их без запроса.

"GWU" - ищет и заменяет целые слова в тексте, игнорируя разницу между заглавными и строчными буквами.

Список опций (если они имеются) следует завершить нажатием клавиши RETURN, после чего и начинается собственно операция. В зависимости от указанных опций может быть найдена последовательность, подходящая по образцу. Когда такая последовательность найдена (и если не указана опция N), курсор устанавливается в конце этой последовательности, а на строке подсказок в верхней части экрана появляется вопрос "Replace (Y/N)?" (Заменить (да/нет)? ). В этом месте можно прервать операцию поиска с заменой путем ввода команды "Прервать" (CTRL-U). Операция поиска с заменой может быть повторена командой "Повторить последнюю операцию поиска" (CTRL-L).

**Повторить последнюю операцию поиска CTRL-L**

Эта команда повторяет последнюю операцию поиска или поиска с заменой без повторного ввода соответствующей информации.

**Префикс управляющих символов CTRL-P**

Турбо-редактор позволяет вводить в текст управляющие символы, предваряя нужный управляющий символ сочетанием CTRL-P, т.е. в начале нажимая CTRL-P, а затем нужный символ. Управляющие символы будут появляться на экране как бледно высвеченные заглавные буквы (или в инверсном изображении, в зависимости от типа Вашего дисплея).

**Прерывание операции CTRL-U**

Команда CTRL-U дает возможность прервать обработку любой команды в тот момент, когда она приостанавливается для ввода, например, когда "Поиск с заменой" запрашивает "Replace (Y/N) ?" или во время ввода образца поиска или имени файла (чтение или запись блока).

**Сравнение редакторов Турбо и WordStar**

Каждый, кто хорошо знаком с редактором WordStar, может заметить, что некоторые команды редактора системы Турбо работают несколько по-другому, и хотя редактор турбо содержит лишь подмножество команд редактора WordStar, ряд специализированных качеств, отсутствующих в WordStar, были добавлены в Турбо-редактор для улучшения работы с исходными текстами программ. Все эти различия обсуждаются в последующих разделах.

**Перемещение курсора**

Команды, управляющие движением курсора, CTRL-S, D, E, X свободно перемещают его по экрану и не переводят его в первую колонку пустых строк. Это не значит, что экран заполняется пробелами, наоборот, все хвостовые пробелы автоматически удаляются. Этот способ передвижения курсора в данном случае очень полезен, например, для выделения сдвигом пар BEGIN/END. CTRL-S и CTRL-D не переводят курсор через разделители строк. Для перехода с одной строки на другую нужно использовать CTRL-E, CTRL-X, CTRL-A, CTRL-F.

**Пометить одиночное слово**

С помощью команды CTRL-K-T можно пометить отдельное слово как блок, что более удобно, чем двухшаговый процесс пометки начала и конца слова отдельно.

**Закончить редактирование**

По команде CTRL-K-D редактирование заканчивается, и управление передается в главное меню. Поскольку редактирование в системе Турбо делается полностью в памяти, эта команда не меняет содержимого файла на диске (в отличие от редактора WordStar). Обновление файла на диске должно быть сделано явно командой "Записать на диск" (Save) из главного меню или автоматически в связи с компиляцией или определением нового рабочего файла. Команда редактора Турбо CTRL-K-D не похожа на команду CTRL-K-Q редактора WordStar (отказ от редактирования), поскольку измененный текст не исчезает; он остается в памяти готовым к записи на диск или компиляции.

**Восстановить строку**

Команда CTRL-Q-L восстанавливает строку к ее первоначальному состоянию до редактирования, если курсор ее не покинул.

**Табуляция**

Фиксированных позиций табуляции нет. Вместо этого позиции табуляции устанавливаются в начале каждого слова той строки, которая располагается непосредственно над строкой, содержащей курсор.

**Автоматический отступ**

Команда CTRL-Q-I включает или выключает режим автоматического отступа.

**ГЛАВА 2.**

**ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА**

**Основные символы**

Основной словарь Турбо-Паскаля содержит символы, подразделяемые на буквы, цифры и специальные символы:

БУКВЫ: A - Z, a - z и знак подчеркивания (\_)

ЦИФРЫ: 0 1 2 3 4 5 6 7 8 9

СПЕЦСИМВОЛЫ: + - \* / = ^ < > ( ) [ ] { } . , : ; ' # $

Не делается различия между строчными и прописными буквами. Некоторые операторы и разделители состоят из двух специальных символов:

Оператор присваивания: :=

Операции отношения: <> <= >=

Отрезок типа: ..

Квадратные скобки: (. и .) можно использовать вместо [ и ]

Комментарии: (\* и \*) можно использовать вместо { и }

**Зарезервированные слова**

Зарезервированные слова являются составной частью Турбо-Паскаля и не могут быть переопределены. Поэтому зарезервированные слова никогда нельзя использовать как идентификаторы, определенные пользователем. Список зарезервированных слов:

|  |  |  |  |
| --- | --- | --- | --- |
| \*avsolute | \*external | nil | \*shr |
| and | file | not | \* string |
| array | for | of | then |
| begin | forward | or | to |
| case | function | packed | type |
| count | goto | procedure | until |
| div | if | program | var |
| do | in | record | while |
| downto | \* inline | repeat | with |
| else | label | set | \*xor |
| end | mod | \*shl |  |

Зарезервированные слова, отмеченные звездочкой, не определены в стандарте Паскаля.

**Стандартные идентификаторы**

В Турбо-Паскале определен ряд стандартных идентификаторов предопределенных типов, констант, переменных, процедур и функций. Каждый из этих идентификаторов может быть переопределен, но при этом теряется возможность пользоваться соответствующим средством.

Следующие стандартные идентификаторы поэтому лучше использовать только для специальных целей:

|  |  |  |  |
| --- | --- | --- | --- |
| Addr | CrtInit | Kbd | Real |
| ArcTan | DelLine | KeyPressed | RecurPtr |
| Assign | Delay | Length | Release |
| Aux | Delete | Ln | Rename |
| AuxInPtr | Dispose | Lo | Reset |
| AuxOutPtr | EOF | LowVideo | Rewrite |
| BDOS | EOLN | Lst | Round |
| BDOSHL | Erase | LstOutPtr | Seek |
| BIOS | Execute | Mark | Sin |
| BIOSHL | Exit | MaxInt | SizeOf |
| BlockRead | Exp | Mem | SeekEof |
| BlockWrite | False | MemAvail | SeekEoln |
| Boolean | FilePos | Move | Sqr |
| BufLen | FileSize | New | Sqrt |
| Byte | FillChar | NormVideo | StackPtr |
| Chain | Flush | Odd | Str |
| Char | Frac | Ord | Succ |
| Chr | FreeMem | Output | Swap |
| Close | GetMem | ParamCount | Text |
| ClrEOL | GotoXY | ParamStr | Trm |
| ClrScr | Halt | Pi | True |
| Con | HeapPtr | Port | Trunc |
| ConInPtr | Hi | Pos | UpCase |
| ConOutPtr | IOresult | Pred | Usr |
| Concat | Input | Ptr | UsrInPtr |
| ConstPtr | InsLine | Random | UsrOutPtr |
| Copy | Insert | Randomize | Val |
| Cos | Int | Read | Write |
| CrtExit | Integer | ReadLn | WriteLn |

**Разделители**

Элементы языка должны быть отделены друг от друга хотя бы одним из следующих разделителей: ПРОБЕЛ, КОНЕЦ СТРОКИ или КОММЕНТАРИЙ.

**Строки программы**

Максимальная длина строки программы - 127 символов; все дальнейшие символы компилятор игнорирует. По этой причине Турбо-редактор допускает только 127 символов в строке, однако исходный текст, подготовленный с помощью других редакторов, может содержать и более длинные строки. Если такой текст читать в Турбо-редакторе, будут автоматически вставлены разделители строк и выдано предупреждение.

**ГЛАВА 3.**

**СТАНДАРТНЫЕ ПРОСТЫЕ ТИПЫ**

Тип данных определяет множество значений, которые может принять переменная. С каждой переменной в программе должен быть связан один и только один тип данных. Хотя типы данных в Турбо-Паскале могут быть достаточно сложными, все они строятся из простых (неструктурированных) типов.

Простой тип может быть, как определен программистом (в этом случае он называется ОБЪЯВЛЕННЫМ ПРОСТЫМ ТИПОМ), так и являться одним из СТАНДАРТНЫХ ПРОСТЫХ ТИПОВ: INTEGER, REAL, BOOLEAN, CHAR или BYTE. Ниже приведено описание этих пяти стандартных простых типов.

**INTEGER - целый**

Это целые числа; в Турбо-Паскале лежат в диапазоне от -32768 до 32767. Число типа INTEGER занимает два байта памяти.

При выполнении целочисленных арифметических операций переполнение не регистрируется. Отметим, в частности, что все промежуточные результаты при вычислении целочисленных выражений должны находиться в пределах диапазона представления целых чисел. Например, выражение 1000\*100/50 не даст в результате 2000, так как умножение вызовет переполнение.

**BYTE - байт**

Этот тип является подмножеством типа INTEGER; диапазон представления 0..255. Следовательно, величины типа BYTE совместимы с величинами типа INTEGER, т.е. всюду, где используется величина типа BYTE, может быть определена величина типа INTEGER, и наоборот, ЗА ИСКЛЮЧЕНИЕМ того случая, когда осуществляется передача этих значений в качестве параметров. Более того, переменные типа BYTE и INTEGER можно смешивать в выражениях, и переменной типа BYTE может быть присвоено значение типа INTEGER. Переменная типа BYTE занимает один байт памяти.

**REAL - вещественный**

Диапазон представления вещественных чисел от 1E-38 до 1E+38 с мантиссой, содержащей 11 значащих цифр. Вещественные числа занимают 6 байт памяти.

Переполнение при арифметических операциях с вещественными числами приводит к прекращению выполнения программы и выдаче сообщения об ошибке периода выполнения. В случае потери значимости результатом является нуль.

Хотя тип REAL относится к стандартным простым типам, имеются следующие различия между ним и остальными простыми типами:

1) Функции PRED и SUCC не могут иметь аргументы типа REAL;

2) Вещественные числа не могут быть использованы как индексы массивов;

3) Вещественные числа не могут использоваться в качестве базового типа множеств;

4) Вещественные числа не могут использоваться для управления операторах FOR и CASE;

5) Задание диапазона вещественных чисел не разрешается.

**BOOLEAN - логический**

Величины типа BOOLEAN могут принимать одно из двух логических значений, обозначаемых стандартными идентификаторами TRUE и FALSE (истина и ложь). По определению FALSE < TRUE. Переменная этого типа занимает один байт памяти.

**CHAR - символьный**

Значение типа CHAR - это один символ из набора символов ASCII. Символы упорядочены в соответствии с их кодами ASCII, например 'A'<'B'. Значения кодов символов лежат в диапазоне от 0 до 255. Переменная типа CHAR занимает один байт в памяти.

**ГЛАВА 4.**

**ЭЛЕМЕНТЫ ЯЗЫКА, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ**

**Идентификаторы**

Идентификаторы используются для обозначения меток, констант, типов, переменных, процедур и функций. Идентификатор состоит из буквы или символа подчеркивания, за которым следует любая комбинация букв, цифр или символов подчеркивания. Длина идентификатора ограничена только длиной строки 127 символов, причем все символы являются значимыми.

**Примеры идентификаторов:**

TURBO

square

persons\_counted

BirthDate

3rdRoot Неверно, начинается с цифры

Two Words Неверно, ПРОБЕЛ недопустим

Вследствие того, что Турбо-Паскаль не различает букв верхнего и нижнего регистров, смешанное использование таких букв в словах, как например BirthDate (дата рождения) не несет никакой функциональной нагрузки. Слово VeryLongIdentifier человеку прочитать легче, чем VERYLONGIDENTIFIER. Такая смесь по возможности будет использоваться во всех идентификаторах этой книги.

**Числа**

Числа - это константы типа REAL или INTEGER. Целые константы - это числа, записанные в десятичной или шестнадцатеричной системе. Шестнадцатеричным константам должен предшествовать знак $, например: $7AB. Диапазон целых чисел в десятичной системе: от -32768 до 32767, в шестнадцатеричной: от $0000 до $FFFF.

**Примеры целых чисел:**

1

12345

-1

$123

$ABC

$123G Неверно, символ G не является 16-ричной цифрой

1.2345 Не является целым, содержит десятичную точку

Диапазон представления вещественных чисел: от 1E-38 до 1E+38 с мантиссой, содержащей 11 значащих цифр. Возможно использование экспоненциальной формы записи, при которой буква E предшествует целому порядку, что означает "умножить на 10 в степени...". Разделители внутри чисел не допускаются. Целые константы можно использовать всюду, где разрешены вещественные константы.

**Примеры вещественных чисел:**

1.0

1234.5678

-0.012

1E6

2E-5

-1.2345678901E+12

1 Допустимо, но не является вещественным,

это константа типа INTEGER

**Строки**

Строковая константа - это последовательность символов, заключенная в одинарные кавычки, например:

'Это строковая константа'

Чтобы поместить одинарную кавычку внутрь строки, нужно записать ее дважды. Строки, содержащие только один символ, относятся к стандартному типу CHAR. Любая строка совместима с типом ARRAY OF CHAR (массив типа CHAR) той же длины. Все строчные константы совместимы со всеми типами STRING (строковый).

**Примеры строковых констант:**

'Turbo'

'ОБ''ЯВЛЕНИЕ'

''''

';'

''

Как видно из второго и третьего примеров, одинарная кавычка внутри строки записывается как две последовательные кавычки. Следовательно, четыре последовательных одинарных кавычки в третьем примере дают строку, состоящую из одной одинарной кавычки.

Последний пример - кавычки, не заключающие ни одного знака, - обозначает ПУСТУЮ СТРОКУ, которая совместима только с типами STRING.

**Управляющие символы**

Турбо-Паскаль позволяет включать в строковые константы управляющие символы. Предусмотрены две формы записи управляющих символов:

1) Знак #, за которым следует целочисленная константа из диапазона 0..255, обозначает символ с соответствующим значением кода ASCII;

2) Знак ^, за которым следует символ, обозначает соответствующий управляющий символ (CTRL).

**Примеры:**

#10 10 в коде ASCII (LF, перевод строки)

#$1B 1BH в коде ASCII (ESC)

^G CTRL-G (BELL, звонок)

^l CTRL-L (FF)

^M CTRL-M (CR, возврат каретки)

^[ CTRL-[ (ESC)

Последовательность управляющих символов может быть записана в строке без разделителей между отдельными символами:

#13#10

#27^U#20

^G^G^G^G

Строки, показанные выше, содержат по два, три и четыре управляющих символа соответственно. Управляющие символы могут смешиваться со строками текста:

'Жду ввода! '^G^G^G' Проснитесь пожалуйста'

#27'U '

'Это другая строка текста '^M^J

Эти строки содержат соответственно по 36, 3 и 27 символов.

**Комментарии**

Комментарии могут быть вставлены всюду, где допускается разделитель. Они отделяются фигурными скобками { }, которые можно заменить символами (\* и \*) .

**Примеры:**

{ Это комментарий }

(\* Здесь допускаются скобки { и } \*)

Фигурные скобки { } внутри других фигурных скобок недопустимы; недопустимы также символы (\* и \*) внутри других таких же символов. Однако, фигурные скобки допустимы внутри комментариев, ограниченных символами (\* и \*), и наоборот, что позволяет убирать в комментарии некоторые части текстов программ, даже если они сами содержат комментарии.

Примечание: При работе на компьютере с кодировкой КОИ7/Н2 (латинские и русские буквы) фигурным скобкам соответствуют русские буквы Ш и Щ. В этом случае для выделения комментариев целесообразно пользоваться заменителями фигурных скобок (\* и \*).

**Директивы компилятора**

Управление рядом возможностей компилятора Турбо-Паскаль осуществляется с помощью директив компилятора. Директивы компилятора вводятся как комментарии с соблюдением специального синтаксиса. Всюду, где разрешены комментарии, допустимы и директивы компилятора.

Директива компилятора состоит из открывающей скобки, за которой непосредственно следует знак $ и первая буква директивы компилятора, или список директив компилятора, разделенных запятыми.

Синтаксис директивы или списка директив зависит от вида директив. Полное описание каждой директивы компилятора находится в Приложении C.

Примеры:

{$I-}

{$I INCLUDE.FIL}

(\*$R-,B+,V-\*)

Заметим, что перед и после знака $ пробелы не допускаются.

**ГЛАВА 5.**

**ЗАГОЛОВОК ПРОГРАММЫ И ПРОГРАММНЫЙ БЛОК**

Программа на языке Паскаль состоит из заголовка программы, за которым следует программный блок. Программный блок, в свою очередь, подразделяется на РАЗДЕЛЫ ОПИСАНИЙ, в которых определяются все локализованные в данной программе объекты, и ОПЕРАТОРНУЮ ЧАСТЬ, в которой определены действия, выполняемые над этими объектами.

**Заголовок программы**

В Турбо-Паскале заголовок программы может быть указан, либо отсутствовать, и не имеет значения для программы. Если он присутствует, он задает имя программы и, возможно, список параметров, через которые программа общается с внешним миром. Этот список содержит список идентификаторов, заключенных в круглые скобки и разделенных запятыми.

Примеры:

program Circles;

program Accountant (Input,Output);

program Writer (Input,Printer);

**Разделы описаний**

Разделы описаний некоторого блока содержат объявления всех идентификаторов, используемых в разделе операторов этого блока (и, возможно, других блоков внутри его). Имеются следующие разделы описаний:

1) раздел описаний меток

2) раздел определения констант

3) раздел определения типов

4) раздел описания переменных

5) раздел описания процедур и функций

Хотя в стандарте Паскаля говорится, что каждая из этих частей может быть либо опущена, либо встречаться только один раз и только в указанном порядке, Турбо-Паскаль допускает произвольное число появлений этих разделов в произвольном порядке, но в пределах описательной части (т.е. вне операторной части).

**Раздел описаний меток**

Любому оператору в программе может предшествовать МЕТКА, дающая возможность прямой передачи управления этому оператору с помощью оператора GOTO. Метка состоит из имени метки, за которым следует двоеточие. Перед использованием метки должны быть описаны в разделе описания меток. Этот раздел начинается зарезервированным словом LABEL, за которым следует список идентификаторов меток, разделенных запятыми, и оканчивающийся символом "**;**".

Пример: label 10,Error,999,Quit;

В отличие от стандарта Паскаля, ограничивающего метки целыми числами длиной не более 4-х цифр, Турбо-Паскаль допускает в качестве меток как числа, так и идентификаторы.

**Раздел определений констант**

В разделе определения констант вводятся идентификаторы как синонимы значений констант. Этот раздел начинается зарезервированным словом CONST, за которым следует список присваиваний значений констант, разделенных символом "**;**". Каждое присваивание состоит из идентификатора, за которым следует знак равенства и константа. Константами могут быть как строки, так и числа.

Пример:

const

Limit=255;

Max=1024;

PassWord='SESAM';

CursHome = ^['V';

Следующие константы в Турбо-Паскале являются предопределенными, т.е. на них можно ссылаться без предварительного определения:

Имя Тип и значение

Pi Real (3.1415926536E+00)

False Boolean (внутреннее значение false)

True Boolean (внутреннее значение true)

MaxInt Integer (32767)

Как описано в главе 13, раздел определения констант может также определять константы с заданным типом.

**Раздел определений типов**

Тип данных в Паскале может быть прямо описан в разделе определения типов, либо описан путем ссылки на идентификатор типа. Предусмотрены несколько стандартных идентификаторов типа; кроме того, программист может создавать свои собственные типы путем использования определения типа. Раздел определения типов начинается зарезервированным словом TYPE, за которым следует одно или несколько присваиваний типов, разделенных символом ";". Каждое присваивание типа состоит из идентификатора типа, за которым следует знак равенства и тип.

Пример:

type

Number = Integer;

Day = (mon,tues,wed,thur,fri,sat,sun);

List = array [1..10] of Real;

Большое количество примеров определений типа Вы найдете в дальнейших разделах.

**Раздел описаний переменных**

Каждая переменная, встречающаяся в программе, должна быть описана (объявлена) перед использованием. Описание должно текстуально предшествовать любому использованию переменной, т.е. переменная должна быть "известна" компилятору перед ее использованием.

Объявление переменных состоит из зарезервированного слова VAR, за которым следует один или более идентификаторов, разделенных запятыми; за ними стоит двоеточие и тип. Это создает новую переменную указанного типа и связывает ее с заданным идентификатором.

Область действия объявления идентификатора - блок, в котором оно помещено, и все блоки внутри этого блока. Заметим, однако, что любой блок внутри некоторого блока может определить другую переменную, используя тот же идентификатор. Эта переменная называется ЛОКАЛЬНОЙ по отношению к блоку, в котором она объявлена (и ко всем блокам внутри него), при этом переменная, объявленная на внешнем уровне (ГЛОБАЛЬНАЯ) становится недоступной.

Пример:

var

Result,Intermediate,SubTotal: Real;

I, J, X, Y: Integer;

Accepted, Valid: Boolean;

Period: Day;

Buffer: array[0..127] of Byte;

**Абсолютные переменные**

Переменные можно объявлять так, чтобы они занимали заданные адреса; такие переменные называются абсолютными.

Это достигается добавлением к описанию переменной зарезервированного слова ABSOLUTE, за которым следует константа типа INTEGER, например:

var

IObyte: byte absolute 3;

CmdLine: string[127] absolute $0080;

Атрибутом ABSOLUTE можно пользоваться также для того, чтобы две переменные начинались с одного и того же адреса - накладывались друг на друга. Когда после слова ABSOLUTE стоит идентификатор переменной или параметра, новая переменная будет начинаться с адреса этой переменной или параметра, например:

var

Str: string[32];

StrLen: Byte absolute Str;

В результате такого объявления переменные Str и StrLen будут иметь один и тот же начальный адрес, а поскольку первый байт строковой переменной содержит длину соответствующей строки, переменная StrLen будет содержать длину переменной Str. В описании абсолютной переменной может быть указан только один идентификатор. Поэтому такая конструкция:

Ident1, Ident2: Integer absolute $80;

является недопустимой.

**Раздел описаний процедур и функций**

Описание процедуры служит для определения процедуры внутри текущей процедуры или программы (см. раздел "Процедуры" в главе 16). Процедура запускается оператором вызова процедуры (см. раздел "Вызов процедуры" в главе 7), и после ее завершения выполнение программы продолжается с оператора, непосредственно следующего за оператором вызова процедуры.

Описание функции служит для определения части программы, которая вычисляет и возвращает определенное значение (см. раздел "Описание функции" в главе 16). Функция запускается, когда ее имя встречается в выражении (см. раздел "Вызовы функций" в главе 6).

**Операторная часть**

Операторная часть является последней частью блока. Она определяет те действия, которые должна выполнять программа. По форме операторная часть имеет вид составного оператора, за которым следует точка или знак ";". Составной оператор состоит из зарезервированного слова BEGIN, за которым следует список операторов, разделенных символом ";", оканчивающийся зарезервированным словом END.

**ГЛАВА 6.**

**ВЫРАЖЕНИЯ**

Выражения - это алгоритмические конструкции, задающие правила вычисления значений. Они состоят из операндов, т.е. переменных, констант и указателей функций в комбинации со знаками операций, описанными ниже.

В этой главе показано, как составлять выражения из величин, принадлежащих к стандартным простым типам INTEGER, REAL, BOOLEAN и CHAR. Выражения, содержащие объявленные простые типы, строковые типы и множества, описаны в соответствующих разделах глав 8, 9 и 12 соответственно.

**Операции**

Операции делятся на 5 категорий согласно приоритета:

1) Унарный минус

2) NOT

3) \* / DIV MOD AND SHL SHR (операции группы умножения)

4) + - OR XOR (операции группы сложения)

5) = <> < > <= >= IN (операции отношений)

Последовательные операции одного приоритета выполняются слева направо. Выражения, заключенные в круглые скобки, вычисляются в первую очередь независимо от предшествующих и последующих операций.

Если оба операнда в операциях групп сложения и умножения имеют тип INTEGER, то и результат будет типа INTEGER. Если хотя бы один из операндов в этих операциях имеет тип REAL, то и результат будет типа REAL.

**Унарный минус**

Унарный минус обозначает отрицание своего операнда, который может иметь тип INTEGER или REAL, например: -A.

**Операция NOT**

Это операция логического отрицания (инверсии) значения своего операнда типа BOOLEAN:

NOT TRUE = FALSE

NOT FALSE = TRUE

Турбо-Паскаль позволяет также применять операцию NOT к операнду типа INTEGER, что вызывает инверсию его внутреннего представления.

Примеры:

NOT 0 = -1

NOT -15 = 14

NOT $2345 = $DCBA

**Операции группы умножения**

|  |  |  |  |
| --- | --- | --- | --- |
| Обозначение | Операция | Тип операндов | Тип результата |
| \* | Умножение | Real | Real |
| \* | Умножение | Integer | Integer |
| \* | Умножение | Real, Integer | Real |
| / | Деление | Real, Integer | Real |
| / | Деление | Integer | Real |
| / | Деление | Real | Real |
| DIV | Деление на цело | Integer | Integer |
| MOD | Взятие остатка | Integer | Integer |
| AND | Арифметич. "И" | Integer | Integer |
| AND | Логическое "И" | Boolean | Boolean |
| SHL | Сдвиг влево | Integer | Integer |
| SHR | Сдвиг вправо | Integer | Integer |

Примеры:

12\*34 = 408

123/4 = 30.75

123 DIV 4 = 30

12 MOD 5 = 2

TRUE AND FALSE = FALSE

12 AND 22 = 4

2 SHL 7 = 256

256 SHR 7 = 2

**Операции группы сложения**

|  |  |  |  |
| --- | --- | --- | --- |
| Обозначение | Операция | Тип операндов | Тип результата |
| + | Сложение | Real | Real |
| + | Сложение | Integer | Integer |
| + | Сложение | Real, Integer | Real |
| - | Вычитание | Real | Real |
| - | Вычитание | Integer | Integer |
| - | Вычитание | Real, Integer | Real |
| OR | Арифм. "ИЛИ" | Integer | Integer |
| OR | Логическое "ИЛИ" | Boolean | Boolean |
| XOR | Арифм. "ИСКЛ.ИЛИ" | Integer | Integer |
| XOR | Логич. "ИСКЛ.ИЛИ" | Boolean | Boolean |

Примеры:

123+456 = 579

456-123.0 = 333.0

TRUE OR FALSE = TRUE

12 OR 22 = 30

TRUE XOR FALSE = TRUE

12 XOR 22 = 26

**Операции отношений**

Операции отношений применимы ко всем стандартным простым типам: REAL, INTEGER, BOOLEAN, CHAR и BYTE. Операнды типа INTEGER, REAL и BYTE могут смешиваться. Тип результата всегда BOOLEAN, т.е. TRUE или FALSE.

**= Равно**

<> Не равно

> Больше чем

< Меньше чем

>= Больше или равно

<= Меньше или равно

Примеры:

A=B TRUE, если A равно B

A<>B TRUE, если A не равно B

A>=B TRUE, если A больше или равно B

**Вызовы функций**

Вызов функции представляет собой идентификатор функции, за которым может следовать список параметров, заключенный в круглые скобки и состоящий из одного или нескольких выражений, разделенных запятыми. Появление в тексте вызова функции приводит к активизации функции с указанным именем. Если указанная функция не является одной из стандартных функций, она должна быть объявлена перед вызовом.

Примеры:

Round (PlotPos)

Writeln(Pi\*(Sqr(R)))

(Max(X,Y) < 25) and (Z > Sqrt(X\*Y))

Volume (Radius,Height)

**ГЛАВА 7.**

**ОПЕРАТОРЫ**

В операторной части определяются действия, которые должны быть выполнены программой (или подпрограммой). Эти действия определяются в форме последовательности ОПЕРАТОРОВ, каждый из которых задает какую-то одну часть общего действия. В этом смысле Паскаль представляет собой язык для последовательного программирования: операторы выполняются последовательно во времени и никогда - параллельно. Операторная часть ограничивается парой зарезервированных слов BEGIN и END, а между ними операторы отделяются друг от друга точкой с запятой. Операторы могут быть ПРОСТЫМИ и СТРУКТУРНЫМИ.

**Простые операторы**

Простые операторы - это операторы, не содержащие внутри себя другие операторы. К этой группе относятся оператор присваивания, оператор вызова процедуры, оператор перехода и пустой оператор.

**Оператор присваивания**

Самым фундаментальным из всех операторов является оператор присваивания. Он указывает, что некоторой переменной присваивается определенное значение. Присваивание состоит из идентификатора переменной, за которым следует знак присваивания “**:=**” и выражение.

Возможно присваивание переменной любого типа (кроме типа FILE) значения выражения только того же типа. Как исключение, если переменная имеет тип REAL, тип выражения может быть INTEGER.

Примеры:

Angle := Angle \* Pi;

AccessOk := False;

Entry := Answer = PassWord;

SpherVol := 4 \* Pi \* R \* R;

**Вызов процедуры**

Оператор вызова процедуры (подпрограммы) служит для активизации предварительно определенной процедуры пользователя или предопределенной стандартной процедуры. Оператор состоит из идентификатора процедуры, за которым может следовать список параметров, который является списком переменных или выражений, разделенных запятыми, и заключен в круглые скобки. Когда в выполняемой программе встречается оператор вызова процедуры, управление передается этой процедуре, при этом ей передаются значения параметров. По окончании процедуры выполнение программы продолжается с оператора, следующего за оператором вызова процедуры.

Примеры:

Find (Name,Address);

Sort (Address);

UpperCase (Text);

UpdateCustFile (CustRecord);

**Оператор перехода**

Оператор перехода состоит из зарезервированного слова GOTO, за которым следует идентификатор метки. Он служит для передачи управления оператору, помеченному указанной меткой. При использовании оператора GOTO необходимо соблюдать следующие правила:

1) Метки должны быть объявлены перед использованием. Это должно делаться в разделе описаний меток того блока, где эти метки используются;

2) Область действия меток ограничена тем блоком, в котором они объявлены. Поэтому невозможно передать управление внутрь процедуры или функции, или из них.

**Пустой оператор**

"Пустой оператор" - это оператор, не содержащий ни одного символа и не выполняющий никаких действий. Он может быть использован там, где синтаксис Паскаля требует наличия оператора, но никакие действия выполнять не нужно.

Примеры:

begin end.

while Answer<>' ' do;

repeat until KeyPressed; { Ожидание нажатия клавиши }

**Структурные операторы**

Структурные операторы - это конструкции, составленные из других операторов, которые должны выполняться последовательно (составные операторы), условно (условные операторы) или многократно (операторы цикла). Обсуждение оператора WITH отложим до главы 11.

**Составной оператор**

Составной оператор используется, если нужно выполнить более, чем один оператор там, где синтаксис Паскаля допускает запись только одного оператора. Составной оператор представляет собой последовательность из любого числа операторов, разделенных символом "**;**" и ограниченных зарезервированными словами **BEGIN** и **END**. Эти операторы будут выполняться в той же последовательности, в которой они записаны.

Пример:

if Small > Big then

begin

Tmp:=Small;

Small:=Big;

Big:=Tmp;

end;

**Условные операторы**

Условный оператор выбирает для выполнения только один из входящих в его состав операторов.

**Оператор IF**

Оператор IF обеспечивает выполнение какого-либо оператора только если некоторое условие (выражение типа BOOLEAN) будет истинно. Если оно ложно, либо этот оператор будет обойден, либо выполнится другой оператор, записанный после зарезервированного слова ELSE. Заметим, что слову ELSE не должен предшествовать символ "**;**".

Двусмысленность понимания следующей синтаксической конструкции:

if Expr1 then

if Expr2 then

Stmt1

else

Stmt2

устраняется путем интерпретации этой конструкции как:

if Expr1 then

begin

if Expr2 then

Stmt1

else

Stmt2

end

т.е. ветвь ELSE соответствует последнему оператору IF, не имеющему ветви ELSE.

Примеры:

if Interest > 25 then

Usury:=true

else

Takeloan:=OK;

if (Entry < 0) or (Entry > 100) then

begin

Write('Диапазон от 1 до 100, пожалуйста повторите: ');

Read(Entry);

end;

**Оператор CASE**

Оператор выбора **CASE** состоит из выражения (селектора) и списка операторов, каждому из которых предшествует метка варианта того же типа, что и селектор. Он вызывает выполнение того из операторов списка, чья метка варианта равна текущему значению селектора. Если ни одна из меток варианта не совпадает с текущим значением селектора, то либо такой оператор выбора эквивалентен пустому оператору, либо выполняются операторы, следующие за зарезервированным словом **ELSE**. Ветвь **ELSE** в операторе выбора является расширением стандарта Паскаля.

Метка варианта состоит из любого числа констант или поддиапазонов, разделенных запятыми; за ними следует двоеточие. Поддиапазон записывается как две константы, разделенные двумя подряд идущими точками. Тип констант должен совпадать с типом селектора. Оператор, следующий за меткой варианта, выполняется, если текущее значение селектора равно одной из констант или лежит внутри одного из поддиапазонов.

Типом селектора может быть любой простой тип, кроме REAL.

Примеры:

case Operator of

'+': Result:=Answer+Result;

'-': Result:=Answer-Result;

'\*': Result:=Answer\*Result;

'/': Result:=Answer/Result;

end;

case Year of

Min..1939: begin

Time:=PreWorldWar2;

Writeln('Мир без войны...');

end;

1946..Max: begin

Time:=PostWorldWar2;

Writeln('Построение нового мира.');

end;

else

begin

Time:=WorldWar2;

Writeln ('Мы во время войны');

end;

end;

**Операторы цикла**

Операторы повторения (цикла) вызывают многократное выполнение некоторых операторов. Если количество повторений известно до начала цикла, то применяется оператор FOR. В других случаях нужно использовать операторы **WHILE** и **REPEAT** (циклы с заранее неизвестным числом повторений).

**Оператор FOR**

Оператор FOR показывает, что входящий в его состав оператор должен многократно выполняться с увеличением или уменьшением значения, присвоенного некоторой переменной, которая называется управляющей переменной или ПАРАМЕТРОМ ЦИКЛА. Если это значение необходимо увеличивать, используется зарезервированное слово TO, а если уменьшать, то слово DOWNTO, после чего указывается конечное значение.

Параметр цикла, его начальное и конечное значения должны быть одного и того же типа. Допускаются все простые типы, кроме REAL.

Если начальное значение больше конечного значения и используется форма TO, или если начальное значение меньше конечного и используется форма DOWNTO, оператор, входящий в состав оператора FOR, не выполняется ни разу.

Примеры:

for i:=2 to 100 do

if A[I] > Max then

Max:=A[I];

for i:=1 to NoOfLines do

begin

Readln (Line);

if Length(Line) < Limit then

ShortLines:=ShortLines+1

else

LongLines:=LongLines+1

end;

Заметим, что оператор, входящий в состав оператора FOR, не должен содержать операторы, изменяющие значения параметра цикла. Если необходимо прекратить цикл до достижения конечного значения, следует использовать оператор GOTO, хотя такие конструкции не рекомендуются: в хорошей программистской практике в такой ситуации пользуются операторами WHILE и REPEAT.

После завершения оператора FOR параметр цикла равен конечному значению, если цикл вообще выполняется, в противном случае параметру цикла не присваивается никаких значений.

**Оператор WHILE**

Выражение, управляющее циклом WHILE, должно быть типа BOOLEAN. Оператор будет выполняться до тех пор, пока выражение истинно. Если его значение перед началом цикла равно FALSE, оператор выполнен не будет.

Примеры:

while Size>1 do

Size:=Sqrt(Size);

while ThisMonth do

begin

ThisMonth:=CurMonth=SampleMonth;

Process; { Дальнейшая обработка }

end;

**Оператор REPEAT**

Выражение, управляющее циклом REPEAT, должно быть типа BOOLEAN. Последовательность операторов между зарезервированными словами REPEAT и UNTIL выполняется многократно до тех пор, пока выражение ложно. В противоположность оператору WHILE оператор REPEAT всегда выполняется хотя бы один раз, так как проверка условия производится по окончании цикла.

Пример:

repeat

Write(^M,'Уничтожить эти данные ? (Y/N): ');

Read(Kbd,Answer);

until UpCase(Answer) in ['Y','N'];

**ГЛАВА 8.**

**ПРОСТЫЕ И ОГРАНИЧЕННЫЕ ТИПЫ**

Основными типами данных в Паскале являются простые типы. Простые типы представляют собой конечное или линейно упорядоченное множество значений. Хотя стандартный тип REAL включен в число простых типов, он не удовлетворяет этому определению. Поэтому тип REAL не может быть использован в некоторых случаях, когда используются другие простые типы.

**Простой тип**

В дополнение к стандартным простым типам (INTEGER, REAL, BOOLEAN, CHAR и BYTE) Паскаль поддерживает простые типы, определенные пользователем. Они также называются объявленными простыми типами. Определение простого типа перечисляет по порядку все его возможные значения. Значения нового типа могут быть представлены идентификаторами и являются константами нового типа.

Примеры:

type

Operator=(Plus,Minus,Multi,Divide);

Day =(Mon,Tues,Wed,Thur,Fri,Sat,Sun);

Month =(Jan,Feb,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec);

Card =(Club,Diamond,Heart,Spade);

Переменные последнего типа Card могут принимать одно из четырех значений, а именно: Club, Diamond, Heart, Spade. Вы уже знакомы со стандартным типом BOOLEAN, который определяется как:

type

boolean=(False,True);

Операции отношений «=» «<>» «>» «<» «>=» «<=» могут быть применимы ко всем простым типам, если оба операнда принадлежат к одному и тому же типу (типы INTEGER и REAL можно смешивать). Порядок значений простого типа является основой для сравнения, т.е. порядок, в котором перечислены значения в определении типа.

Так, для типа Card:

Club < Diamond < Heart < Spade

Определены следующие стандартные функции над аргументами простого типа:

Succ(Diamond) последователь Diamond (Heart)

Pred(Diamond) предшественник Diamond (Club)

Ord (Diamond) порядковый номер Diamond (1, поскольку порядковый номер первого значения простого типа равен 0).

**Ограниченный тип**

Тип может быть определен в виде поддиапазона или отрезка другого, ранее определенного простого типа. Такие типы называют ограниченными типами или отрезками типов. Определение ограниченного типа просто содержит наименьшее и наибольшее значения. Первая константа определяет нижнюю границу и не должна быть больше, чем вторая константа, определяющая верхнюю границу. Отрезки типа REAL не разрешаются.

Примеры:

type

HemiSphere =(North,South,East,West);

World =(East..West);

CompassRange =0..360;

Upper ='A'..'Z';

Lower ='a'..'z';

Degree =(Celc,Fahr,Reom,Kelv);

Wine =(Red,White,Rose,Sparkling);

В этом примере тип WORLD является ограничением простого типа HEMISPHERE, который называется БАЗОВЫМ простым типом. Базовый простой тип для COMPASSRANGE - это тип INTEGER, а для типов UPPER и LOWER - тип CHAR.

Вы уже знакомы со стандартным ограниченным типом BYTE, который определяется как:

type

Byte = 0..255;

Ограниченный тип сохраняет все свойства базового простого типа, будучи ограниченным только в значениях.

Использование объявленных простых типов и ограниченных типов настоятельно рекомендуется для улучшения читабельности программ. Более того, в разделе "Проверка границ" показано, что в текст программы можно включить такие проверки, которые во время выполнения программы контролируют значения, присваиваемые переменным объявленного простого типа или ограниченного типа. Другое преимущество таких типов состоит в том, что они часто позволяют экономить память. Для переменных объявленного простого типа или ограниченного типа, общее число элементов которых не превосходит 256, Турбо-Паскаль выделяет только один байт памяти.

**Преобразование типов**

Функция ORD может быть использована для преобразования простых типов в значения типа INTEGER. Стандарт Паскаля не предусматривает обратный процесс, т.е. преобразование INTEGER в значение простого типа.

В Турбо-Паскале значение простого типа может быть преобразовано в значение другого простого типа с тем же порядковым номером. Для этого нужно записать идентификатор требуемого типа в виде вызова функции с параметром в круглых скобках. Параметром может быть значение любого простого типа, за исключением REAL.

Так, для определений типов из примеров двух предыдущих разделов справедливо следующее:

Integer(Heart)=2

Month(10) =Nov

HemiSphere(2) =East

Upper(14) ='O'

Degree(3) =Kelvin

Char(78) ='N'

Integer('7') =55

**Проверка границ**

Генерация кода, обеспечивающего во время выполнения проверку границ для переменных простого и ограниченного типа, управляется директивой R компилятора. По умолчанию принимается {$R-}, т.е. отсутствие такой проверки. Когда выполняется присваивание переменной простого или ограниченного типа в условиях действия этой директивы: {$R+}, присваиваемое значение проверяется на вхождение в диапазон данного типа. Этим рекомендуется пользоваться до тех пор, пока программа не будет полностью отлажена.

Пример:

program RangeCheck;

type

Digit=0..9;

var

Dig1,Dig2,Dig3: Digit;

begin

Dig1:=5; { Верно }

Dig2:=Dig1+3; { Верно, поскольку DIG1+3<=9 }

Dig3:=47; { Неверно, но не вызывает ошибку }

{$R+} Dig3:=55; { Неверно и приводит к ошибке выполнения }

{$R-} Dig3:=167; { Неверно, но не вызывает ошибку }

end.

**ГЛАВА 9.**

**СТРОКОВЫЙ ТИП**

В Турбо-Паскале предусмотрены удобные строковые типы для обработки строк символов, т.е. последовательностей символов. Строковые типы - это структурные типы, и они во многом похожи на типы ARRAY (см. главу 10). Однако, между ними имеется принципиальное отличие: число символов в строке (т.е. ДЛИНА строки) может меняться динамически от 0 до объявленной верхней границы, в то время как число элементов массива постоянно.

**Определение строкового типа**

Определение строкового типа должно указывать максимальное число символов, которое может содержать строка данного типа, т.е. максимальную длину строки данного типа. Определение состоит из зарезервированного слова STRING, за которым следует максимальная длина строки в квадратных скобках. Длина - это целая константа от 1 до 255. Заметим, что строки не имеют длину по умолчанию: длина всегда должна быть указана.

Пример:

type

FileName = string[14];

ScreenLine = string[80];

Строковая переменная занимает в памяти заданную максимальную длину байт плюс один байт, содержащий ее текущую длину. Отдельные символы в строке индексируются от 1 до длины строки.

**Строковые выражения**

Манипуляции со строками выполняются с помощью СТРОКОВЫХ ВЫРАЖЕНИЙ. Строковые выражения состоят из строковых констант, строковых переменных, указателей функций и операций.

Знак "+" может быть использован для конкатенации (сцепления) строк. Функция CONCAT (см. раздел "Функция CONCAT" в этой главе) выполняет ту же функцию, но знак "+" часто более удобен. Если длина результата более 255, фиксируется ошибка периода выполнения.

Примеры:

'TURBO '+'Pascal' = 'TURBO Pascal'

'123'+'.'+'456' = '123.456'

'A '+'B'+' C '+'D' = 'A B C D'

Операции отношений = <> > < >= <= имеют меньший приоритет, чем операция конкатенации. Когда они применяются к строковым операндам, результат имеет значение типа BOOLEAN (TRUE или FALSE). При сравнении двух строк одиночные символы сравниваются слева направо. Если строки имеют разную длину, но их символы одинаковы, за исключением последних символов более длинной строки, более короткая строка считается меньшей. Строки равны между собой только в том случае, если их длины и содержание одинаковы.

Примеры**:**

'A' < 'B' true

'A' > 'b' false

'2' < '12' false

'TURBO' = 'TURBO' true

'TURBO ' = 'TURBO' false

'Pascal Compiler' < 'Pascal compiler' true

**Строковое присваивание**

Для присваивания строковой переменной значения строкового выражения используется оператор присваивания.

Пример:

Age:='fiftieth';

line:='Many happy returns on your '+Age+' birthday';

Если максимальная длина строковой переменной превышена (в результате присваивания переменной значения, содержащего слишком много символов), лишние символы справа отсекаются. Так, если переменная Age была объявлена как STRING[5], после присваивания она будет содержать только 5 левых символов: 'fifti'.

**Процедуры обработки строк**

В Турбо-Паскале определены следующие стандартные процедуры, работающие со строками:

**Процедура DELETE**

Синтаксис: DELETE(St,Pos,Num);

Эта процедура удаляет подстроку, содержащую Num символов, из строки St, начиная с позиции Pos. Если Pos больше, чем текущая длина строки, ни один символ не удаляется. При попытке уничтожить символы за концом строки (т.е. если Pos+Num превышает длину строки) удаляются только символы, принадлежащие строке. Если Pos не находится в диапазоне 1..255, фиксируется ошибка периода выполнения.

Если St имеет значение 'ABCDEFG', то

DELETE (St,2,4) присвоит St значение 'AFG',

DELETE (St,2,10) присвоит St значение 'A'.

**Процедура INSERT**

Синтаксис: INSERT(Obj,Target,Pos);

Процедура INSERT вставляет строку Obj в строку Target, начиная с позиции Pos. Obj - это строковое выражение, Target - строковая переменная, Pos - это целое выражение. Если Pos превышает длину Target, строка Obj присоединяется справа к строке Target. Если Pos не принадлежит диапазону 1..255, фиксируется ошибка периода выполнения.

Если St имеет значение 'ABCDEFG', то

INSERT ('XX',St,3) присвоит St значение 'ABXXCDEFG'.

**Процедура STR**

Синтаксис: STR(Value,St);

Эта процедура преобразует численное значение Value в строку и запоминает результаты в St. Value - это параметр вывода типа INTEGER или REAL, St - это строковая переменная. Параметры вывода - это выражения со специальными командами форматирования. Подробнее об этом можно узнать из раздела, посвященного процедуре WRITE в главе 14.

Если переменная I имеет значение 1234, то:

STR(I:5,St) присвоит St значение ' 1234'.

Если переменная X имеет значение 2.5E4, то:

STR (X:10:0,St) присвоит St значение ' 2500'.

Функция, использующая процедуру STR, никогда не должна вызываться из выражения внутри операторов WRITE или WRITELN.

**Процедура VAL**

Синтаксис: VAL(St,Var,Code);

Эта процедура преобразует строковое выражение St в целое или вещественное значение (в зависимости от типа переменной Var) и запоминает это значение в переменной Var. St должно быть строкой, содержащей численное значение согласно правилам для числовых констант (см. раздел "Числа" в главе 4). Ни ведущие, ни концевые пробелы не разрешаются. Var должно быть целой или вещественной переменной, и Code должно быть целой переменной.

Если ошибки не обнаружены, переменная Code устанавливается в 0, иначе Code принимает значение позиции первого ошибочного символа, в этом случае значение Var не определено.

Если St имеет значение '234', то:

VAL (St,I,Result) присвоит I значение 234, а Result - значение 0;

Если St имеет значение '12X', то:

VAL (St,I,Result) присвоит I неопределенное значение, а Result - значение 3;

Если St имеет значение '2.5E4' и X имеет тип REAL, то:

VAL (St,X,Result) присвоит X значение 2500, а Result - значение 0.

Функция, использующая процедуру VAL, никогда не должна вызываться в выражении внутри процедур WRITE и WRITELN.

**Функции обработки строк**

В Турбо-Паскале определены следующие стандартные строковые функции:

**Функция COPY**

Синтаксис: COPY(St,Pos,Num)

Эта функция возвращает подстроку, содержащую Num символов строки St, начиная с позиции Pos. St - это строковое выражение, Pos и Num - целые выражения. Если Pos превышает длину строки, возвращается пустая строка. Если делается попытка получить символы за концом строки (т.е. Pos+Num превышает длину строки), возвращаются только символы из строки. Если Pos не находится в диапазоне 1..255, фиксируется ошибка периода выполнения.

Если St имеет значение 'ABCDEFG', то:

COPY (St,3,2) вернет значение 'CD',

COPY (St,4,10) вернет значение 'DEFG'

COPY (St,4,2) вернет значение 'DE'

**Функция CONCAT**

Синтаксис: CONCAT(St1,St2[,..,Stn])

Эта функция возвращает строку, полученную путем сцепления аргументов в том порядке, в котором они указаны. Аргументами может быть любое число строковых выражений St1,St2,..,Stn, разделенных запятыми. Если длина результата превышает 255, фиксируется ошибка периода выполнения. Как было указано в разделе "Строковые выражения" главы 9, операция "+" может быть использована для получения того же результата, что часто более удобно. Функция CONCAT предусмотрена только для совместимости с другими компиляторами с языка Паскаль.

Если St1 имеет значение 'TURBO' и St2 имеет значение 'is fastest', то:

CONCAT(St1,' Pascal ',St2) вернет значение 'TURBO Pascal is fastest'.

**Функция LENGTH**

Синтаксис: LENGTH(St)

Возвращает длину строкового выражения St, т.е. число символов в St. Тип результата - INTEGER.

Если St имеет значение '123456789', то:

LENGTH (St) вернет значение 9.

**Функция POS**

Синтаксис: POS (Src,Trg)

Эта функция "просматривает" строку Trg для поиска первого вхождения подстроки Src в строку Target. Src и Trg – строковые выражения, тип результата - INTEGER. Результатом является целое число, означающее позицию в Trg первого символа искомой подцепочки. Номер позиции первого символа равен 1. Если подстрока не найдена, возвращается значение 0.

Если St имеет значение 'ABCDEFG', то:

POS ('DE',St) вернет 4,

POS ('H',St) вернет 0.

**Строки и символы**

Строковый тип STRING и стандартный простой тип CHAR совместимы. Поэтому всюду, где предполагается строковое значение, вместо него может быть записана величина типа CHAR и наоборот.

Более того, величины строкового и символьного типов можно смешивать в выражениях. Если переменной типа CHAR присваивается строковое значение, длина этой строки должна быть равна 1, иначе возникает ошибка периода выполнения.

Возможен отдельный доступ к символам внутри строки с помощью индексирования. Это достигается путем добавления индексного выражения целого типа, заключенного в квадратные скобки, к имени строковой переменной.

Примеры:

Buffer [5]

Line [Length(Line)-1]

Ord (Line[0])

Вследствие того, что в начале строки по индексу 0 содержится длина строки, справедливо следующее равенство:

Length (Line) = Ord (Line[0]).

Если нулевому символу строки присваивается какое-либо значение, то ответственность за то, чтобы присваиваемое значение не превышало максимальной длины данной строковой переменной, ложится на программиста. Если включен режим проверки диапазонов (директива {$R+} компилятора), генерируется код, который проверяет, что значение индекса строки не превосходит максимальную длину строковой переменной. Однако можно задать индекс, превышающий текущую динамическую длину переменной. При этом читаемые из строки символы будут случайными, а присваивание за пределами текущей длины строки не изменит действительное значение строковой переменной.

**ГЛАВА 10.**

**МАССИВЫ**

Массив - это структурный тип данных, состоящий из фиксированного числа компонент одного и того же типа, называемого базовым типом. К каждой компоненте возможен отдельный доступ при помощи индексов массива. Индексы - это целые выражения, заключенные в квадратные скобки и добавляемые к идентификатору массива. Их тип называется индексным типом.

**Определение массива**

Определение массива состоит из зарезервированного слова ARRAY, за которым следует индексный тип, заключенный в квадратные скобки, затем следует зарезервированное слово OF и базовый тип (тип компонент массива).

Примеры:

type

Day=(Mon,Tue,Wed,Thu,Fri,Sat,Sun);

var

WorkHour: array[1..8] of Integer;

Week : array[1..7] of Day;

type

Players =(Player1,Player2,Player3,Player4);

Hand =(One,Two,Pair,TwoPair,Three,Straight,

Flush,FullHouse,Four,StraightFlush,RSF);

LegalBid=1..200;

Bid =array[Players] of LegalBid;

var

Player: Array[Players] of Hand;

Pot: Bid;

Доступ к компоненте массива осуществляется путем добавления индекса в квадратных скобках к идентификатору массива:

Player [Player3]:=FullHouse;

Pot [Player3]:=100;

Player [Player4]:=Flush;

Pot [Player4]:=50;

Поскольку разрешено присваивание значения одной переменной другой переменной того же типа, массивы можно копировать простым оператором присваивания.

Директива R компилятора управляет генерацией кода, обеспечивающего проверку диапазона индексных выражений массивов во время выполнения программы. По умолчанию этот режим выключен, т.е. {$R-}, и задание {$R+} приводит к проверке принадлежности индексных выражений заданным пределам.

**Многомерные массивы**

Тип компонент массива может быть любым типом данных, в том числе тип компонент может быть другим массивом. Такая структура называется многомерным массивом.

Пример:

type

Card = (Two,Three,Four,Five,Six,Seven,Eight,Nine,

Ten,Knight,Queen,King,Ace);

Suit = (Hearts,Spade,Clubs,Diamonds);

AllCards = array [Suit] of array[1..13] of Card;

var

Deck: AllCards;

Многомерный массив может быть более удобно определен путем указания нескольких индексов:

type

AllCards = array [Suit,1..13] of Card;

Аналогичное сокращение может быть использовано при выборе компоненты массива:

Deck [Hearts,10] эквивалентно Deck [Hearts][10].

Можно, конечно, определять многомерные массивы через предварительно определенные типы массивов.

Пример:

type

Pupils = string[20];

Class = array [1..30] of Pupils;

School = array [1..100] of Class;

var

J,P,Vacant: Integer;

ClassA,ClassB: Class;

NewTownSchool: School;

После этих определений возможны следующие присваивания:

ClassA [J]:='ВАСЯ';

NewTownSchool [5][21]:='ПЕТЯ ИВАНОВ';

NewTownSchool [8,J]:=NewTownSchool [7,J];

{ Ученик номер J меняет класс }

ClassA [Vacant]:=ClassB [P];

{ Ученик номер P меняет класс и номер }

**Массивы символов**

Массивы символов - это массивы с одним индексом и компонентами стандартного простого типа CHAR. Массивы символов могут рассматриваться как строки постоянной длины.

В Турбо-Паскале массивы символов могут использоваться в строковых выражениях; в этом случае массив преобразуется в строку длины, равной длине массива. Таким образом, массивы можно сравнивать и манипулировать ими так же, как и строками; значение строковой КОНСТАНТЫ может быть присвоено массиву символов той же длины. Строковые переменные и значения, являющиеся результатом строковых выражений, не могут быть присвоены массиву символов.

**Стандартные массивы**

Турбо-Паскаль дает возможность пользоваться двумя стандартными массивами типа BYTE, а именно MEM и PORT, которые используются для доступа к памяти процессора и портам данных.

**Массив MEM**

Массив MEM используется для доступа к оперативной памяти. Каждая компонента массива имеет тип BYTE, а индексы соответствуют адресам памяти. Индекс имеет тип INTEGER. Если некоторое значение присваивается компоненте массива MEM, то это значение записывается в память по адресу, определяемому индексным выражением. Когда массив MEM участвует в выражениях, используется значение байта, расположенного по адресу, который задается индексом.

Примеры:

Mem [I]:=2;

Mem [I+1]:=$1B;

Mem [I+2]:=Ord(' ');

IObyte:=Mem [3];

Mem [Addr+Offset]:=Mem [Addr];

**Массив PORT**

Массив PORT используется для доступа к портам ввода/вывода микропроцессора Z80. Каждый элемент массива представляет один порт данных, причем индексы соответствуют номерам портов. Поскольку для выбора портов используются 8-разрядные адреса, индекс имеет тип BYTE. Когда некоторое значение присваивается компоненте массива PORT, это значение выводится в соответствующий порт. Когда же компонента такого массива употребляется в выражении, ее значение вводится из соответствующего порта. Использование массива портов ограничивается присваиванием значений и употреблением в качестве элементов выражений; в частности, компоненты массива PORT не могут использоваться как параметры-переменные процедур и функций. Более того, запрещены операции со ссылкой на весь массив портов как на единое целое (без индекса).

Примеры:

Port[$F8]:=$0E;

X:=Port[$0A];

**Оптимизация индексов массивов**

Директива компилятора X позволяет программисту указать, следует ли проводить оптимизацию индексов массивов в отношении скорости выполнения или размера кода. По умолчанию подразумевается активный режим этой директивы: {$X+}, который порождает оптимизацию скорости выполнения. При пассивном состоянии: {$X-} минимизируется размер кода.

**ГЛАВА 11.**

**ЗАПИСИ**

Запись - это структура, состоящая из фиксированного числа компонент, называемых ПОЛЯМИ. Поля могут быть различных типов; каждому полю присваивается имя - ИДЕНТИФИКАТОР ПОЛЯ, которое используется для обращения к этому полю.

**Определение записи**

Определение типа записи состоит из зарезервированного слова RECORD, за которым следует СПИСОК ПОЛЕЙ, оканчивающийся зарезервированным словом END. Список полей - это последовательность СЕКЦИЙ ЗАПИСИ, разделенных точками с запятой. Каждая секция записи, в свою очередь, состоит из одного или более идентификаторов, разделенных запятыми, за которыми следует двоеточие и либо ИДЕНТИФИКАТОР ТИПА, либо ОПИСАТЕЛЬ ТИПА. Таким образом, каждая секция записи определяет идентификатор и тип для одного или более полей.

Пример:

type

DaysOfMonth = 1..31;

Date = record

Day: DaysOfMonth;

Month: (Jan,Feb,Mar,Apr,May,Jun,

July,Aug,Sep,Oct,Nov,Dec);

Year: 1900..1999;

end;

var

Birth: Date;

WorkDay: array [1..5] of Date;

В данном примере Day, Month и Year - это идентификаторы полей. Идентификаторы полей должны быть уникальными только в пределах записи, в которой они определены. На поле можно сослаться путем указания идентификатора переменной и идентификатора поля, разделенных точкой.

Примеры:

Birth.Month:=Mar;

Birth.Year:=1962;

WorkDay [Current]:=WorkDay [Current-1];

Отметим, что как и для массивов, для записей допускается присваивание значений между записями одинаковых типов. Поскольку компонентами записи могут быть данные любого типа, допустимыми являются конструкции, подобные той, что показана ниже, где запись содержит другую запись, которая, в свою очередь, тоже содержит запись.

type

Name = record

FamilyName: string[32];

ChristianNames: array [1..3] of string[16];

end;

Rate = record

NormalRate, OverTime,

NightTime, WeekEnd: integer;

end;

Date = record

Day: 1..31;

Month: (Jan,Feb,MAr,Apr,May,Jun,

July,Aug,Sep,Oct,Nov,Dec);

Year: 1900..1999;

end;

Person = record

ID: Name;

Time: Date;

end;

Wages = record

Individual: Person;

Cost: Rate;

end;

var

Salary, Fee: Wages;

После этих определений возможны следующие операторы присваивания:

Salary:=Fee;

Salary.Cost.OverTime:=950;

Salary.Individual.Time:=Fee.Individual.Time;

Salary.Individual.ID.FamilyName:='Smith';

**Оператор WITH**

Использование записей в том виде, как это показано выше, иногда приводит к необходимости очень длинных операторов, и в этом смысле было бы удобнее обращаться к отдельным полям записи как к простым переменным. Именно для этого и предназначен оператор WITH: он "раскрывает" запись так, что идентификаторы полей могут использоваться как идентификаторы переменных.

Оператор WITH состоит из зарезервированного слова WITH, за которым следуют: список переменных типа RECORD, разделенных запятыми, зарезервированное слово DO и, наконец, оператор.

Внутри оператора WITH поле обозначается только своим идентификатором, т.е. без идентификатора переменной-записи:

with Salary do

begin

Individual:=NewEmployee;

Cost:=StandardRates;

end;

Допускается вложенность операторов WITH, т.е. "раскрытие" записей внутри других записей. Например, конструкция:

with Salary,Individual,ID do

begin

FamilyName:='Smith';

ChristianNames [1]:='James';

end;

эквивалентна следующей:

with Salary do

with Individual do

with ID do

. . .

Максимальная глубина вложения операторов WITH, подразумеваемая по умолчанию, равна 2, однако с помощью директивы W компилятора это значение может быть установлено равным от 0 до 9. Для каждого блока оператор WITH требует двух байтов памяти на каждый разрешенный уровень вложенности. Поэтому, для экономии памяти вложенность операторов WITH необходимо поддерживать на возможно более низком уровне.

**Записи с вариантами**

Синтаксис типа RECORD предусматривает также наличие вариантной части, что позволяет организовывать альтернативные структуры записей так, чтобы конкретные экземпляры записей могли содержать различное число компонент различных типов, обычно в зависимости от некоторого ПОЛЯ ПРИЗНАКА.

Вариантная часть состоит из поля признака ранее определенного типа, за которым следуют метки, соответствующие каждому возможному значению поля признака. Различные значения поля признака определяют разные варианты записи, а каждая метка возглавляет СПИСОК ПОЛЕЙ, который определяет тип варианта, соответствующего данной метке.

Предположим существование типа:

Origin = (Citizen,Alien);

и типов Name и Date, определенных в в предыдущем примере.

Следующая запись может иметь различную структуру в зависимости от значения поля признака CitizenShip: Citizen или Alien.

type

Person = record

PersonName: Name;

BirthDate: Date;

case CitizenShip: Origin of

Citizen: (BirthPlace: Name);

Alien: (CountryOfOrigin: Name;

DateOfEntry : Date;

PermittedUntil : Date;

PortOfEntry : Name);

end;

В этом определении записи с вариантами поле признака представляет собой явно описанное поле, которое можно извлекать и обновлять, как и всякое другое поле. Следовательно, если переменная Passenger имеет тип Person, то показанные ниже операторы являются допустимыми:

Passenger.CitizenShip:=Citizen;

With Passenger,PersonName do

if CitizenShip=Alien then

Writeln (FamilyName);

Фиксированная часть записи, т.е. часть, содержащая общие поля, должна всегда предшествовать вариантной части. В предыдущем примере поля PersonName и BirthDate - это фиксированные поля. Запись может иметь только одну вариантную часть. Присутствие круглых скобок в вариантах является обязательным, даже если они заключают в себе пустоту.

Турбо-Паскаль не отвечает за установку соответствующих значений в поля признака: ответственность за это возлагается на программиста. Следовательно, в показанном выше типе Person доступ к полю DateOfEntry можно получить даже тогда, когда значение поля признака CitizenShip не равно Alien. В действительности идентификатор поля признака может быть вообще опущен, за исключением идентификатора типа. Такие записи с вариантами называются СВОБОДНЫМИ ОБЪЕДИНЕНИЯМИ, в противоположность вариантам записей с полем признака, которые называются СВЯЗАННЫМИ ОБЪЕДИНЕНИЯМИ. Свободные объединения применяются редко и должны использоваться только опытными программистами.

**ГЛАВА 12.**

**МНОЖЕСТВА**

Множество - это совокупность связанных объектов, которая может рассматриваться как единое целое. Каждый объект в таком множестве называется ЧЛЕНОМ или ЭЛЕМЕНТОМ множества. Примерами множеств могут служить:

1) Все целый числа от 0 до 100;

2) Буквы алфавита;

3) Согласные буквы алфавита.

Два множества равны тогда и только тогда, когда все их элементы совпадают. При этом порядок их несущественный. Так, множества [1,3,5], [5,3,1], [3,5,1] равны. Если члены одного множества являются также членами другого множества, первое множество называется включенным во второе. Так, в приведенных выше примерах множество 3) включено во множество 2).

Существует три операции над множествами, аналогичных операциям сложения, умножения и вычитания целых чисел:

1) ОБЪЕДИНЕНИЕ (или сумма) двух множеств A и B (записывается как A+B) - это множество, содержащее все члены множеств A и B. Например, объединение множеств [1,3,5,7] и [2,3,4] дает [1,2,3,4,5,7].

2) ПЕРЕСЕЧЕНИЕ (или произведение) двух множеств A и B (записывается как A\*B) - это множество тех элементов, которые входят и во множество A, и во множество B. Так, пересечение множеств [1,3,4,5,7] и [2,3,4] дает [3,4].

3) ДОПОЛНЕНИЕ (или вычитание) множества B до множества A (записывается как A-B) - это множество тех элементов множества A, которые не являются элементами множества B.

Так, [1,3,5,7]-[2,3,4] дает [1,5,7].

**Определение типа множества**

Как и в математике, не существует ограничений на объекты, которые могут быть элементами множества. Паскаль допускает только ограниченную форму множеств. Все элементы множества должны быть одного типа, который называется БАЗОВЫМ ТИПОМ. Базовый тип должен быть простым типом, т.е. любым простым типом, за исключением REAL. Тип множества вводится зарезервированными словами SET OF, за которыми следует простой тип.

Примеры:

type

DayOfMonth = set of 0..31;

WorkWeek = set of Mon..Fri;

Letter = set of 'A'..'Z';

AdditiveColors = set of (Red,Green,Blue);

Characters = set of Char;

В Турбо-Паскале максимальное число элементов множества равно 256, и порядковые значения базового типа должны быть в диапазоне от 0 до 255.

**Выражения с множествами**

Значения типа множества могут быть получены из других значений типа множества с помощью выражений. Выражения с множествами строятся из констант типа множества, переменных типа множества, множественных конструкций и знаков операций над множествами.

**Множественные конструкции**

Множественная конструкция состоит из одной или более спецификаций элемента, разделенных запятыми и заключенных в квадратные скобки. Спецификация элемента - это выражение того же типа, что и базовый тип множества, или диапазон, записанный как два таких выражения, разделенных двумя последовательными точками.

Примеры:

['T','U','R','B','O']

[X,Y]

[X..Y]

[1..5]

['A'..'Z','a'..'z','0'..'9']

[1,3..10,12]

[]

Последний пример показывает ПУСТОЕ МНОЖЕСТВО, которое, поскольку не содержит выражений, указывающих базовый тип, совместимо с любым типом множества. Множество [1..5] эквивалентно множеству [1,2,3,4,5]. Если X > Y, то [X..Y] задает пустое множество.

**Операции над множествами**

По приоритету операции над множествами разделяются на следующие три категории:

1) \* Пересечение множеств

2) + Объединение множеств

- Дополнение множеств

3) = Проверка на равенство

<> Проверка на неравенство

>= Истина (TRUE), если второй операнд включен в первый

<= Истина (TRUE), если первый операнд включен во второй

IN Проверка принадлежности множеству. Второй операнд должен быть типа множества, а первый операнд должен быть того же типа, что и базовый тип множества. Результатом операции будет TRUE, если первый операнд принадлежит второму операнду, и FALSE в противном случае.

Не существует операции проверки не включения, но она может быть запрограммирована как:

A\*B=[]

Выражения с множествами часто полезны для записи сложных проверок. Например, проверка:

if (Ch='T') or (Ch='U') or (Ch='R') or (Ch='B') or (Ch='O')

then ...

может быть записана более компактно как:

if Ch in ['T','U','R','B','O'] then ...

А проверку:

if (Ch>='0') and (Ch<='9') then ...

лучше записать как:

if Ch in ['0'..'9'] then ...

**Присваивание значений множествам**

Значения, являющиеся результатом выражений с множествами, могут быть присвоены переменным множественного типа с помощью оператора присваивания := .

Примеры:

type

ASCII = set of 0..127;

var

NoPrint,Print,AllChars: ASCII;

begin

AllChars:= [0..127];

NoPrint := [0..31,127];

Print := AllChars-NoPrint;

end.

**ГЛАВА 13.**

**КОНСТАНТЫ С ЗАДАННЫМ ТИПОМ**

Константы с заданным типом - это расширение стандарта, предлагаемое Турбо-Паскалем. Такие константы могут использоваться точно так же, как переменные того же типа. Константы с заданным типом могут поэтому использоваться как "инициализированные переменные", потому что значение константы с указанным типом определено, а значение переменной не определено до присваивания ей значения. Однако не нужно, конечно, присваивать различные значения константе с указанным типом, если ее значение предполагается постоянным.

Использование констант с заданным типом уменьшает длину программного кода, особенно если константа используются в программе часто, поскольку константа с указанным типом включается в программу только один раз, а константы без типа включаются при каждом использовании.

Константы с заданным типом определяются подобно константам без типа (см. подраздел "Раздел определений констант" в главе 5) за тем исключением, что указывается не только значение константы, но и ее тип. В определении таких констант записывается идентификатор, затем двоеточие, после него - идентификатор типа, знак равенства и значение константы.

**Неструктурные константы с заданным типом**

Неструктурная константа с заданным типом - это константа, принадлежащая одному из простых типов.

Примеры:

const

NumberOfCards: Integer = 1267;

Interest: Real = 12.67;

Heading: string[7] = 'SECTION';

Xon: Char = ^Q;

В противоположность константам без типа константы с заданным типом можно использовать в качестве фактического параметра-переменной при вызове процедуры или функции. Поскольку константа с заданным типом по сути является переменной с постоянным значением, она не может быть использована в определении других констант или типов. Так, поскольку Min и Max – константы с заданным типом, следующая конструкция НЕДОПУСТИМА:

const

Min: Integer = 0;

Max: Integer = 50;

type

Range: array [Min..Max] of Integer;

**Структурные константы с заданным типом**

Структурными константами могут быть константы-массивы, константы-записи и константы-множества. Они часто используются для построения инициализированных таблиц и множеств тестов, преобразований и т.д. Следующие разделы содержат подробное описание каждого из этих типов.

**Константы-массивы**

Определение константы-массива состоит из идентификатора константы, за которым следует двоеточие и идентификатор предварительно определенного типа массива, за ним следует знак равенства и значение константы, записанное как множество констант, разделенных запятыми и заключенных в круглые скобки.

Примеры:

type

Status = (Active,Passive,Waiting);

StringRep = array [Status] of string[7];

const

Stat: StringRep = ('active','passive','waiting');

В этом примере определена константа-массив Stat, которая может быть использована для преобразования значений простого типа Status в их соответствующее строковое представление. Компоненты Stat таковы:

Stat [Active] = 'active'

Stat [Passive]= 'passive'

Stat [Waiting]= 'waiting'

Тип компонент константы-массива может быть любым, кроме FILE и POINTER. Символьные константы-массивы могут состоять как из одиночных символов, так и из строк. Так, определение:

const

Digits: array [0..9] of Char =

('0','1','2','3','4','5','6','7','8','9');

может быть записано более удобно как:

const

Digits: array [0..9] of Char = '0123456789';

**Многомерные константы-массивы**

Многомерные константы-массивы определяются путем заключения констант каждого измерения в отдельные круглые скобки, разделенные запятыми. Константы самого внутреннего уровня вложенности соответствуют самому правому измерению.

Пример:

type

Cube = array [0..1,0..1,0..1] of Integer;

const

Maze: Cube = (((0,1),(2,3)),((4,5),(6,7)));

begin

Writeln (Maze[0,0,0],' = 0');

Writeln (Maze[0,0,1],' = 1');

Writeln (Maze[0,1,0],' = 2');

Writeln (Maze[0,1,1],' = 3');

Writeln (Maze[1,0,0],' = 4');

Writeln (Maze[1,0,1],' = 5');

Writeln (Maze[1,1,0],' = 6');

Writeln (Maze[1,1,1],' = 7');

end.

**Константы-записи**

Определение константы-записи состоит из идентификатора константы, за которым следует двоеточие и идентификатор предварительно определенного типа записи, за ним следует знак равенства и значение константы, записанное как список констант-полей, разделенных символом ";" и заключенных в круглые скобки.

Примеры:

type

Point = record

X,Y,Z: Integer;

end;

OS = (CPM80,CPM86,MSDOS,UNIX);

UI = (CCP,SomethingElse,MenuMaster);

Computer = record

OperatingSystems: array [1..4] of OS;

UserInterface: UI;

end;

const

Origo: Point = (X:0; Y:0; Z:0);

SuperComp: Computer =

(OperatingSystems:(CPM80,CPM86,MSDOS,UNIX);

UserInterface: MenuMaster);

Planel: array [1..3] of Point =

((X: 1; Y: 4; Z: 5),

(X: 10; Y:-78; Z:45),

(X:100; Y: 10; Z:-7));

Константы-значения полей должны указываться в том же порядке, в котором они появляются в определении типа записи. Если запись содержит поля типа FILE или POINTER, то константы такого типа записи задать невозможно. Если запись имеет варианты, то ответственность за спецификацию полей, соответствующих требуемому варианту, возлагается на программиста. Если вариант содержит поле признака, то значение этого поля должно быть указано.

**Константы-множества**

Константа-множество состоит из одной или более спецификаций элемента, разделенных запятыми и заключенных в квадратные скобки. Спецификация элемента - это константа или выражение, задающее отрезок, которое состоит из двух констант, разделенных точками ".." .

Пример:

type

Up = set of 'A'..'Z';

Low = set of 'a'..'z';

const

UpperCase: Up = ['A'..'Z'];

Vocals: Low = ['a','e','i','o','u','y'];

Delimiter: set of Char =

[' '..'/',':'..'?','{'..'~','['..'\_'];

**ГЛАВА 14.**

**ФАЙЛЫ**

ФАЙЛЫ обеспечивают программу каналами, через которые она может передавать данные. В качестве файла может выступать некоторый ДИСКОВЫЙ ФАЙЛ, когда операции чтения/записи данных осуществляются с каким-либо магнитным устройством, или ЛОГИЧЕСКОЕ УСТРОЙСТВО, такое как стандартные предопределенные файлы INPUT и OUTPUT, которые соответствуют стандартным каналам ввода/вывода компьютера - клавиатуре и экрану дисплея.

Файл состоит из последовательности компонент одного и того же типа. Число этих компонент в файле (РАЗМЕР ФАЙЛА) не указывается в определении файла. Вместо этого Турбо-Паскаль следит за доступом к файлу с помощью так называемого УКАЗАТЕЛЯ ФАЙЛА, и каждый раз, когда над некоторой компонентой файла выполняется операция чтения или записи, указатель этого файла продвигается на следующую компоненту. Поскольку все компоненты файла имеют одинаковую длину, может быть вычислена позиция любой заданной компоненты. Таким образом, указатель файла может быть перемещен на любую компоненту файла, что обеспечивает произвольный доступ к ней.

**Определение типа файла**

В определении типа файла указываются зарезервированные слова FILE OF, за которыми следует тип компонент этого файла. Идентификатор файла объявляется теми же самыми зарезервированными словами, за которыми следует идентификатор ранее определенного типа файла.

Примеры:

type

ProductName = string[80];

Product = file of record

Name: ProductName;

ItemNumber: Real;

InStock: Real;

MinStock: Real;

Supplier: Integer;

end;

var

ProductFile: Product;

ProductNames: file of ProductName;

Компоненты файла могут иметь любой тип, за исключением типа FILE (это означает, в частности, что в предыдущем примере определение "file of Product" является недопустимым).

**Операции над файлами**

Ниже описываются процедуры, предназначенные для обработки файлов. Обозначению FilVar соответствует идентификатор переменной типа FILE, объявленной как описано ранее.

**Процедура ASSIGN**

Синтаксис: ASSIGN(FilVar,StrExp);

StrExp - это строковое выражение, результатом которого является строка, представляющая любое допустимое имя файла. Это имя файла присваивается переменной FilVar, и все дальнейшие операции над FilVar будут на самом деле выполняться с дисковым файлом StrExp. Эта операция никогда не должна выполняться над файлом, который находится в работе.

**Процедура REWRITE**

Синтаксис: REWRITE(FilVar);

Создается новый дисковый файл с именем, которое связано с файловой переменной FilVar; этот файл подготавливается для последующей обработки, а его указатель устанавливается на начало файла, т.е. на компоненту с номером 0. Если на диске ранее существовал файл с таким же именем, то этот файл стирается. Дисковый файл, созданный посредством процедуры REWRITE, первоначально является пустым, т.е. не содержит ни одного элемента.

**Процедура RESET**

Синтаксис: RESET(FilVar);

Дисковый файл с именем, связанным с переменной FilVar, подготавливается к обработке, и указатель этого файла устанавливается на начало файла, т.е. нулевую компоненту. Файл с указанным именем должен существовать, в противном случае фиксируется ошибка ввода/вывода.

**Процедура READ**

Синтаксис: READ(FilVar,VarList);

VarList обозначает одну или более переменных, разделенных запятыми и имеющих тот же тип, что и компоненты файла FilVar. Каждая переменная считывается из дискового файла, и после каждой операции чтения указатель файла продвигается к следующей компоненте файла.

**Процедура WRITE**

Синтаксис: WRITE(FilVar,VarList);

VarList обозначает одну или более переменных, разделенных запятыми и имеющих тот же тип, что и компоненты файла FilVar. Каждая переменная записывается в дисковый файл, и после каждой операции записи указатель файла перемещается на очередную компоненту.

**Процедура SEEK**

Синтаксис: SEEK(FilVar,N);

Эта процедура перемещает указатель файла так, что он устанавливается на N-ю компоненту файла FilVar, N - целое выражение. Первой компоненте соответствует значение N=0. Заметим, что для расширения файла можно установить указатель на компоненту, непосредственно следующую за последней компонентой файла. Таким образом, оператор

SEEK(FilVar,FileSize(FilVar));

устанавливает указатель файла в конец этого файла (функция FileSize возвращает число компонент в файле, а поскольку компоненты нумеруются с нуля, это число на единицу больше номера последней компоненты). Процедура SEEK неприменима к текстовым файлам.

**Процедура FLUSH**

Синтаксис: FLUSH(FilVar);

Эта процедура "опорожняет" внутренний буфер (размером в один сектор диска) дискового файла, связанного с файловой переменной FilVar, и тем самым обеспечивает запись этого буфера на диск, если с момента последнего обновления диска имели место какие-либо операции ввода/вывода. Процедура FLUSH также обеспечивает то, что при последующей операции чтения будет в действительно выполнено физическое чтение из дискового файла. Нельзя применять эту процедуру к закрытому файлу. Процедура FLUSH неприменима к текстовым файлам.

**Процедура CLOSE**

Синтаксис: CLOSE(FilVar);

Дисковый файл, связанный с FilVar, закрывается и модифицируется оглавление диска для отражения нового состояния файла. Заметим, что файл необходимо закрывать, даже если этот файл использовался только для чтения, ибо в противном случае можно выйти из-под контроля файловой системы.

**Процедура ERASE**

Синтаксис: ERASE(FilVar);

Дисковый файл, связанный с FilVar, стирается. Если файл открыт, т.е. были выполнены процедуры REWRITE или RESET, перед стиранием его лучше закрыть.

**Процедура RENAME**

Синтаксис: RENAME(FilVar,StrExp);

Дисковый файл, связанный с FilVar, получает новое имя, являющееся результатом строкового выражения StrExp. Оглавление диска модифицируется, и последующие операции с FilVar будут выполняться над файлом с новым именем. Процедуру RENAME нельзя

применять к открытому файлу.

Заметим, что ответственность за то, чтобы новое имя не было именем уже существующего файла, лежит на программисте. Если это произошло, в результате на диске появится несколько файлов с одинаковыми именами. Следующая функция возвращает значение TRUE, если файл с именем, переданным ей в качестве параметра, имеется на диске, и FALSE в противном случае.

type

Name = string[14];

. . .

. . .

function Exist (FileName: Name): boolean;

var

Fil: file;

begin

Assign (Fil,FileName);

{$I-} Reset (Fil); {$I+}

Exist:=(IOresult=0);

{$I-} Close (Fil); {$I+}

if (IOresult=0) then;

end;

**Файловые стандартные функции**

В последующих подразделах рассматриваются стандартные функции, предназначенные для работы с файлами, которые имеются в Турбо-Паскале.

**Функция EOF**

Синтаксис: EOF(FilVar)

Эта функция типа BOOLEAN возвращает TRUE, если указатель позиционируется за концом дискового файла, т.е. после последней компоненты файла. Если нет, возвращается значение FALSE.

**Функция FILEPOS**

Синтаксис: FILEPOS(FilVar)

Функция типа INTEGER, возвращает номер компоненты, на которую установлен указатель файла. Для первой компоненты возвращается нуль.

**Функция FILESIZE**

Синтаксис: FILESIZE(FilVar)

Функция типа INTEGER, возвращает размер дискового файла, выраженный как количество компонент в этом файле. Если FILESIZE (FilVar) равно нулю, файл пуст. Файл должен быть открыт.

**Использование файлов**

Перед использованием файла необходимо вызвать процедуру ASSIGN для присвоения имени файла переменной типа FILE. Перед выполнением операций ввода/вывода файл должен быть открыт путем вызова процедур REWRITE или RESET. Этот вызов устанавливает указатель файла на первую компоненту дискового файла, т.е. FILEPOS (FilVar) равно 0. После вызова процедуры REWRITE значение FILESIZE (FilVar) равно 0.

Дисковый файл может быть расширен только путем добавления компонент в конец существующего файла. Указатель файла может быть перемещен в конец существующего файла с помощью оператора:

SEEK (FilVar,FileSize(FilVar));

Когда программа заканчивает операции ввода/вывода над файлом, она всегда должна закрыть файл с помощью вызова процедуры CLOSE. Невыполнение этого может привести к потере данных, поскольку оглавление диска не будет обновлено.

Приведенная ниже программа создает дисковый файл PRODUCT.DTA и записывает в этот файл 100 записей типа Product. Тем самым файл инициализируется для последующего произвольного доступа (т.е. записи могут быть считаны и записаны в любом месте файла).

program InitProductFile;

const

MaxNumberOfProducts = 100;

type

ProductName = string[20];

Product = record

Name: ProductName;

ItemNumber: Integer;

InStock: Real;

Supplier: Integer;

end;

var

ProductFile: file of Product;

ProductRec: Product;;

i: Integer;

begin

Assign (ProductFile,'PRODUCT.DTA');

{ Открытие файла и уничтожение старых данных }

Rewrite (ProductFile);

with ProductRec do

begin

Name:='';

InStock:=0;

Supplier:=0;

for i:=1 to MaxNumberOfProducts do

begin

ItemNumber:=i;

Write (ProductFile,ProductRec);

end;

end;

Close (ProductFile);

end.

Следующая программа показывает использование процедуры SEEK для файлов с произвольным доступом. Программа модифицирует файл PRODUCT.DTA, созданный программой из предыдущего примера.

program UpdateProductFile;

const

MaxNumberOfProducts = 100;

type

ProductName = string[20];

Product = record

Name: ProductName;

ItemNumber: Integer;

InStock: Real;

Supplier: Integer;

end;

var

ProductFile: file of Product;

ProductRec: Product;;

i,Pnr: integer;

begin

Assign (ProductFile,'PRODUCT.DTA');

Reset (ProductFile);

ClrScr;

Write ('Enter product number (0-stop) ');

Readln(Pnr);

while Pnr in [1..MaxNumberOfProducts] do

begin

Seek (ProductFile,Pnr-1);

Read (ProductFile,ProductRec);

with ProductRec do

begin

Write('Enter name of product (',Name:20,') ');

Readln(Name);

Write('Enter number in stock (',InStock:20:0,') ');

Readln(InStock);

Write('Enter supplier number (',Supplier:20,') ');

Readln(Supplier);

ItemNumber:=Pnr;

end;

Seek(ProductFile,Pnr-1);

Write(ProductFile,ProductRec);

ClrScr;

Writeln;

Write('Enter product number (0-stop) ');

Readln(pnr);

end;

Close (ProductFile);

end.

**Текстовые файлы**

В отличие от других типов файлов ТЕКСТОВЫЕ ФАЙЛЫ организованы несколько сложнее, чем просто последовательность значений некоторого типа. Хотя базовыми компонентами текстового файла являются символы (тип CHAR), эти символы сгруппированы в строки, каждая из которых заканчивается ПРИЗНАКОМ КОНЦА СТРОКИ (это комбинация CR/LF). Конец всего файла отмечается признаком КОНЦА ФАЙЛА (CTRL-Z). Поскольку длина строк может изменяться, положение строки в файле не может быть вычислено. Поэтому текстовые файлы можно обрабатывать только последовательно. Более того, с текстовым файлом невозможно одновременно выполнять ввод и вывод.

**Операции над текстовыми файлами**

Переменная типа текстовый файл объявляется с помощью ссылки на стандартный идентификатор типа TEXT. Всем операциям над таким файлом должен предшествовать вызов процедуры ASSIGN и одной из процедур REWRITE или RESET.

Процедура REWRITE используется для создания нового текстового файла, и единственной разрешенной операцией является добавление новых компонент в конец файла. Процедура RESET открывает уже существующий текстовый файл для чтения, и единственной разрешенной операцией для такого файла является последовательное чтение. При закрытии текстового файла к нему автоматически добавляется признак конца файла.

Операции ввода и вывода над текстовыми файлами выполняются стандартными процедурами READ и WRITE. Для обработки строк в текстовых файлах служат специальные процедуры READLN, WRITELN и функция EOLN.

Далее обозначению FilVar будет соответствовать идентификатор переменной типа TEXT.

**Процедура READLN**

Синтаксис: READLN(FilVar);

Выполняется продвижение к началу следующей строки, т.е. пропускаются все знаки до очередной комбинации CR/LF, включая ее.

**Процедура WRITELN**

Синтаксис: WRITELN(FilVar);

В текстовый файл записывается признак конца строки, т.е. комбинация CR/LF.

**Функция EOLN**

Синтаксис: EOLN(FilVar)

Логическая функция EOLN возвращает значение TRUE, если был достигнут конец текущей строки, т.е. если указатель файла установлен на символ CR признака конца строки CR/LF. Если значение EOF(FilVar) равно TRUE, то значение EOLN(FilVar) также равно

TRUE.

**Функция SEEKEOLN**

Синтаксис: SEEKEOLN(FilVar)

Эта функция работает аналогично EOLN, но перед проверкой на наличие признака конца строки она пропускает пробелы и знаки табуляции. Результат имеет тип BOOLEAN.

**Функция SEEKEOF**

Синтаксис: SEEKEOF(FilVar)

Эта функция работает аналогично EOF, но перед проверкой на наличие признака конца файла она пропускает пробелы, знаки табуляции и признаки конца строки (комбинации CR/LF). Результат имеет тип BOOLEAN.

Применительно к текстовому файлу функция EOF возвращает значение TRUE, если указатель файла позиционирован на признаке конца файла (символ CTRL-Z). Процедуры SEEK и FLUSH, а также функции FILEPOS и FILESIZE к текстовым файлам неприменимы.

Программа, приведенная в следующем примере, читает с диска текстовый файл и печатает его на стандартном устройстве LST - принтере. Те слова в файле, которые обрамляются символами CTRL-S, при печати подчеркиваются.

program TextFileDemo;

var

FilVar: text;

Line,

ExtraLine: string[255];

i: Integer;

UnderLine: Boolean;

FileName: string[14];

begin

UnderLine:=False;

Write('Enter name of file to list: ');

Readln(FileName);

Assign(FilVar,FileName);

Reset(FilVar);

While not Eof(Filvar) do

begin

Readln(Filvar,Line);

i:=1;

ExtraLine:='';

for i:=1 to Length(Line) do

begin

if Line[i]<>^S then

begin

Write(Lst,Line[i]);

if UnderLine then

ExtraLine:=ExtraLine+'\_'

else

ExtraLine:=ExtraLine+' ';

end

else

UnderLine:=not UnderLine;

end;

Write(Lst,^M);

Writeln(Lst,ExtraLine);

end;

Close(FilVar);

end.

Дальнейшие расширения процедур READ и WRITE, которые облегчают обработку форматированного ввода/вывода, описаны в подразделах "Процедура READ" и "Процедура WRITE" раздела "Ввод и вывод текста" настоящей главы.

**Логические устройства**

В Турбо-Паскале внешние устройства, такие как терминалы, печатающие устройства, модемы рассматриваются как ЛОГИЧЕСКИЕ УСТРОЙСТВА, работа с которыми подобна работе с текстовыми файлами. Допустимы следующие логические устройства:

CON: Консольное устройство. Выводимые символы направляются на консольное устройство вывода операционной системы, обычно дисплей, а вводимая информация принимается с консольного устройства ввода - клавиатуры. В отличие от устройства TRM: устройство CON: обеспечивает буферизацию ввода. Коротко говоря, это означает, что каждый вызов READ или READLN для текстового файла, связанного с устройством CON: , вводит целую строку в буфер строки, и что при вводе оператору предоставляется некоторый набор средств редактирования. Дальнейшие подробности ввода с консоли описаны в двух последующих подразделах.

TRM: Терминальное устройство. Выводимые символы направляются на консольное устройство вывода операционной системы, обычно дисплей, а вводимая информация принимается с консольного устройства ввода, обычно с клавиатуры. Ввод символов сопровождается эхом на экране (за исключением управляющих символов). Единственный из управляющих символов, который изображается на экране -это возврат каретки (CR), причем отображение этого символа выглядит как CR/LF.

KBD: Клавиатура (только для ввода). Производится ввод с консольного устройства операционной системы без эха.

LST: Устройство печати (только для вывода). Выводимые символы посылаются на системное устройство печати (обычно принтер).

AUX: Вспомогательное устройство. Выводимые символы посылаются на системное устройство перфорации (PUN:), а вводимая информация принимается из системного устройства ввода с перфоленты (RDR:). Обычно этим устройствам соответствует модем.

USR: Устройство пользователя. Выводимая информация передается пользовательской программе вывода, а за вводимой информацией производится обращение к пользовательской программе ввода. Указанная возможность описана в следующем подразделе "Пользовательские драйверы ввода/вывода".

Доступ к этим логическим устройствам возможен через стандартные файлы (они обсуждаются ниже), с другой стороны такие устройства можно привязывать к файловым переменным посредством процедуры ASSIGN в точности так, как это делается с дисковыми файлами. Для файла, связанного с логическим устройством, процедуры REWRITE и RESET выполняют одинаковые действия, процедура CLOSE не выполняет никаких действий, а попытка выполнить процедуру ERASE для такого файла приведет к ошибке ввода/вывода.

Стандартные функции EOF и EOLN действуют для логических устройств иначе, чем для дисковых файлов. Для дисковых файлов EOF возвращает TRUE, когда следующим символом файла является символ CTRL-Z или если достигнут физический конец файла. EOLN возвращает TRUE, если следующим символом является символ CR или CTRL-Z. Таким образом, EOF и EOLN являются фактически функциями "просмотра вперед".

Поскольку невозможно "смотреть вперед" на логическом устройстве. функции EOF и EOLN работают с последним прочитанным символом, а не со следующим символом. EOF возвращает TRUE, если последним прочитанным символом был символ CTRL-Z, EOLN возвращает TRUE, если последним прочитанным символом был символ CR или CTRL-Z. В таблице ниже показаны различные варианты работы функций EOF и EOLN.

|  |  |  |
| --- | --- | --- |
|  | Для файлов | Для лог. устройств |
| EOLN возвращает TRUE | Если следующий символ CR или CTRL-Z, или если EOF возвращает значение TRUE | Если текущий сим вол CR или CTRL-Z |
| EOF возвращает TRUE | Если следующий символ CTRL-Z, или достигнут физический конец файла | Если текущий символ CTRL-Z |

Процедура READLN также работает по-разному для логических устройств и для дисковых файлов. Из дисковых файлов эта процедура считывает все символы до конца строки, включая последовательность CR/LF, тогда как с логических устройств она читает только строку и символ CR. Причиной этого является невозможность "просмотра вперед" на логических устройствах, т.е. система не может определить, какой символ будет следовать за символом CR.

**Пользовательские драйверы ввода/вывода**

В некоторых приложениях у программиста возникает потребность создания своих собственных драйверов ввода/вывода, т.е. подпрограмм, которые осуществляют ввод и вывод знаков на физическом уровне, т.е. работают непосредственно с внешними устройствами. Ниже приводится список драйверов, которые являются частью среды Турбо-системы и используются стандартными драйверами ввода/вывода (хотя они и не являются доступными в качестве стандартных процедур и функций):

function ConSt: Boolean;

function ConIn: Char;

procedure ConOut (Ch: Char);

procedure LstOut (Ch: Char);

procedure AuxOut (Ch: Char);

function AuxIn: Char;

procedure UsrOut (Ch: Char);

function UsrIn: Char;

Подпрограмма ConSt вызывается функцией KEYPRESSED, подпрограммы ConIn и ConOut используются устройствами CON:, TRM: и KBD: , подпрограмма LstOut используется устройством LST: , подпрограммы AuxOut и AuxIn используются устройством AUX: , и, наконец, подпрограммы UsrOut и UsrIn используются устройством USR: .

По умолчанию эти драйверы используют соответствующие точки входа подсистемы BIOS операционной системы CP/M. Однако программист может изменить эти установки, присвоив одной из показанных ниже переменных адрес написанной им процедуры или функции драйвера:

|  |  |
| --- | --- |
| Переменная: | Содержит адрес: |
| ConStPtr | Функции ConSt |
| ConInPtr | Функции ConIn |
| ConOutPtr | Процедуры ConOut |
| LstOutPtr | Процедуры LstOut |
| AuxOutPtr | Процедуры AuxOut |
| AuxInPtr | Функции AuxIn |
| UsrOutPtr | Процедуры UsrOut |
| UsrInPtr | Функции UsrIn |

Определенная пользователем процедура или функция драйвера должна удовлетворять приведенным выше требованиям, т.е. драйвер ConSt должен быть функцией типа BOOLEAN, драйвер ConIn должен быть функцией типа CHAR и т.д.

**Стандартные файлы**

В качестве альтернативы назначению текстовым файлам логических устройств Турбо-Паскаль предлагает несколько стандартных текстовых файлов, готовых к обработке, которым уже назначены соответствующие логические устройства. Таким образом, отпадает необходимость в открытии и закрытии таких файлов, кроме того, использование стандартных файлов уменьшает длину рабочей программы. Ниже приведен список стандартных файлов:

Input Первичный файл ввода. Этому файлу может быть назначено как устройство CON: , так и устройство TRM: (дальнейшие подробности см.ниже).

Output Первичный файл вывода. Этому файлу также может быть назначено как устройство CON: , так и устройство TRM: .

Con Назначено консольное устройство CON: .

Trm Назначено терминальное устройство TRM: .

Kbd Назначена клавиатура KBD: .

Lst Назначено устройство печати LST: .

Aux Назначено вспомогательное устройство AUX: .

Usr Назначено устройство пользователя USR: .

Заметим, что использование для этих файлов процедур ASSIGN, RESET, REWRITE, CLOSE не только нецелесообразно, но и не разрешается.

Если в вызове процедуры READ не указан идентификатор файла, то эта процедура всегда считывает строку, даже в том случае, если в буфере строки все еще остаются несколько знаков, которые должны быть считаны; причем эта процедура игнорирует CTRL-Z, так что оператор вынужден заканчивать строку нажатием клавиши RETURN. Нажатие на клавишу RETURN не сопровождается эхом, а во внутреннем представлении строка хранится с добавленным в ее конец управляющим символом CTRL-Z. Следовательно, если в строке ввода содержится меньше значений, чем параметров в списке при вызове процедуры READ, то все лишние переменные типа CHAR получат значение CTRL-Z, переменные типа STRING будут пустыми, а числовые переменные останутся неизменными.

Директива компилятора B управляет только что описанным свойством "принудительного чтения". По умолчанию подразумевается состояние {$B+} этой директивы, и в этом состоянии операторы READ без явно заданной файловой переменной всегда считывают строку с консоли. Если же в начале программы (перед разделами описаний) помещена директива {$B-}, то укороченная версия READ (без файловой переменной) будет работать так, как если бы был указан стандартный файл INPUT, т.е.

READ(X1,X2,..,Xn) равно READ (INPUT,X1,X2,..,Xn)

и в этом случае строки вводятся только тогда, когда буфер строки освобожден. Состояние {$B-} директивы компилятора B соответствует определению ввода-вывода в стандартном Паскале, а состояние {$B+}, подразумеваемое по умолчанию, хотя и не полностью соответствует стандарту, зато позволяет лучше управлять операциями ввода.

Эхо-отображение ввода на экран можно запретить: для этого нужно читать из стандартного файла KBD:

READ (Kbd,Ch)

Поскольку стандартные файлы INPUT и OUTPUT используются очень часто, они выбираются по умолчанию, когда при вызове процедуры или функции явно не указан идентификатор файла. В следующем списке приведены сокращенные версии операций с текстовыми файлами и их полные эквиваленты:

|  |  |
| --- | --- |
| Write(Ch) | Write(Output,Ch) |
| Read(Ch) | Read(Input,Ch) |
| WriteLn | WriteLn(Ooutput |
| ReadLn | ReadLn(Input) |
| Eof | Eof(Input) |
| Eoln | Eoln(Input) |

Следующая программа демонстрирует использование стандартного файла LST для распечатки на принтере файла PRODUCT.DTA (этот файл был создан программой из раздела "Применение файлов" главы 14).

program ListProductFile;

const

MaxNumberOfProducts = 100;

type

ProductName = string[20];

Product = record

Name: ProductName;

ItemNumber: Integer;

InStock: Real;

Supplier: Integer;

end;

var

ProductFile: file of Product;

ProductRec: Product;;

i: Integer;

begin

Assign (ProductFile,'PRODUCT.DTA');

Reset (ProductFile);

for i:=1 to MaxNumberOfProducts do

begin

Read(ProductFile,ProductRec);

with ProductRec do

begin

if Name<>'' then

Writeln(Lst,'Item: ',ItemNumber:5,' ',

Name:20,' From: ',Supplier:5,

' Now in stock: ',InStock:0:0);

end;

end;

Close (ProductFile);

end.

**Ввод и вывод текста**

Ввод и вывод данных в форме, пригодной для чтения, осуществляется с применением текстовых файлов в соответствии с тем, как это описано в разделе "Текстовые файлы" настоящей главы.

Текстовый файл может быть назначен на любое устройство, т.е. на дисковый файл или на одно из стандартных устройств ввода/вывода. Операции ввода и вывода для текстовых файлов выполняются с применением процедур READ, READLN, WRITE и WRITELN, списки параметров которых имеют особую форму, обеспечивающую максимальную гибкость при вводе/выводе.

В частности, параметры могут быть различных типов, в каждом случае процедуры ввода/вывода предусматривают автоматическое преобразование данных в базовый тип текстовых файлов CHAR и из него.

Если первый параметр процедуры ввода/вывода - это идентификатор переменной, представляющей текстовый файл, операции ввода/вывода выполняются над этим файлом, в противном случае операции выполняются над стандартными файлами INPUT и OUTPUT, что более подробно описано в разделе "Стандартные файлы" настоящей главы.

**Процедура READ**

Эта процедура дает возможность ввода символов, строк и числовых данных.

Синтаксис:

READ(Var1,Var2,..,VarN);

или

READ(FilVar,Var1,Var2,..,VarN);

где Var1,Var2,..,VarN - переменные типа CHAR, STRING, INTEGER или REAL. В первом случае переменные вводятся из стандартного файла INPUT, обычно с клавиатуры. Во втором случае переменные вводятся из текстового файла, который предварительно назначен переменной FilVar и подготовлен для чтения.

Для переменных типа CHAR эта процедура считывает один символ из файла и присваивает значение этого символа переменной.

Для дискового файла функция EOLN возвращает значение TRUE, если следующим символом является символ CR или CTRL-Z. Функция EOF возвращает значение TRUE, если следующий символ - CTRL-Z или достигнут физический конец файла. Если файл представляет собой логическое устройство (включая стандартные файлы INPUT и OUT-PUT), то функция EOLN возвращает значение TRUE, если прочитан символ CR или если функция EOF возвращает TRUE, а функция EOF возвращает TRUE, если прочитан символ CTRL-Z.

Для переменной типа STRING процедура READ читает столько символов, сколько допускает определенная максимальная длина строки, или до тех пор, пока не будет достигнут конец строки или конец файла. Функция EOLN возвращает значение TRUE, если прочитан символ CR или если функция EOF возвращает TRUE, а функция EOF возвращает TRUE, если прочитан символ CTRL-Z или встречен физический конец файла.

Для числовых переменных (INTEGER или REAL) процедура READ вводит строку символов, которая преобразуется в формат численной константы соответствующего типа. Все пробелы, символы табуляции, CR, LF, предшествующие строке, пропускаются. Строка не может быть длиннее, чем 30 символов, и за ней должен следовать ПРОБЕЛ или один из символов CR, табуляция, CTRL-Z. Если введенная строка не может быть преобразована в нужный формат, фиксируется ошибка ввода/вывода. Если преобразование выполнено успешно, численное значение, в которое была преобразована строка, присваивается указанной переменной. При чтении из дискового файла, если вводимая строка заканчивается пробелом или символом табуляции, при следующем вызове процедур READ или READLN чтение начнется с символа, непосредственно следующего за пробелом или символом табуляции. Как для исковых файлов, так и для логических устройств функция EOLN возвращает значение TRUE, если строка заканчивается символом CR или CTRL-Z, функция EOF возвращает значение TRUE, если строка заканчивается символом CTRL-Z.

ОСОБЫМ СЛУЧАЕМ ЧИСЛОВОГО ВВОДА является ситуация, когда EOLN или EOF имеют значение TRUE в самом начале чтения с помощью процедуры READ (например, когда с клавиатуры введен только символ CR). В этом случае никакое новое значение переменной не присваивается, и она сохраняет свое прежнее значение.

Если файлу ввода назначена консоль (устройство CON:), или если используется стандартный файл INPUT в режиме {$B+} (т.е. по умолчанию), то чтение переменных происходит по особым правилам. При вызове процедур READ и READLN вводимая с консоли строка запоминается в буфере, который затем используется как источник для чтения значений переменных. Это позволяет редактировать информацию в процессе ввода. Доступны следующие средства редактирования:

BACKSPACE, DEL Производится возврат на одну позицию в строке и уничтожение находящегося там символа. Действие BACKSPACE обычно генерируется при нажатии клавиши <-- или при нажатии CTRL-H. DEL обычно генерируется клавишей с такой же маркировкой или обозначением ЗАБОЙ.

ESC, Возврат к началу строки со стиранием всех

CTRL-X введенных символов. Клавиша ESC может быть обозначена как АР2.

CTRL-D Восстановление одного знака из последней введенной строки.

CTRL-R Восстановление всей последней введенной строки.

RETURN Эта клавиша используется для окончания ввода или строки. При этом признак конца строки (CR/LF)

CTRL-M заносится в буфер. Комбинация CR/LF не отображается эхом на экране. Указанная клавиша может быть обозначена как ВК или ВВОД.

CTRL-Z Завершение ввода строки и занесение маркера конца файла (CTRL-Z) в буфер строки.

Во внутреннем представлении строка хранится с добавленным в ее конец управляющим символом CTRL-Z. Поэтому, если количество значений, имеющихся в строке ввода, меньше количества переменных в списке параметров при вызове процедуры READ, то все лишние переменные типа CHAR получают значение CTRL-Z, переменные типа STRING (также лишние) становятся пустыми, а числовые переменные остаются неизменными.

Максимальное число символов, которое может содержать вводимая с консоли строка, по умолчанию равно 127. Однако можно уменьшить этот предел путем присвоения целого значения от 0 до 127 предопределенной переменной BUFLEN.

Пример:

Write('File name (max 14 chars): ');

BufLen:=14;

Read(FileName);

Заметим, что присвоенное переменной BUFLEN значение действует только на непосредственно следующий оператор READ. После этого значение BUFLEN восстанавливается равным 127.

**Процедура READLN**

Эта процедура идентична процедуре READ, за исключением того, что после ввода последней переменной все оставшиеся символы строки пропускаются, т.е. все символы до следующей последовательности CR/LF включительно (или до следующего символа CR на логическом устройстве).

Синтаксис:

READLN(Var1,Var2,..,VarN);

или

READLN(FilVar,Var1,Var2,..,VarN);

После вызова процедуры READLN следующий вызов одной из процедур READ или READLN приведет к чтению от начала следующей строки. Процедуру READLN можно вызывать без параметров:

READLN;

или

READLN(FilVar);

В этом случае остаток строки пропускается. При считывании с консоли (стандартный файл INPUT или файл, которому назначено устройство CON:) процедура READLN, в отличие от процедуры READ, отображает завершающий символ CR на экране в виде комбинации

CR/LF.

**Процедура WRITE**

Эта процедура осуществляет вывод символов, строк, булевских и числовых значений.

Синтаксис:

WRITE(Var1,Var2,..,VarN);

или

WRITE(FilVar,Var1,Var2,..,VarN);

Здесь Var1,Var2,..,VarN - это переменные типа CHAR, STRING, BOOLEAN, INTEGER или REAL, за каждой из которых может следовать двоеточие и целое выражение, определяющее ширину поля вывода. В первом случае переменные выводятся в стандартный файл OUTPUT -

обычно это экран. Во втором случае переменные выводятся в текстовый файл, ранее связанный с файловой переменной FilVar.

Параметры Var1,Var2,..,VarN называют параметрами вывода, формат которых зависит от типа переменной. Ниже приводится описание различных форматов и того, как они работают. При этом применяются следующие обозначения:

I,M,N - обозначают выражения типа INTEGER;

R - обозначает выражение типа REAL;

Ch - обозначает выражение типа CHAR;

S - обозначает выражение типа STRING;

B - обозначает выражение типа BOOLEAN.

Параметры вывода:

Ch - Выводится символ Ch.

Ch:N - Символ Ch выводится с правой стороны поля шириной N символов, т.е. символу Ch предшествует N-1 пробелов.

S - Выводится строка S. Массивы символов также могут выводиться, так как они совместимы со строками.

S:N - Строка S выводится с выравниванием по правой границе поля шириной N символов, т.е. строке S предшествует N-Length(S) пробелов.

B - В зависимости от значения выводится либо слово TRUE, либо слово FALSE.

B:N - В зависимости от значения выводится либо слово TRUE, либо слово FALSE, выровненное по правой границе поля шириной N символов.

I - Выводится десятичное представление значения I.

I:N - Выводится десятичное представление значения I, выровненное по правой границе поля шириной N символов.

R - Выводится десятичное представление значения R, выровненное справа в поле шириной 18 символов в формате с плавающей запятой:

R>=0.0: \_\_#.##########E\*##

R< 0.0: \_-#.##########E\*##

где знак \_ представляет пробел, # представляет цифру, \* представляет либо + , либо - .

R:N - Выводится десятичное представление значения R, выровненное по правой границе поля шириной N символов, в формате с плавающей запятой:

R>=0.0: Пробелы #.ЦифрыE\*##

R< 0.0: Пробелы-#.ЦифрыE\*##

где "Пробелы" - это 0 или более пробелов, "Цифры" - это от 1 до 10 цифр, # представляет цифру, \* представляет либо + , либо - . Поскольку после десятичной точки выводится по крайней мере одна цифра, минимальная ширина поля равна 7 символам (8, если R<0). Если N>16 (N>17 для R<0), числу будут предшествовать N-16 (N-17, если R<0) пробелов.

R:N:M - Выводится десятичное представление значения R, выровненное по правой границе поля шириной N символов в формате с фиксированной запятой с M цифрами после десятичной точки. Если M=0, то не выводится ни дробная часть, ни десятичная точка. Число M должно быть в диапазоне от 0 до 24, в противном случае будет использован формат с плавающей запятой. Числу предшествует необходимое количество пробелов таким образом, что ширина поля равна N символов.

**Процедура WRITELN**

Эта процедура идентична процедуре WRITE, за исключением того, что вслед за последним значением выводится последовательность CR/LF.

Синтаксис:

WRITELN(Var1,Var2,..,VarN);

или

WRITELN(FilVar,Var1,Var2,..,VarN);

Вызов процедуры WRITELN без параметров вывода приводит к выводу пустой строки, состоящей из последовательности CR/LF:

WRITELN;

или

WRITELN(FilVar);

**Файлы, не имеющие типа**

Файлы, не имеющие типа, (файлы без типа) являются средством ввода/вывода низкого уровня; они используются в первую очередь для прямого доступа к любому дисковому файлу с использованием записи длиной 128 байт.

При операциях ввода/вывода с файлом, не имеющим типа, производится обмен данными непосредственно между переменной и дисковым файлом; таким образом экономится область памяти, выделяемая в качестве буфера сектора для файла с заданным типом.

Переменная, являющаяся файлом, не имеющим типа, занимает меньше памяти, чем любая другая переменная типа FILE. Поскольку файл, не имеющий типа, не совместим ни с каким файлом, использование такого файла предпочтительно только тогда, когда связанная с ним переменная используется только в обращениях к процедурам ERASE, RENAME или в других операциях, не связанных с вводом/выводом.

Файл, не имеющий типа, объявляется зарезервированным словом

FILE:

var

DataFile: File;

**Процедуры BLOCKREAD и BLOCKWRITE**

Все процедуры и функции, применимые к стандартным файлам, кроме READ, WRITE, FLUSH, разрешается применять к файлам, не имеющим типа. Вместо процедур READ и WRITE имеются специальные высокоскоростные процедуры обмена: BLOCKREAD и BLOCKWRITE.

Синтаксис:

BLOCKREAD (FilVar,Var,Recs);

BLOCKWRITE(FilVar,Var,Recs);

или

BLOCKREAD (FilVar,Var,Recs,Result);

BLOCKWRITE(FilVar,Var,Recs,Result);

где FilVar - идентификатор переменной, связанной с файлом без типа, Var - это переменная, Recs - это целое выражение, определяющее число 128-байтных записей, подлежащих передаче между дисковым файлом и переменной, необязательный параметр Result содержит количество реально переданных записей. Передача начинается с первого байта, занимаемого переменной Var. Программист должен обеспечить, чтобы переменная Var занимала достаточно места для передачи данных. Вызов BLOCKREAD или BLOCKWRITE продвигает указатель файла на Recs записей вперед.

Файл, подлежащий обработке с помощью процедур BLOCKREAD и BLOCKWRITE, должен быть сначала подготовлен с помощью вызова процедуры ASSIGN и одной из процедур REWRITE или RESET. Процедура REWRITE создает и открывает новый файл, а процедура RESET открывает существующий файл. По окончании обработки файл необходимо закрыть с помощью процедуры CLOSE.

Стандартная функция EOF работает точно так же, как и с файлами, имеющими тип. Аналогично работают стандартные функции FILEPOS и FILESIZE, а также стандартная процедура SEEK, с использованием компонент длиной 128 байт.

Следующая программа демонстрирует применение файла, не имеющего типа. Она читает любой дисковый файл и копирует его содержимое в любой другой дисковый файл. Обратите внимание на использование необязательного четвертого параметра процедуры BLOCKREAD для контроля количества записей, реально считанных из исходного файла.

program FileCopy;

const

RecSize = 128;

BufSize = 200;

var

Sour,

Dest: File;

SourName,

DestName: string[14];

Buffer: array[1..RecSize,1..BufSize] of Byte;

RecsRead: Integer;

begin

Write ('Имя файла источника: ');

ReadLn (SourName);

Assign (Sour,SourName);

Reset (Sour);

Write ('Имя файла приемника: ');

ReadLn (DestName);

Assign (Dest,DestName);

Rewrite (Dest);

repeat

BlockRead (Sour,Buffer,BufSize,RecsRead);

BlockWrite(Dest,Buffer,RecsRead);

until RecsRead = 0;

Close (Sour);

Close (Dest);

end.

**Контроль ввода/вывода**

Директива компилятора I используется для управления генерацией кода, обнаруживающего в процессе выполнения ошибки ввода/вывода. По умолчанию этот режим включен, т.е. {$I+}, что приводит к вызову подпрограмм контроля ввода/вывода после каждой операции ввода/вывода. Ошибки ввода/вывода вызывают прекращение выполнения программы и выдачу сообщения, указывающего тип ошибки.

Если этот режим выключен, т.е. {$I-}, то во время работы программы проверка не выполняется. Ошибка ввода/вывода не вызывает при этом останова программы, однако все дальнейшие операции ввода/вывода должны быть приостановлены до тех пор, пока не будет вызвана стандартная функция IORESULT. При вызове этой функции состояние ошибки сбрасывается, и операции ввода/вывода могут выполняться снова. Программист должен сам выполнить соответствующие действия в зависимости от типа ошибки ввода/вывода.

Нуль, возвращаемый функцией IORESULT, свидетельствует об успешном выполнении последней процедуры ввода/вывода; любое другое значение означает, что последняя операция ввода/вывода завершилась с ошибкой. Приложение G содержит коды всех ошибок ввода/вывода и соответствующие им сообщения. Заметим, что, если состояние ошибки сброшено вызовом функции IORESULT, последующие вызовы функции IORESULT будут возвращать нуль до тех пор, пока не произойдет следующая ошибка ввода/вывода.

Функция IORESULT очень удобна в тех ситуациях, когда останов программы является неприемлемой реакцией на ошибку ввода/вывода, как в следующем примере, где продолжаются запросы имени файла до тех пор, пока попытка его открыть не окажется успешной (т.е. пока не будет введено имя уже существующего файла).

procudure OpenInFile;

var

InFileName: string[14];

InFile: File;

Ok: Boolean;

begin

repeat

Write ('Введите имя файла: ');

ReadLn (InFileName);

Assign (InFile,InFileName);

{$I-}

Reset (InFile);

{$I+}

Ok:=(IOresult=0);

if not Ok then

WriteLn ('Файл ',InFileName,' не найден !');

until Ok;

end.

Когда директива I отключена, т.е. {$I-}, результат выполнения следующих стандартных процедур нужно проверять с помощью вызова функции IORESULT, чтобы избежать возможных ошибок:

|  |  |  |  |
| --- | --- | --- | --- |
| Assign | Close | Read | Rewrite |
| BlockRead | Erase | ReadLn | Seek |
| BlockWrite | Execute | Rename | Write |
| Chain | Flush | Reset | Writeln |

**ГЛАВА 15.**

**УКАЗАТЕЛИ**

Переменные, которые рассматривались до этой главы, были СТАТИЧЕСКИМИ, т.е. их форма и размер были определены и они существовали в течение всего времени выполнения блока, в котором они были объявлены. Программы, однако, часто нуждаются в использовании структур данных, которые изменяют форму и размер в процессе выполнения. Для этой цели служат ДИНАМИЧЕСКИЕ переменные, поскольку они могут быть созданы, когда в них возникает необходимость и уничтожены по окончании использования.

Такие динамические переменные не объявляются с помощью явного описания переменной, как это имеет место со статическими переменными, и к таким переменным невозможен прямой доступ по идентификаторам. Вместо такого прямого доступа к переменной используется специальная переменная, содержащая адрес памяти, отведенной динамической переменной. Эта специальная переменная используется для ССЫЛКИ на динамическую переменную и называется УКАЗАТЕЛЕМ.

**Определение переменной-указателя**

Определение типа указателя состоит из двух следующих друг за другом элементов: символ указателя "^", затем идентификатор типа динамических переменных, на которые можно будет ссылаться с помощью переменных-указателей данного типа.

В следующем примере показано, как объявляется запись и ссылающийся на нее указатель. Тип PersonPointer объявлен как указатель на переменные типа PersonRecord:

type

PersonPointer = ^PersonRecord;

PersonRecord = record

Name: string[50];

Job: string[50];

Next: PersonPointer;

end;

var

FirstPerson,LastPerson,NewPerson: PersonPointer;

Переменные FirstPerson, LastPerson, NewPerson являются, таким образом, ПЕРЕМЕННЫМИ-УКАЗАТЕЛЯМИ, которые могут указывать на записи типа PersonRecord.

Как показано выше, идентификатор типа в определении типа указателя может быть определен позже.

**Размещение переменных (процедура NEW)**

Для того, чтобы применение указателей имело смысл, нужно, естественно, иметь какие то переменные, на которые эти указатели будут ссылаться. Для создания (размещения в памяти) новых переменных любого типа служит стандартная процедура NEW. Эта процедура имеет один параметр, который должен быть указателем на переменные того типа, который мы собираемся создавать.

Синтаксис:

NEW(PtrVar);

где PtrVar - имя переменной указателя.

Таким образом, новая переменная типа PersonRecord может быть создана оператором:

NEW(FirstPerson);

Действие этого оператора состоит в том, что после его выполнения переменная FirstPerson указывает на динамически размещенную в памяти запись типа PersonRecord.

Присваивания между переменными-указателями допустимы только в том случае, если оба указателя имеют одинаковый тип. Указатели идентичного типа можно также сравнивать с помощью операций отношений = и <> , которые возвращают результат типа BOOLEAN (TRUE или FALSE).

В качестве аналога нуля для указателей принято специальное значение NIL. Если переменная имеет значение NIL, то это означает, что она не указывает ни на одну переменную. Значение NIL совместимо со всеми типами указателей. Оно может быть использовано также в операциях сравнения.

Переменные, создаваемые стандартной процедурой NEW, размещаются в специальной структуре памяти, похожей на стек, которая носит название "ХИП" (heap - буквально "Куча"). Турбо-система управляет хипом с помощью указателя хипа, который в начале выполнения программы инициализируется адресом первого свободного байта памяти. При каждом вызове процедуры NEW указатель хипа продвигается по направлению к верхним адресам свободной памяти на число байт, соответствующее размеру новой динамической переменной.

**Процедуры MARK и RELEASE**

Если некоторая динамическая переменная больше не нужна программе, то для возврата памяти, занятой под эту переменную, можно воспользоваться стандартными процедурами MARK и RELEASE.

Процедура MARK присваивает некоторой переменной значение указателя хипа.

Синтаксис:

MARK(PtrVar);

где PtrVar является переменной типа указатель. Процедура RELEASE устанавливает указатель хипа на адрес, который содержится в ее аргументе.

Синтаксис:

RELEASE(PtrVar);

где PtrVar - переменная типа указатель, предварительно установленная с помощью процедуры MARK. Таким образом, процедура RELEASE удаляет все динамические переменные, расположенные "выше" этого адреса, и не в состоянии освобождать пространство, занятое переменными в средней части хипа. Поэтому, если нужно освободить память из средней части хипа, то вместо MARK и RELEASE нужно пользоваться процедурой DISPOSE, описанной далее.

**Функция MEMAVAIL**

Синтаксис: MEMAVAIL

Стандартная функция MEMAVAIL позволяет определить размер доступного пространства в хипе в любой момент. Результат имеет тип INTEGER, причем, если в хипе доступно более, чем 32767 байт, то функция MEMAVAIL возвращает отрицательное значение. При этом правильное число свободных байтов вычисляется как 65536.0+MEMAVAIL. Обратите внимание на применение константы типа REAL для получения результата типа REAL; это связано с тем, что результат превышает величину MAXINT.

**Использование указателей**

Предположим, что с помощью процедуры NEW мы создали последовательность экземпляров записей типа PersonRecord (см. предыдущий пример программы), и что поле Next внутри каждой записи указывает на следующую созданную запись. Тогда приведенная ниже совокупность операторов выполнит проход через весь список записей и вывод содержимого каждой записи (FirstPerson в начале указывает на первую запись в списке):

while FirstPerson<>nil do

with FirstPerson^ do

begin

WriteLn(Name,' имеет должность: ',Job);

FirstPerson:=Next;

end;

Обозначение FirstPerson^.Name можно воспринимать как поле Name в записи, на которую ссылается указатель FirstPerson.

Показанная ниже программа демонстрирует применение указателей для ведения списка фамилий и связанных с ними запросов на работу. Фамилии и запросы на работу вводятся до тех пор, пока вместо фамилии не будет введена пустая строка, после этого введенный список выводится на экран. Наконец, вслед за этим, память, занятая под список, освобождается для других целей. Переменная-указатель HeapTop используется для единственной цели - записи в нее и сохранения первоначального значения указателя хипа, поэтому ее определение в качестве INTEGER (указателя на целое) является совершенно произвольным.

program Jobs;

type

PersonPointer = ^PersonRecord;

PersonRecord = record

Name: string[50];

Job: string[50];

Next: PersonPointer;

end;

var

HeapTop: ^Integer;

FirstPerson,LastPerson,NewPerson: PersonPointer;

Name: string[50];

begin

FirstPerson:=nil;

Mark (HeapTop);

repeat

Write('Введите фамилию: ');

ReadLn(Name);

if Name<>'' then

begin

New(NewPerson);

NewPerson^.Name:=Name;

Write('Введите профессию: ');

ReadLn(NewPerson^.Job);

WriteLn;

NewPerson^.Next:=nil;

if FirstPerson=nil then

FirstPerson:=NewPerson

else

LastPerson^.Next:=NewPerson;

LastPerson:=NewPerson;

LastPerson^.Next:=nil;

end;

until Name='';

Writeln;

while FirstPerson<>nil do

with FirstPerson^ do

begin

WriteLn(Name,' - ',Job);

FirstPerson:=Next;

end;

Release(HeapTop);

end.

**Процедура DISPOSE**

Вместо процедур MARK/RELEASE для возврата памяти, полученной из хипа, обратно в хип можно воспользоваться процедурой стандартного Паскаля DISPOSE.

Примечание. Очень важно отдавать себе отчет в том, что процедуры DISPOSE и MARK/RELEASE используют совершенно разные подходы к управлению хипом, и НИ ПРИ КАКИХ ОБСТОЯТЕЛЬСТВАХ "ЭТИ ДВОЕ" НЕ ДОЛЖНЫ ВСТРЕЧАТЬСЯ ДРУГ С ДРУГОМ! Программа должна пользоваться для управления хипом либо процедурой DISPOSE, либо парой процедур MARK/RELEASE. Смешение этих двух подходов приведет к непредсказуемым результатам.

Вызов процедуры DISPOSE имеет следующий синтаксис:

DISPOSE(PtrVar);

где параметр PtrVar обозначает переменную-указатель.

Процедура DISPOSE освобождает (т.е. возвращает системе) динамическую память, занятую КОНКРЕТНОЙ УКАЗУЕМОЙ ПЕРЕМЕННОЙ. В этом состоит отличие этой процедуры от MARK/RELEASE, которые освобождают всю область хипа, НАЧИНАЯ С КОНКРЕТНОГО ЗНАЧЕНИЯ УКАЗАТЕЛЯ И ВЫШЕ.

Предположим, что у Вас имеется несколько переменных, память для которых была выделена из хипа. Следующая таблица иллюстрирует состояние хипа и действие вызовов DISPOSE (Var3) и MARK (Var3)/RELEASE (Var3):

|  |  |  |  |
| --- | --- | --- | --- |
|  | Хип | После  DISPOSE | После MARK/RELEASE |
|  | Var1 | Var1 | Var1 |
|  | Var2 | Var2 | Var2 |
|  | Var3 |  |  |
|  | Var4 | Var4 |  |
|  | Var5 | Var5 |  |
| HiMem | Var6 | Var6 |  |

Отсюда ясно, что после ряда вызовов процедуры DISPOSE хип может состоять из нескольких занятых областей памяти, перемежающихся свободными областями. При последующих вызовах система будет пытаться использовать эти свободные области, если среди них найдется подходящая по размеру для новой переменной.

**Процедура GETMEM**

Стандартная процедура GETMEM используется с целью выделения пространства в хипе. В отличие от процедуры NEW, которая выделяет столько пространства, сколько требуется для типа, на который указывает аргумент, процедура GETMEM позволяет программисту управлять количеством выделяемого пространства. Процедура GETMEM вызывается с двумя параметрами:

GETMEM(PtrVar,IntExp);

где PtrVar - это любая переменная типа указатель, IntExp - целое выражение, задающее число выделяемых байт.

**Процедура FREEMEM**

Стандартная процедура FREEMEM используется для возврата в хип целого блока памяти. В этом смысле она является партнером процедуры GETMEM. Эта процедура вызывается с двумя параметрами:

FREEMEM(PtrVar,IntExp);

где параметр PtrVar обозначает произвольную переменную-указатель, а параметр IntExp - целое выражение, задающее число возвращаемых байт, которое должно быть в ТОЧНОСТИ РАВНО числу байт, ранее выделенных этой переменной процедурой GETMEM.

**Функция MAXAVAIL**

Синтаксис: MAXAVAIL

Стандартная функция MAXAVAIL возвращает размер наибольшего непрерывного блока свободной памяти, имеющегося в хипе. Это пространство измеряется в байтах. Результат имеет тип INTEGER, и если в наличии имеется более 32767 байт, то MAXAVAIL возвращает отрицательное число. В этом случае корректное число свободных байт вычисляется как 65536.0+MAXAVAIL. Обратите внимание на использование вещественной константы для получения результата типа REAL, поскольку результат превышает значение MAXINT.

**Указатели и целые**

Стандартные функции обеспечивают непосредственный контроль адреса, который содержится в указателе; это функции ORD и PTR. Функция ORD возвращает адрес, содержащийся в ее аргументе-указателе как величину типа INTEGER, а PTR преобразует свой аргумент типа INTEGER в указатель, совместимый со всеми типами указателей.

Эти две функции превращаются в мощный инструмент в руках опытного программиста, поскольку они позволяют направить указатель на любую точку памяти. Однако при небрежном обращении они становятся весьма опасными, поскольку с их помощью можно затереть динамическими переменными другие переменные или даже код программы.

**Полезные советы**

Следует помнить, что для указателей не выполняется никакой проверки границ значений. Ответственность за то, чтобы указатель ссылался на допустимый адрес, возлагается на программиста.

Если у Вас возникают трудности с использованием указателей, то Вам может оказаться полезным такой прием: попробуйте нарисовать на бумаге то, что Вы пытаетесь сделать - это нередко проясняет ситуацию.

**ГЛАВА 16.**

**ПРОЦЕДУРЫ И ФУНКЦИИ**

Программа на языке Паскаль состоит из одного или более блоков, каждый из которых в свою очередь может также состоять из блоков и т.д. Одним из таких блоков является ПРОЦЕДУРА, еще одним - ФУНКЦИЯ (то и другое называют одним общим названием -ПОДПРОГРАММА). Итак, процедура есть отдельная часть программы, которая запускается из какого-либо другого места программы посредством ВЫЗОВА ПРОЦЕДУРЫ (см. подраздел "Вызов процедуры" в главе 7). Функция очень похожа на процедуру, но она вычисляет и возвращает значение всякий раз, когда ее идентификатор (или ВЫЗОВ) встречается во время выполнения (см. раздел "Вызовы функций" в главе 6).

**Параметры**

Процедурам и функциям могут быть переданы некоторые значения через ПАРАМЕТРЫ. Наличие параметров обеспечивает программиста механизмом подстановки, благодаря которому подпрограмму можно использовать с различными начальными значениями, тем самым получая различные результаты.

Оператор вызова процедуры или указатель функции, которые запускают подпрограмму, могут содержать список параметров, называемых ФАКТИЧЕСКИМИ ПАРАМЕТРАМИ. Они передаются через ФОРМАЛЬНЫЕ ПАРАМЕТРЫ, определяемые в ЗАГОЛОВКЕ подпрограммы. Паскаль допускает два различных метода передачи параметров: по ЗНАЧЕНИЮ и по АДРЕСУ, что определяет действие изменений значений формальных параметров на значения фактических параметров.

Если параметры передаются ПО ЗНАЧЕНИЮ, то формальный параметр представляется как переменная, локализованная внутри подпрограммы, и изменение значения формального параметра никак не влияет на фактический параметр. Фактическим параметром может быть произвольное выражение, в том числе - переменная такого же типа, что и соответствующий формальный параметр. Такие параметры называют ПАРАМЕТРАМИ-ЗНАЧЕНИЯМИ и объявляют в заголовке подпрограммы так, как это делается в следующем примере (этот и следующие примеры показывают заголовки процедур; описание заголовка функции вы найдете в этой же главе в разделе "Функции"):

procedure Example (Num1,Num2: Number; Str1,Str2: Txt);

где Number и Txt - это предварительно определенные типы (например, Integer и string[255]), Num1, Num2, Str1, Str2 - ФОРМАЛЬНЫЕ ПАРАМЕТРЫ, которым передаются значения фактических параметров. Типы формальных и фактических параметров должны соответствовать друг другу.

Обратите внимание на то, что тип параметров должен быть указан в заголовке только с помощью предварительно определенного идентификатора типа. Так, конструкция:

procedure Select (Model: array[1..500] of Integer);

является НЕДОПУСТИМОЙ. Вместо этого требуемый тип должен быть определен в разделе определений типов данного блока, а в описании параметров должен быть использован идентификатор этого типа:

type

Range = array[1..500] of Integer;

procedure Select (Model: Range);

Когда параметры передаются по АДРЕСУ, формальный параметр в действительности представляет фактический параметр при выполнении подпрограммы. Поэтому, все изменения, которым подвергается формальный параметр, отражаются на фактическом параметре, который, следовательно, должен быть переменной. Параметры, передаваемые по адресу, называются ПАРАМЕТРАМИ-ПЕРЕМЕННЫМИ и объявляются как показано ниже:

procedure Example (var Num1,Num2: Number);

Параметры-значения и параметры-переменные можно смешивать в одной процедуре, как например здесь:

procedure Example (var Num1,Num2: Number; Str1,Str2: Txt);

где Num1 и Num2 - параметры-переменные, а Str1,Str2 - параметры-значения.

Все вычисления адресов выполняются при вызове процедуры. Так, если переменная является компонентой массива, то ее индексные выражения вычисляются при вызове подпрограммы.

Заметим, что файл, являющийся параметром, всегда должен быть объявлен как параметр-переменная.

Когда большую структуру данных, такую как массив, нужно передать подпрограмме как параметр, использование параметра-переменной экономит как время выполнения, так и память, поскольку в этом случае подпрограмме передается лишь адрес фактического параметра. Параметры-значения требуют выделения памяти для копирования всей структуры данных, а также времени на это копирование.

**Ослабление ограничений на типы параметров**

Обычно при использовании параметров-переменных формальные и фактические параметры должны в точности соответствовать друг другу. Это означает, в частности, что подпрограмме, требующей параметр-переменную типа STRING может быть передана лишь строка той же длины, что и определенная в программе. Это ограничение можно обойти с помощью директивы V компилятора. По умолчанию установлен режим {$V+}, при котором требуется полное соответствие типов, в то время как задание {$V-} ослабляет это ограничение и допускает в качестве фактического параметра передавать строку любой длины, независимо от длины формального параметра.

Пример:

program Encoder;

{$V-}

type

WorkString = string[255];

var

Line1: string[80];

Line2: string[100];

procedure Encode (var LineToEncode: WorkString);

var

i: Integer;

begin

for i:=1 to Length(LineToEncode) do

LineToEncode[i]:=Chr(Ord(LineToEncode[i])-30);

end;

begin

Line1:='Секретное сообщение';

Encode (Line1);

Line2:='Другое (более длинное) секретное сообщение';

Encode (Line2);

end;

**Параметры-переменные, не имеющие типа**

Если тип формального параметра не определен, т.е. определение типа опущено в области параметров заголовка подпрограммы, такой параметр называется НЕ ИМЕЮЩИМ ТИПА. Соответствующий ему фактический параметр может быть любого типа.

Сам по себе формальный параметр без типа совместим с любым типом и может использоваться только в таких контекстах, где тип данных не имеет значения. Примерами этого могут быть параметры для ADDR, BLOCKREAD/BLOCKWRITE, FILLCHAR или MOVE или спецификация адреса для АБСОЛЮТНЫХ переменных.

Процедура SwithVar, приведенная в следующем примере, демонстрирует использование параметров, не имеющих типа. Она производит обмен содержимого переменных A1 и A2.

procedure SwithVar (var A1p,A2p; Size: Integer);

type

A = array[1..MaxInt] of Byte;

var

A1: A absolute A1p;

A2: A absolute A2p;

Tmp: Byte;

Count: Integer;

begin

for Count:=1 to Size do

begin

Tmp:=A1[Count];

A1[Count]:=A2[Count];

A2[Count]:=Tmp;

end;

end;

Предположим, что сделаны объявления:

type

Matrix = array [1..50,1..25] of Real;

var

TestMatrix,BestMatrix: Matrix;

Тогда процедура SwithVar может быть использована для обмена значениями между двумя матрицами:

SwithVar (TestMatrix,BestMatrix,SizeOf(Matrix));

**Процедуры**

Процедуры делятся на стандартные и определенные пользователем. Стандартные процедуры являются частью Турбо-системы и могут быть вызваны без объявления. Процедурам, определенным пользователем, можно присваивать имена стандартных процедур, но в этом случае стандартная процедура становится недоступной в области действия объявления одноименной процедуры, определенной пользователем.

**Описание процедуры**

Описание (объявление) процедуры состоит из заголовка процедуры, за которым следует блок, содержащий разделы описаний и операторную часть.

Заголовок процедуры состоит из зарезервированного слова PROCEDURE, за которым следует идентификатор, который становится именем процедуры, затем может следовать необязательный список формальных параметров (рассматривавшийся в разделе "Параметры" настоящей главы).

Примеры:

procedure LogOn;

procedure Position (X,Y: Intrger);

procedure Compute (var Data: Matrix; Scale: Real);

Разделы описаний в процедуре имеет такой же вид, как и в программе. Все идентификаторы, объявленные в списке формальных параметров и в разделе объявлений процедуры являются локальными для данной процедуры и для всех процедур внутри ее. В этом состоит понятие области действия идентификатора, вне которой он является неизвестным. Процедура может получить доступ к любой константе, типу, переменной, процедуре или функции, определенным во внешних по отношению к ней блоках.

Операторная часть определяет действия, которые должны быть выполнены процедурой и имеет форму составного оператора (см. подраздел "Составной оператор" в главе 7). Если идентификатор процедуры используется внутри операторной части этой же процедуры, такая процедура будет выполняться рекурсивно. При использовании рекурсии директива A компилятора должна быть выключена, т.е. {$A-} .

В следующем примере показана программа, использующая процедуру и передающая ей параметры. Поскольку фактические параметры, передаваемые процедуре, в некоторых случаях являются константами, формальные параметры должны обеспечивать передачу по значению.

program Box;

var

I: Integer;

procedure DrawBox(X1,Y1,X2,Y2: Integer);

var

i: Integer;

begin

GotoXY(X1,Y1);

for i:=X1 to X2 do write ('-');

for i:=Y1+1 to Y2 do

begin

GotoXY(X1,i); Write('!');

GotoXY(X2,i); Write('!');

end;

GotoXY(X1,X2);

for i:=X1 to X2 do Write('-');

end; { of procedure DrawBox }

begin

ClrScr;

for i:=1 to 5 do DrawBox(i\*4,i\*2,10\*i,4\*i);

DrawBox(1,1,80,23);

end;

Часто возникает необходимость, чтобы фактический параметр подвергался таким же изменениям, которым подвергается формальный параметр. В таких случаях используются параметры-переменные, например:

procedure Switch(var A,B: Integer);

var Tmp: Integer;

begin

Tmp:=A; A:=B; B:=Tmp;

end;

Когда эта процедура вызывается оператором Swith(I,J), произойдет обмен значений I и J. Если бы заголовок этой процедуры был объявлен по-другому:

procedure Switch(A,B: Integer);

т.е. с передачей параметров ПО ЗНАЧЕНИЮ, то вызов Switch(I,J) не изменил бы значения I и J.

**Стандартные процедуры**

В Турбо-Паскале имеется целый ряд стандартных процедур. Они делятся на следующие категории:

1) Процедуры обработки строк,

2) Процедуры для работы с файлами,

3) Процедуры для размещения динамических переменных,

4) Процедуры ввода/вывода.

Процедуры каждой из этих категорий были рассмотрены ранее в разделах с соответствующими названиями. В дополнение к ним имеется еще целый ряд стандартных процедур, которые доступны при условии, что для Вашего терминала были инсталлированы соответствующие команды (см. раздел "Инсталляция" в главе 1). Ниже дается перечень этих процедур, а также соответствующие описания.

**Экранные процедуры**

**Процедура CLREOL**

Синтаксис: CLREOL;

Эта процедура удаляет с экрана все символы, начиная от текущей позиции курсора включительно до конца строки без перемещения курсора.

**Процедура CLRSCR**

Синтаксис: CLRSCR;

Эта процедура очищает экран и устанавливает курсор в левый верхний угол экрана. При этом следует быть осторожным, ибо в некоторых дисплеях очистка экрана сопровождается сбрасыванием в начальное состояние всех видео-атрибутов, т.е. может привести к разрушению тех установок, которые сделал программист.

**Процедура CRTINIT**

Синтаксис: CRTINIT;

Эта процедура посылает СТРОКУ ИНИЦИАЛИЗАЦИИ ТЕРМИНАЛА, определенную в процедуре инсталляции экрана.

**Процедура CRTEXIT**

Синтаксис: CRTEXIT;

Эта процедура посылает СТРОКУ СБРОСА ТЕРМИНАЛА, определенную в процедуре инсталляции экрана.

**Процедура DELLINE**

Синтаксис: DELLINE;

Эта процедура удаляет строку, содержащую курсор, и сдвигает все расположенные под ней строки на одну строку вверх.

**Процедура INSLINE**

Синтаксис: INSLINE;

Эта процедура вставляет пустую строку в текущей позиции курсора. Все строки, расположенные ниже, сдвигаются на одну строку вниз, а последняя строка исчезает с экрана.

**Процедура GOTOXY**

Синтаксис: GOTOXY(Xpos,Ypos);

Эта процедура перемещает курсор в позицию экрана, заданную целыми выражениями Xpos (столбец) и Ypos (строка). Верхний левый угол соответствует позиции (1,1) - это позиция курсора HOME.

**Процедура LOWVIDEO**

Синтаксис: LOWVIDEO;

Экран получает видео-атрибут, определенный в процедуре инсталляции как "START OF LOW VIDEO", т.е. бледное свечение знаков.

**Процедура NORMVIDEO**

Синтаксис: NORMVIDEO;

Экран получает видео-атрибут, определенный в процедуре инсталляции как "START NORMAL VIDEO", т.е. "нормальное" свечение экрана.

**Специальные процедуры**

**Процедура DELAY**

Синтаксис: DELAY(Time);

Эта процедура генерирует цикл, который выполняется в течение приблизительно стольких миллисекунд, сколько указано в ее параметре Time. Параметр должен иметь тип INTEGER. Точное время может несколько меняться в разных операционных окружениях.

**Процедура EXIT**

Синтаксис: EXIT;

Эта процедура осуществляет выход из текущего исполняемого блока. Когда EXIT выполняется в подпрограмме, это приводит к возврату управления из подпрограммы в вызывающую программу. Когда EXIT выполняется в операторной части программы, он завершает выполнение программы. Вызов EXIT можно сравнить с оператором GOTO, который передает управление на метку, расположенную как раз перед словом END, закрывающим данный блок.

**Процедура HALT**

Синтаксис: HALT;

Эта процедура останавливает выполнение программы и возвращает управление операционной системе.

**Процедура MOVE**

Синтаксис: MOVE(Src,Trg,Cnt);

Эта процедура выполняет копирование больших блоков данных непосредственно в памяти. Объем копируемой области задается в байтах. Параметры Src и Trg представляют собой две переменные произвольного типа, а параметр Cnt есть выражение типа INTEGER. Процедура выполняет копирование блока данных, размер которого задается параметром Cnt (в байтах), начиная с первого байта, занятого переменной Src, в блок, начало которого совпадает с первым байтом переменной Trg. Обратите внимание на отсутствие специализированных процедур "Пересылка вправо" и "Пересылка влево". Это объясняется тем, что процедура MOVE автоматически обрабатывает возможное перекрытие областей копирования.

**Процедура FILLCHAR**

Синтаксис: FILLCHAR(Var,Cnt,Val);

Эта процедура заполняет область памяти заданным значением. Параметр Var - это переменная произвольного типа, параметр Cnt - выражение целого типа, а параметр Val - выражение типа BYTE или CHAR. Область памяти, которая начинается с первого байта, занятого Var, и занимает Cnt байт, заполняется значением параметра Val.

**Процедура RANDOMIZE**

Синтаксис: RANDOMIZE;

Эта процедура инициализирует генератор псевдослучайных чисел некоторым случайным значением.

**Функции**

Подобно процедурам, функции могут быть либо стандартными (предопределенными), либо определенные программистом.

**Описание функции**

Описание функции включает в себя ЗАГОЛОВОК функции и блок, который, в свою очередь состоит из разделов описаний и операторной части.

Заголовок функции эквивалентен заголовку процедуры, за исключением того, что первый должен определять тип результата функции. Это достигается путем добавления к заголовку двоеточия и типа, как например, здесь:

function KeyHit: Boolean;

function Compute (var Value: Simple): Real;

function Power (X,Y: Real): Real;

Тип результата функции должен быть простым (т.е. INTEGER, REAL, BOOLEAN, CHAR, объявленный простой или ограниченный тип), типом STRING или типом указателя.

Разделы описаний функции такие же, как и в процедуре.

Операторная часть функции представляет собой составной оператор (см. подраздел "Составной оператор" в главе 7). Внутри операторной части должен присутствовать хотя бы один оператор присваивания значения идентификатору функции. Последнее выполненное из таких присваиваний определяет результат функции. Если вызов функции появляется в операторной части этой же функции, то функция будет вызвана рекурсивно. При использовании рекурсии директива компилятора A должна быть отключена, т.е. {$A-}.

Следующий пример демонстрирует использование функции для вычисления суммы ряда целых чисел от 1 до J.

function RowSum (I,J: Integer): Integer;

function SimpleRowSum (S: Integer): Integer;

begin

SimpleRowSum:= S\*(S+1) div 2;

end;

begin

RowSum:= SimpleRowSum(J)-SimpleRowSum(I);

end;

В этом примере функция SimpleRowSum вложена в функцию RowSum. Поэтому первая функция является доступной только в области действия второй функции.

Ниже показана программа, которая является классическим примером применения рекурсивной функции для вычисления факториала числа.

{$A-}

program Factorial;

var

Number: Integer;

function Factorial (Value: Integer): Real;

begin

if Value = 0 then

Factorial:=1

else

Factorial:=Value \* Factorial(Value-1);

end;

begin

Read(Number);

WriteLn(^M,Number,'! = ',Factorial(Number));

end;

Обратите внимание на то, что тип, который используется в определении типа функции, должен быть ранее указан в качестве ИДЕНТИФИКАТОРА ТИПА. Поэтому такая конструкция НЕЗАКОННА:

function LowCase (Line: UserLine): string[80];

Вместо этого с типом STRING[80] следует связать идентификатор типа, и именно этот идентификатор использовать для определения типа результата функции, например так:

type

Str80 = string[80];

function LowCase (Line: UserLine): Str80;

Особенности реализации стандартных процедур WRITE и WRITELN таковы, что функция, использующая любую из стандартных процедур READ, READLN, WRITE, WRITELN, никогда не должна вызываться из выражения внутри оператора WRITE или WRITELN. Это же замечание справедливо и для стандартных процедур STR и VAL.

**Стандартные функции**

В Турбо-Паскале реализованы следующие стандартные (предопределенные) функции:

1) Функции для обработки строк (см. главу 9),

2) Функции для работы с файлами (см. главу 14),

3) Функции для работы с указателями (см. главу 15).

**Арифметические функции**

**Функция ABS**

Синтаксис: ABS(Num)

Эта функция возвращает абсолютную величину своего аргумента. Параметр Num должен иметь тип либо INTEGER, либо REAL; тип результата совпадает с типом аргумента.

**Функция ARCTAN**

Синтаксис: ARCTAN(Num)

Эта функция возвращает угол в радианах, тангенс которого равен значению параметра Num. Аргумент должен иметь тип INTEGER или REAL, а результат имеет тип REAL.

**Функция COS**

Синтаксис: COS(Num)

Эта функция возвращает косинус аргумента. Параметр Num выражается в радианах, и его тип должен быть INTEGER или REAL. Результат имеет тип REAL.

**Функция EXP**

Синтаксис: EXP(Num)

Эта функция возвращает значение экспоненты от аргумента Num. Параметр Num должен иметь тип INTEGER или REAL. Результат имеет тип REAL.

**Функция FRAC**

Синтаксис: FRAC(Num)

Эта функция возвращает дробную часть своего аргумента, т.е. FRAC(Num) = Num - INT(Num). Параметр Num должен иметь тип REAL или INTEGER, а результат имеет тип REAL.

**Функция INT**

Синтаксис: INT(Num)

Эта функция возвращает целую часть своего аргумента, т.е. наибольшее целое, которое меньше или равно параметру Num, если Num неотрицательно, или наименьшее целое, которое больше или равно параметру Num, если Num отрицательно. Параметр Num должен иметь тип INTEGER или REAL, а результат имеет тип REAL.

**Функция LN**

Синтаксис: LN(Num)

Эта функция возвращает натуральный логарифм своего аргумента. Параметр Num должен иметь либо тип REAL, либо INTEGER, а результат всегда имеет тип REAL.

**Функция SIN**

Синтаксис: SIN(Num)

Эта функция возвращает синус своего аргумента. Параметр Num выражается в радианах и должен иметь тип REAL или INTEGER. Результат всегда имеет тип REAL.

**Функция SQR**

Синтаксис: SQR(Num)

Эта функция возвращает квадрат своего аргумента, т.е. Num \* Num = SQR(Num). Параметр Num должен иметь тип REAL или INTEGER, а тип результата совпадает с типом параметра.

**Функция SQRT**

Синтаксис: SQRT(Num)

Эта функция возвращает квадратный корень из своего аргумента. Параметр Num должен иметь тип REAL или INTEGER, а тип результата всегда REAL.

**Функции для работы с простыми типами**

**Функция PRED**

Синтаксис: PRED(X)

Эта функция возвращает значение, являющееся предшественником значения параметра X в последовательности значений соответствующего типа, если такое значение существует. Типом X может быть любой простой тип, кроме REAL.

**Функция SUCC**

Синтаксис: SUCC(X)

Эта функция возвращает значение, следующее за значением параметра X в последовательности значений соответствующего типа, если такое значение существует. Типом X может быть любой простой тип, кроме REAL.

**Функция ODD**

Синтаксис: ODD(X)

Эта функция возвращает булевское значение TRUE, если X есть четное число, и FALSE в противном случае. Параметр X должен иметь тип INTEGER.

**Функции преобразования типов**

Эти функции используются для преобразования значений одного простого типа в значения другого простого типа. В дополнение к функциям, описанным ниже, этой же цели служит средство преобразования типов, описанное в разделе "Преобразование типов" главы 8.

**Функция CHR**

Синтаксис: CHR(Num)

Эта функция возвращает в качестве результата символ, порядковый номер которого определяется целочисленным выражением Num. Например, CHR(65) возвращает символ 'A'.

**Функция ORD**

Синтаксис: ORD(X)

Эта функция возвращает порядковый номер значения параметра X во множестве значений, определяемом типом этого параметра. ORD(X) эквивалентно INTEGER(X) (см. раздел "Преобразование типов" в главе 8). Тип параметра X может быть любым простым типом, за исключением REAL, а результат всегда имеет тип INTEGER.

**Функция ROUND**

Синтаксис: ROUND(X)

Эта функция возвращает значение своего аргумента X, округленное до ближайшего целого следующим образом: если X неотрицательно, то ROUND(X) = TRUNC(X+0.5), а если X отрицательно, то ROUND(X) = TRUNC(X-0.5). Параметр X должен иметь тип REAL, а результат имеет тип INTEGER.

**Функция TRUNC**

Синтаксис: TRUNC(X)

Эта функция возвращает наибольшее целое, меньшее или равное значению параметра X, если X неотрицательно, и наименьшее целое, большее или равное значению параметра X, если X отрицательно. Параметр X должен иметь тип REAL, а результат имеет тип INTEGER.

**Прочие стандартные функции**

**Функция ADDR**

Синтаксис: ADDR(Name)

Эта функция возвращает адрес первого байта типа, переменной или подпрограммы с идентификатором, заданным параметром Name. Если Name является идентификатором массива, то его можно индексировать, если идентификатором записи, то можно указывать конкретное поле. Возвращаемое значение имеет тип INTEGER.

**Функция HI**

Синтаксис: HI(Num)

Младший байт результата содержит старший байт значения целого выражения Num. Старший байт результата равен нулю. результат имеет тип INTEGER.

**Функция KEYPRESSED**

Синтаксис: KEYPRESSED

Эта функция возвращает булевское значение TRUE, если на клавиатуре была нажата клавиша, и FALSE в противном случае. Результат достигается с помощью вызова подпрограммы опроса статуса клавиатуры, которая имеется в операционной системе.

**Функция LO**

Синтаксис: LO(Num)

Эта функция возвращает младший байт значения целого выражения Num, при этом старший байт результата устанавливается в ноль. Результат имеет тип INTEGER.

**Функция RANDOM**

Синтаксис: RANDOM

RANDOM(Num)

При отсутствии параметра эта функция возвращает псевдослучайное число в интервале от 0 до 1. Результат имеет тип REAL.

Если функция с параметром, она возвращает псевдослучайное число в интервале от нуля до Num. Как параметр Num, так и результат имеют тип INTEGER.

**Функция PARAMCOUNT**

Синтаксис: PARAMCOUNT

Эта целочисленная функция возвращает в качестве результата количество параметров, переданных программе в буфере командной строки. Пробелы и знаки табуляции служат разделителями.

**Функция PARAMSTR**

Синтаксис: PARAMSTR(N)

Эта строковая функция возвращает в качестве результата N-й параметр из буфера командной строки.

**Функция SIZEOF**

Синтаксис: SIZEOF(Name)

Эта функция возвращает количество байтов, которое занимает в памяти переменная Name или количество байтов, необходимых для величины типа Name. Результат имеет тип INTEGER.

**Функция SWAP**

Синтаксис: SWAP(Num)

Эта функция меняет местами старший и младший байты своего параметра Num и возвращает полученное значение как целое.

Пример:

SWAP($1234) возвращает $3412 (**для большей ясности значения показаны в 16-ном виде**)

**Функция UPCASE**

Синтаксис: UPCASE(Chr)

Эта функция возвращает символ - эквивалент из верхнего регистра для символа, заданного параметром Chr, который должен иметь тип CHAR. Если такого эквивалента не существует, то значение Chr возвращается без изменения.

**Вызовы функций операционной системы**

В целях обращения к подсистемам BDOS и BIOS операционной системы CP/M-80 в Турбо-Паскале имеются две стандартные процедуры BDOS и BIOS, и четыре стандартные функции BDOS, BDOSHL, BIOS, BIOSHL.

Подробности о подпрограммах подсистем BDOS и BIOS Вы сможете найти в "Руководстве по операционной системе CP/M" фирмы DIGITAL RESEARCH.

**Процедура и функция BDOS**

Синтаксис: BDOS(Func,Parm)

Процедура BDOS применяется для запуска подпрограммы BDOS (базовой дисковой операционной системы). Целое выражение Func определяет номер вызываемой подпрограммы и загружается в регистр C. Целое выражение Parm является необязательным и задает параметр, который загружается в регистровую пару DE. После этого вызов по адресу 0005 активизирует BDOS.

Функция BIOS вызывается аналогично процедуре с тем же именем и возвращает значение типа INTEGER, которое представляет собой результат, возвращаемый подсистемой BDOS в регистре A.

**Функция BDOSHL**

Синтаксис: BDOSHL(Func,Parm)

Эта функция в точности подобна описанной выше функции BDOS, за исключением того, что возвращаемый результат здесь определяется значением пары регистров HL.

**Процедура и функция BIOS**

Синтаксис: BIOS(Func,Parm)

Процедура BIOS применяется для запуска подпрограмм BIOS (базовой системы ввода/вывода). Целое выражение Func определяет номер вызываемой подпрограммы, причем 0 обозначает подпрограмму WBOOT, 1 - подпрограмму CONST и т.д. Это означает, что адрес вызываемой подпрограммы равен (Func)\*3 + (адрес WBOOT), а адрес WBOOT находится в байтах с адресами 0001 и 0002. Целое выражение Parm является необязательным и задает параметр, который перед вызовом загружается в пару регистров BC.

Функция BIOS вызывается точно так же, как и процедура с тем же именем, и возвращает значение типа INTEGER, которое представляет собой результат, возвращаемый подсистемой BIOS в регистре A.

**Функция BIOSHL**

Синтаксис: BIOSHL(Func,Parm)

Эта функция в точности подобна только что описанной функции BIOS, за исключением того, что возвращаемый результат определяется значением пары регистров HL.

**Предварительные объявления**

Предварительное объявление подпрограммы - это отделение заголовка этой подпрограммы от ее блока. Отделенный таким образом заголовок подпрограммы ничем не отличается от обычного заголовка, за исключением того, что этот заголовок оканчивается зарезервированным словом FORWARD. Блок подпрограммы размещается по тексту ниже в разделе описаний. Обратите внимание на то, что блок начинается копией заголовка, в котором на этот раз содержится только имя подпрограммы, и не содержится никаких параметров, типов и проч.

Пример:

program Catch22;

var

X: Integer;

function Up(Var I: Integer): Integer; forward;

function Down (Var I: Integer): Integer;

begin

I := I div 2; Writeln(I);

if I <> 1 then I := Up(I);

end;

function Up;

begin

while I mod 2 <> 0 do

begin

I := I\*3+1; Writeln(I);

end;

I := Down(I);

end;

begin

Write('Введите любое целое число: ');

Readln(X);

X := Up(X);

Write('Все в порядке. Программа опять остановилась');

end;

Если, запустив эту программу, Вы введете, например, число 6, то будет выведено следующее:

3

10

5

16

8

4

2

1

Все в порядке. Программа опять остановилась.

Показанная только что программа в действительности является усложненной версией следующей программы:

program Catch222;

var

X: Integer;

begin

Write('Введите любое целое число: ');

Readln(X);

while X <> 1 do

begin

if X mod 2 = 0 then X := X div 2 else X := X\*3+1;

Writeln(X);

end;

Write('Все в порядке. Программа опять остановилась');

end;

Возможно, Вам будет небезынтересно узнать, что невозможно доказать, что эта маленькая и очень простая программа в действительности остановится для любого целого числа!

**Внешние подпрограммы**

Для объявления внешних процедур и функций (обычно это процедуры и функции, написанные на машинном коде) используется зарезервированное слово EXTERNAL.

Для внешней подпрограммы не существует блока, т.е. в ней нет разделов описаний и операторной части. Указывается только заголовок подпрограммы, за которым непосредственно следует зарезервированное слово EXTERNAL и целая константа, задающая адрес данной подпрограммы в памяти:

procedure DiskReset; external $EC00;

function IOstatus: Boolean; external $D123;

Внешним подпрограммам можно передавать параметры, причем синтаксис этой передачи в точности совпадает с синтаксисом передачи параметров для обычных процедур и функций, например:

procedure Plot(X,Y: Integer); external $F003;

procedure QuickSort(var List: PartNo); external $1C00;

Дальнейшее обсуждение вопроса о передаче параметров внешним подпрограммам отложим до главы 22.

**ГЛАВА 17.**

**ВКЛЮЧЕНИЕ ФАЙЛОВ В КОМПИЛЯЦИЮ**

Тот факт, что редактор системы Турбо выполняет редактирование только в памяти, ограничивает размер исходных текстов программ. Директива компилятора I может быть использована для того, чтобы обойти это ограничение, поскольку она делает возможным разбиение исходного текста на части и обработку их в нужной последовательности при компиляции. Помимо этого, средство подключения позволяет достичь большей ясности и читабельности программ, поскольку часто используемые подпрограммы, отлаженные ранее, могут храниться в "библиотеке" файлов, из которой могут включаться в любую другую программу по мере необходимости.

Синтаксис директивы компилятора I:

{$I FileName}

где FileName - это любое допустимое имя файла. Пробелы игнорируются, а строчные буквы преобразуются в соответствующие заглавные. Если не указан тип файла, по умолчанию принимается тип .PAS . Эта директива должна быть записана в отдельной строке программы.

Примеры:

{$I first.pas}

{$I COMPUTE.MOD}

{$I StdProc }

Обратите внимание на то, что когда файл не имеет трехбуквенного расширения (типа), между именем файла и закрывающей фигурной скобкой должен ставиться пробел; в противном случае фигурная скобка будет воспринята как часть имени файла.

Чтобы продемонстрировать использование средства подключения файлов предположим, что в Вашей "библиотеке" часто используемых процедур и функций имеется файл с именем STUPCASE.FUN. Пусть он содержит функцию STUPCASE, которая вызывается с символом или строкой в качестве параметра и возвращает значение этого параметра, в котором все строчные буквы заменены на соответствующие заглавные.

Содержимое файла STUPCASE.FUN:

function StUpCase(St:AnyString):AnyString;

var

I: Integer;

begin

for I:=1 to Length(St) do

St[I]:=UpCase(St[I]);

StUpCase:=St;

end;

Если в программе, которую Вы пишете, нужно преобразовать строку указанным образом, Вам нужно только подключить файл STUPCASE.FUN для компиляции, вместо того, чтобы переписывать его текст:

program IncludeDemo;

type

InData = string[80];

AnyString = string[255];

var

Answer: InData;

{$I STUPCASE.FUN}

begin

ReadLn(Answer);

WriteLn(StUpCase(Answer));

end;

Достоинство этого метода не только в том, что он прост и позволяет экономить память, но и в том, что он делает процесс обновления программ более быстрым и безопасным, поскольку любое изменение в "библиотечной" подпрограмме автоматически появится и во всех программах, включающих в себя эту подпрограмму.

Следует отметить, что Турбо-Паскаль допускает произвольный порядок и даже многократное вхождение различных разделов описаний. Вы можете поэтому иметь несколько файлов, содержащих различные часто используемые определения типов и подключать их к различным программам.

Все директивы компилятора, за исключением B и C, являются локальными по отношению к файлу, в котором они появляются, т.е. если во включаемом файле некоторой директивой установлено значение, отличное орт того, которое она имеет в основном файле, то после возврата в этот основной (включающий) файл будет восстановлено первоначальное значение этой директивы. Директивы B и C являются глобальными. Приложение C содержит описание директив компилятора.

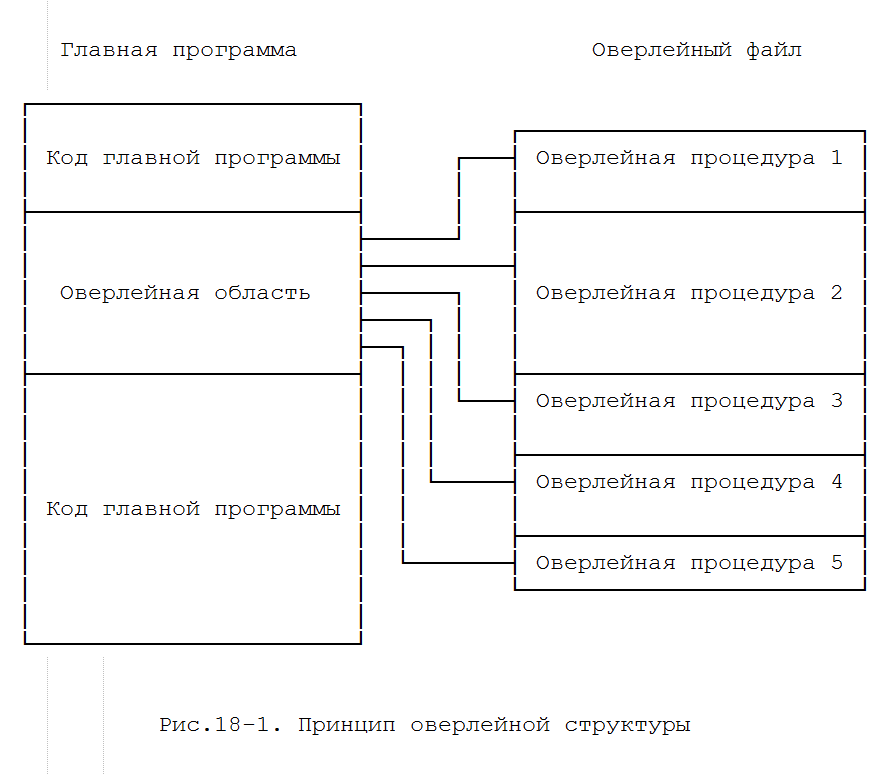
Включаемые файлы не могут быть вложенными, т.е. подключаемый файл не может содержать директиву подключения другого файла с последующим продолжением обработки.

**ГЛАВА 18.**

**ОВЕРЛЕЙНЫЕ СТРУКТУРЫ**

Система поддержки оверлейных структур (структур с перекрытиями) позволяет создавать программы, размеры которых значительно превосходят величину оперативной памяти машины. Основной принцип состоит в том, чтобы объединить несколько подпрограмм (процедур и функций) в одном или нескольких файлах отдельно от файла главной программы с тем, чтобы впоследствии автоматически загружать эти файлы по одному в ту же самую область памяти.

Рис.18-1 иллюстрирует программу, которая использует один оверлейный файл с пятью оверлейными подпрограммами, собранными в одну ОВЕРЛЕЙНУЮ ГРУППУ. Эти подпрограммы разделяют одну и ту же область памяти в главной программе.



Когда вызывается оверлейная процедура, она автоматически загружается в ОВЕРЛЕЙНУЮ ОБЛАСТЬ, зарезервированную в главной программе. Этот промежуток должен быть достаточно большим, чтобы вместить самый большой из оверлеев, входящих в данную оверлейную группу. Таким образом, память, требуемая главной программе, уменьшается приблизительно на сумму размеров всех подпрограмм данной оверлейной группы за вычетом размера самой большой из этих подпрограмм.

В последнем примере оверлейная процедура 2 является самой большой из всех пяти процедур, поэтому именно она определяет размер оверлейной области в коде главной программы. При загрузке в память эта процедура занимает всю оверлейную область.

Подпрограммы меньшего размера загружаются в ту же самую область, причем каждая из них начинается (после загрузки) с самого первого адреса оверлейной области. Остаток оверлейной области не используется.

Поскольку процедуры 1,3,4 и 5 выполняются в том же месте, что и процедура 2, легко видеть, что они не требуют дополнительной памяти в главной программе. Также очевидно, что ни одна из этих процедур никогда не должна вызывать другую из той же оверлейной группы, поскольку они никогда не присутствуют в памяти одновременно.

В данной группе оверлеев могло бы быть гораздо больше оверлейных процедур; в действительности, суммарный размер оверлейных процедур может значительно превышать размер главной программы. И тем не менее, для этих подпрограмм все равно потребуется лишь столько оперативной памяти, сколько занимает самая большая из них.

Цена, которую придется платить за эту дополнительную память для кода программы, состоит в том, что работа программы замедляется на время, требуемое для загрузки каждой процедуры с диска. При хорошем планировании оверлеев эти задержки могут быть пренебрежимо малыми.

**Создание оверлеев**

Оверлейные подпрограммы создаются автоматически - для этого нужно просто добавить к описанию любой процедуры или функции слово OVERLAY.

Примеры:

overlay procedure Initialize;

overlay function TimeOfDay: Time;

Встретив такое описание, компилятор прекращает выводить код в главный файл, а вместо этого выводит его в отдельный оверлейный файл. Имя этого файла будет таким же, как и имя файла главной программы, а его типом будет номер, идентифицирующий оверлейную группу - число от 000 до 999.

Последовательно расположенные оверлейные подпрограммы группируются вместе. то означает, что все оверлейные подпрограммы, между которыми нет никаких других объявлений, принадлежат к одной группе и помещаются в один оверлейный файл.

Пример 1.

overlay procedure One;

begin

:

:

end;

overlay procedure Two;

begin

:

:

end;

overlay procedure Three;

begin

:

:

end;

Эти три оверлейные процедуры будут сгруппированы вместе и помещены в один оверлейный файл. Если полученная группа является первой оверлейной группой в данной программе, то оверлейный файл получит номер .000 .

Три оверлейные процедуры из следующего примера будут помещены в последовательные оверлейные файлы .000 и .001 , поскольку оверлейные процедуры Two и Three разделены объявлением не оверлейной процедуры Count.

В качестве разделяющего объявления может быть использовано любое объявление. Например, для принудительного разделения оверлейных областей можно использовать фиктивное объявление TYPE.

Пример 2.

overlay procedure One;

begin

:

:

end;

overlay procedure Two;

begin

:

:

end;

procedure Count;

begin

:

:

end;

overlay procedure Three;

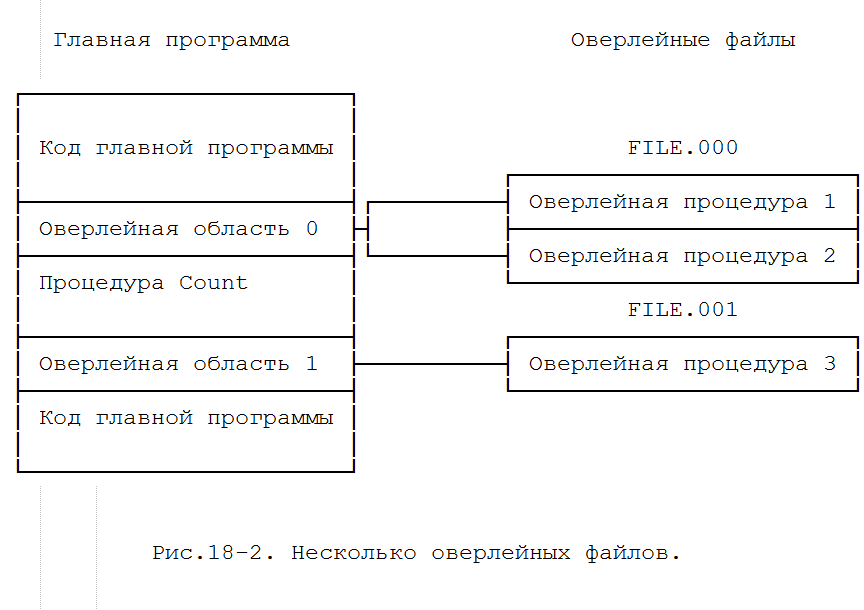
begin

:

:

end;

Для каждой оверлейной группы в коде главной программы резервируется отдельная область; кроме того, будут созданы следующие файлы: FILE.000 и FILE.001 (см. Рис.18-2).



**Вложенные оверлейные структуры**

Оверлейные подпрограммы могут быть вложенными, т.е. одна оверлейная подпрограмма может содержать внутри себя другую оверлейную подпрограмму, которая сама содержит оверлейные подпрограммы и т.д.

Пример 3.

program OverlayDemo;

:

:

overlay procedure One;

begin

:

:

end;

overlay procedure Two;

overlay procedure Three;

begin

:

:

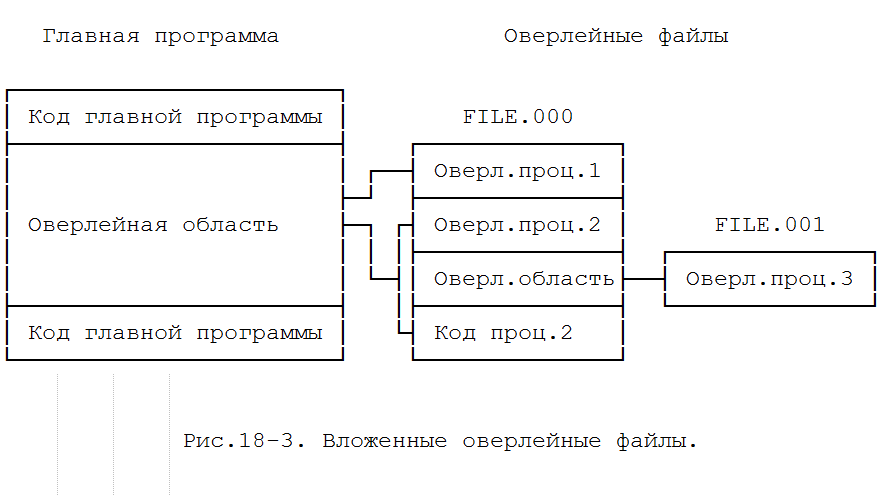
end;

begin

:

end;

В этом примере будут созданы два оверлейных файла. FILE.000 содержит оверлейные процедуры One и Two, а в главной программе будет зарезервирована область, достаточная для вмещения максимальной из них. Оверлейный файл FILE.001 содержит оверлейную процедуру Three, которая является локальной по отношению к оверлейной процедуре Two, так что оверлейная область для нее создается в коде оверлейной процедуры Two (см.Рис.18-3).



**Автоматическое управление оверлеями**

Оверлейная подпрограмма загружается в память только тогда, когда к ней происходит обращение (т.е. в момент вызова). При каждом вызове оверлейной подпрограммы система прежде всего проверяет, не присутствует ли уже сейчас эта подпрограмма в оверлейной области. Если подпрограмма в оверлейной области отсутствует, то выполняется автоматическое считывание этой подпрограммы из соответствующего оверлейного файла.

**Размещение оверлейных файлов**

Во время компиляции оверлейные файлы размещаются на текущем дисководе, т.е. на том дисководе, где располагается главный программный файл (.COM).

Во время выполнения система обычно предполагает, что оверлейные файлы также находятся на текущем дисководе. Для изменения этой установки можно воспользоваться процедурой OVRDRIVE.

Синтаксис: OVRDRIVE(Disk);

Здесь Disk - целое выражение, задающее дисковод (0 - текущий, 1 - дисковод A: , 2- B: и т.д.). При последующих обращениях к оверлейным файлам эти файлы будут отыскиваться на указанном дисководе.

Пример:

program OvrTest;

overlay procedure ProcA;

begin

WriteLn('Оверлей A');

end;

overlay procedure ProcB;

begin

WriteLn('Оверлей B');

end;

procedure Dummy;

begin

{ Пустая процедура для разделения файлов на группы }

end;

overlay procedure ProcC;

begin

WriteLn('Оверлей C');

end;

begin

OvrDrive(2);

ProcA;

OvrDrive(0);

ProcC;

OvrDrive(2);

ProcB;

end.

Первый вызов OVRDRIVE(2) предписывает, чтобы система искала оверлеи на дисководе B: . Следовательно, вызов ProcA приводит к открытию на этом дисководе первого оверлейного файла, который содержит две оверлейные процедуры ProcA и ProcB.

Следующий вызов OVRDRIVE(0) указывает, чтобы система искала дальнейшие оверлеи на текущем дисководе. Вызов ProcC приводит к открытию второго оверлейного файла именно на этом дисководе.

Далее мы видим оператор ProcB; он вызывает оверлейную процедуру, которая находится в первом оверлейном файле, и для того, чтобы обеспечить поиск этого файла на дисководе B:, перед этим вызовом необходимо выполнить оператор OVRDRIVE(2).

**Эффективное использование оверлеев**

Естественно, что применение оверлейной структуры влечет за собой определенные накладные расходы: в программе появляется дополнительный код для управления оверлеями, а при выполнении появляются дополнительные обращения к диску. Отсюда следует, что оверлеи требуют тщательного планирования.

Для того, чтобы не приводить к существенным задержкам, оверлейная подпрограмма не должна вызываться слишком часто; те же подпрограммы, которые все-таки требуют частого вызова, должны по крайней мере вызываться без вклинивания вызовов других подпрограмм из того же оверлейного файла - это позволит свести к минимуму обращение к диску. Разумеется, дополнительное время будет различным в зависимости от конкретной конфигурации диска. 5,25-дюймовая дискета приведет к серьезному увеличению времени выполнения, жесткий диск - к значительно меньшим издержкам, и совсем немного увеличит время выполнения широко используемый в последнее время RAM-квазидиск.

Для максимальной экономии памяти в главной программе одна группа оверлеев должна содержать как можно больше отдельных подпрограмм. С точки зрения чистой экономии памяти, чем больше подпрограмм Вы объедините в один оверлейный файл, тем лучше. Это связано с тем, что для оверлейной области в главной программе достаточно выделить память, равную размеру максимальной из этих подпрограмм, поэтому остальные подпрограммы смогут разместиться в этой области совершенно свободно. Между этими соображениями и рассмотренными выше временными издержками должен быть установлен разумный компромисс.

**Ограничения на оверлейные структуры**

**Область данных**

Оверлейные подпрограммы одной группы разделяют одну и ту же область оперативной памяти и, следовательно, не могут присутствовать одновременно. Это, в частности, означает, что они не могут вызывать одна другую. Отсюда следует, что они могут разделять одну и ту же область данных, что является еще одной прибавкой к экономии памяти с помощью оверлеев.

В примере 1 из раздела "Создание оверлеев" ни одна из процедур не может вызвать ни одну другую. А вот в примере 2 оверлейные процедуры One и Two могут вызвать процедуру Three, и оверлейная процедура Three может вызвать любую из первых двух процедур, поскольку они находятся в разных файлах и, следовательно, в разных оверлейных областях главной программы.

**Предварительные объявления**

Для оверлейных подпрограмм недопустимы предварительные объявления FORWARD. Это ограничение, однако, нетрудно обойти, если сделать предварительное объявление обычной подпрограммы, которая, в свою очередь, будет вызывать оверлейную подпрограмму.

**Рекурсия**

Оверлейные подпрограммы не могут быть рекурсивными. Это ограничение также можно обойти путем объявления обычной подпрограммы, которая сама вызывает оверлейную подпрограмму.

**Ошибки периода выполнения**

Ошибки, возникающие во время выполнения в оверлеях, не обнаруживаются обычным путем; при этом система обработки ошибок генерирует некоторый адрес. Однако этот адрес соответствует некоторой ячейке внутри оверлейной области, и не существует способа определить, какая из оверлейных подпрограмм была в действительности активной в момент возникновения ошибки.

Поэтому ошибки периода выполнения в оверлеях не всегда легко обнаруживаются с помощью средства "Поиск ошибок периода выполнения" из меню опций компилятора. Единственное, что таким путем может быть получено, это первое появление кода по указанному адресу. Безусловно, ошибка может находиться в этом месте, но с таким же успехом она может быть в одной из последующих подпрограмм той же самой оверлейной группы.

Тем не менее, это не является серьезным ограничением, поскольку тип ошибки и характер ее возникновения в большинстве случаев позволяют определить, в какой подпрограмме она произошла. После этого для точной локализации ошибки можно поместить подозрительную подпрограмму на первое место в оверлейной группе. В этом случае средство поиска ошибок периода выполнения сработает правильно.

Наилучшая стратегия состоит в том, чтобы не помещать подпрограммы в оверлеи до тех пор, пока они не будут ПОЛНОСТЬЮ ОТЛАЖЕНЫ.

**ГЛАВА 19.**

**ПРОЦЕДУРЫ CHAIN и EXECUTE**

В Турбо-Паскале имеются две процедуры, посредством которых одни программы на Турбо-Паскале могут активизировать другие программы на Турбо-Паскале. Синтаксис вызова этих процедур таков:

CHAIN(FilVar);

EXECUTE(FilVar);

Здесь FilVar - переменная произвольного типа файла, предварительно назначенная на дисковый файл процедурой ASSIGN. Если соответствующий файл существует, он загружается в память и выполняется.

Процедура CHAIN используется для активизации только особых .CHN-файлов, т.е. программ на Турбо-Паскале, скомпилированных с H-опцией, выбранной из меню опций. Такой файл содержит только код программы и не содержит библиотеку Паскаля. Он загружается в память и исполняется с начального адреса текущей программы, т.е. с того адреса, который был указан при компиляции текущей программы. Затем этот загруженный файл использует библиотеку Паскаля, уже присутствующую в памяти. Следовательно, текущая программа и программа, загруженная из текущей посредством CHAIN, должны использовать один и тот же начальный адрес.

Процедуру EXECUTE можно использовать для активизации произвольного .COM-файла, т.е. любого файла, который содержит исполняемый код. Это может быть файл, созданный Турбо-Паскалем с опцией C из меню опций. Файл загружается и выполняется с адреса 0100H (стандарт CP/M-80).

Если указанного дискового файла нет, то возникает ошибка ввода-вывода. При пассивном состоянии I-директивы компилятора: {$I-} выполнение программы продолжается с оператора, следующего за ошибочным оператором CHAIN или EXECUTE, и перед продолжением операций ввода/вывода необходимо вызвать функцию IORESULT.

Для передачи данных из текущей программы в ту, которую текущая программа запускает по CHAIN, можно воспользоваться РАЗДЕЛЯЕМЫМИ ГЛОБАЛЬНЫМИ ПЕРЕМЕННЫМИ или ПЕРЕМЕННЫМИ С АБСОЛЮТНЫМИ АДРЕСАМИ.

Чтобы обеспечить перекрытие, разделяемые глобальные переменные должны объявляться как самые первые переменные в обеих программах, причем порядок их появления в обоих описаниях должен совпадать. более того, обе программы должны быть скомпилированы с одинаковым размером памяти (см. описание начального и конечного адреса в меню опций). При выполнении этих условий переменные в обеих программах будут занимать одинаковые адреса памяти, а поскольку Турбо-Паскаль не выполняет автоматической инициализации своих переменных, эти переменные могут совместно использоваться двумя программами.

Пример:

Программа MAIN.COM:

program Main;

var

Txt: string[80];

CntPrg: File;

begin

Write('Введите любой текст: '); ReadLn(Txt);

Assign(CntPrg,'CHRCOUNT.CHN');

Chain(CntPrg);

end.

Программа CHRCOUNT.CHN:

program ChrCount;

var

Txt: string[80];

NoOfChar,

NoOfUpC,

I: Integer;

begin

NoOfUpC:=0;

NoOfChar:=Length(Txt);

for I:=1 to Length(TxT) do

if Txt[I] in ['A'..'Z'] then

NoOfUpC:=Succ(NoOfUpC);

Write('Число введенных символов: ',NoOfChar);

WriteLn('. Число заглавных букв: ',NoOfUpC,'.');

end.

Для того, чтобы определить, как была запущена Турбо-программа - непосредственно из командной строки CP/M-80 или процедурой EXECUTE из другой программы, нужно воспользоваться абсолютной переменной по адресу $0080. Это байт длины командной строки, и если программа запускается из CP/M, он содержит значение от 0 до 127. Следовательно, при запуске программы посредством процедуры EXECUTE запускающая программа должна занести в эту переменную значение, превышающее 127. Таким образом, вызываемая программа, проверив эту переменную и обнаружив в ней значение от 0 до 127, будет знать, что ее запустила операционная система, а значение, превышающее 127, будет означать, что эта программа была активизирована некоторой другой программой на Турбо-Паскале.

Следует помнить, что ни CHAIN, ни EXECUTE нельзя использовать в режиме непосредственного исполнения, т.е. в программе, которая скомпилирована с опцией M компилятора.

**ГЛАВА 20.**

**ВКЛЮЧЕНИЕ МАШИННОГО КОДА В ИСХОДНЫЙ ТЕКСТ**

В Турбо-Паскале имеется оператор INLINE, который предоставляет программисту очень удобный способ включения машинного кода непосредственно в исходный текст программы. Этот оператор состоит из зарезервированного слова INLINE, за которым следует один или более ЭЛЕМЕНТОВ КОДА, разделенных косыми чертами и заключенных в круглые скобки.

Элемент кода строится из одного или более элементов данных, соединенных знаками "+" и "-". В качестве элемента данных может выступать целая константа, идентификатор переменной, процедуры, функции или ссылка на значение счетчика адреса. Ссылка на значение счетчика адреса записывается в виде символа "\*" .

Пример:

INLINE(10/$2345/Count+1/Sort-\*+2);

Каждый элемент кода порождает один байт или одно слово (два байта) кода. Значение этого байта или слова вычисляется путем сложения или вычитания соответствующих элементов данных согласно соединяющим их знакам. Значением идентификатора переменной является адрес этой переменной. Значением идентификатора процедуры или функции является адрес этой процедуры или функции. Значением ссылки на счетчик адреса является адрес, по которому должен генерироваться очередной байт кода.

Элемент кода порождает один байт кода, если он состоит только из целых констант, и если его значение находится внутри диапазона 8-разрядных значений (0..255). Если полученное значение выходит за рамки 8-разрядного диапазона или если в элементе кода участвуют идентификаторы переменных, процедур, функций или ссылки на значение счетчика адреса, то порождается одно слово кода (младший по значению байт идет ПЕРВЫМ).

Для отмены действия только что описанного автоматического определения размера можно воспользоваться знаками "<" и ">". Если элемент кода начинается со знака "<" , то кодируется только младший по значению байт полученной величины, даже если эта величина - 16-разрядная. Если элемент кода начинается со знака ">" , то всегда кодируется слово, даже если старший по значению байт равен нулю.

Пример:

INLINE(<$1234/>$44);

Этот оператор генерирует три байта кода: $34, $44, $00.

Ниже показан пример оператора INLINE, который генерирует машинный код для преобразования всех символов своего строкового аргумента в заглавные.

procedure UpperCase(var Strg: Str); {Str - тип string[255]}

{$A+}

begin

inline ($2A/Strg/ { LD HL,(Strg); HL:= адрес Strg }

$46/ { LD B,(HL) ; B := длина Strg }

$04/ { INC B ; B := B+1 }

$05/ { L1:DEC B ; B := B-1 }

$CA/\*+20/ { JP Z,L2 ; Если конец -> L2 }

$23/ { INC HL ; К след.символу }

$7E/ { LD A,(HL) ; A := след.символ}

$FE/$61/ { CP 'a' ; A < 'a' ? }

$DA/\*-9/ { JP C,L1 ; Если да -> L1 }

$FE/$7B/ { CP 'z'+1 ; A > 'z' ? }

$D2/\*+14/ { JP NC,L1 ; Если да -> L1 }

$D6/$20/ { SUB 20H ; Отнять 20H }

$77/ { LD (HL),A ; Занести в Strg }

$C3/\*-20); { JP L1 ; Продолжить }

end; { L2:EQU $ ; Выход }

Операторы INLINE можно свободно смешивать с другими операторами на протяжении всей операторной части блока, причем в операторах INLINE могут использоваться все регистры микропроцессора. Отметим однако, что содержимое регистра SP (указателя стека) должно оставаться на выходе таким же, каким оно было на входе.

**ГЛАВА 21.**

**ОБРАБОТКА ПРЕРЫВАНИЙ**

При необходимости процедуры обслуживания прерываний могут быть написаны на Паскале. Такие процедуры всегда должны компилироваться при активном состоянии A-директивы компилятора. Эти процедуры не должны иметь параметров и должны любыми средствами обеспечивать сохранение всех используемых регистров. Это достигается включением в самое начало процедуры оператора INLINE с соответствующими командами микропроцессора PUSH (втолкнуть в стек), а в конец процедуры - оператора INLINE с соответствующими командами POP (вытолкнуть из стека). Последней командой завершающего оператора INLINE должна быть команда EI ($FB), которая разрешает дальнейшие прерывания. При использовании сцепленных прерываний в конце процедуры может быть также указана команда RETI ($ED, $4D), которая заменяет собой команду RET, генерируемую компилятором.

Процедура обработки прерываний не должна задействовать никакие операции ввода/вывода, использующие стандартные процедуры и функции Турбо-Паскаля. Отметим также, что вызовы BDOS, а в некоторых случаях и вызовы BIOS не должны осуществляться в процедурах обработки прерываний.

Программист может запрещать или разрешать вызов подпрограммы обработки прерываний посредством команд EI ($FB) и DI ($F3), генерируя их с помощью оператора INLINE.

Если задействован режим прерываний 0 (IM 0) то на программиста возлагается ответственность за использование ячеек рестарта в базовой странице (отметим, что RST 0 использовать нельзя, поскольку CP/M-80 использует ячейки с 0000 по 0007).

Если задействован режим прерываний 2 (IM 2), то программист должен сгенерировать инициализированную таблицу переходов (массив целых) по некоторому абсолютному адресу и установить в регистр I значение, соответствующее началу процедуры (посредством оператора INLINE).

**ГЛАВА 22.**

**ВНУТРЕННИЕ ФОРМАТЫ ДАННЫХ**

В последующих описаниях символ @ обозначает адрес первого байта, занимаемого переменной заданного типа. Чтобы получить это значение для данной переменной, можно воспользоваться стандартной функцией ADDR.

**Базовые типы данных**

Базовые типы данных могут группироваться в структуры (массивы, записи, дисковые файлы), однако это структурирование никак не влияет на их внутренние форматы.

**Простые типы**

Следующие простые типы при хранении занимают один байт: BYTE, отрезки типа INTEGER с обеими границами внутри диапазона 0..255, BOOLEAN, CHAR и объявленные простые типы с числом возможных значений не более 256. Байт внутреннего представления содержит порядковое значение переменной.

Следующие простые типы занимают при хранении два байта: INTEGER, отрезки типа INTEGER, у которого хотя бы одна граница выходит за пределы диапазона 0..255, а также объявленные простые типы с числом возможных значений более 256. Эти два байта содержат 16-разрядное значение в виде двоичного дополнения, причем младший по значению байт хранится в памяти первым.

**Данные типа REAL**

Данные типа REAL занимают в памяти 6 байт и представляют собой значение с плавающей точкой, состоящее из 40-разрядной мантиссы и 8-разрядного двоичного порядка. Порядок хранится в первом байте, а мантисса - в следующих пяти байтах, причем младший по значению байт мантиссы хранится первым. Это выглядит следующим образом:

@ Порядок

@+1 Младший по значению байт мантиссы

......................................

@+5 Старший по значению байт мантиссы

Для представления порядка используется двоичный формат со смещением $80. Следовательно, порядок, равный $84, означает, что значение мантиссы должно умножаться на 2^($84-$80) = 2^4 = 16. Если порядок равен нулю, то значение с плавающей точкой считается равным нулю.

Значение мантиссы получается посредством деления 40-разрядного беззнакового целого на 2^40. Мантисса всегда является нормализованной, т.е. старший по значению разряд мантиссы всегда должен интерпретироваться как 1 (это разряд 7 пятого байта). Поэтому в этом разряде хранится знак мантиссы, причем 1 означает, что число отрицательное, а 0 - положительное.

**Строки**

Число байтов памяти, занимаемых строкой, равно ее длине плюс единица. Первый байт внутреннего представления содержит текущую длину строки. Остальные байты содержат символы строки, причем код первого символа хранится в ячейке с наименьшим адресом. В показанной ниже таблице L обозначает длину строки, а Max - ее максимальную длину.

@ Текущая длина строки (L)

@+1 Первый символ

@+2 Второй символ

.......................

@+L Последний символ

@+L+1 Не используется

.......................

@+Max Не используется

**Множества**

Каждый элемент множества занимает один бит, а поскольку максимальное число элементов множества равно 256, переменная-множество никогда не может занимать в памяти больше 32 байтов (256/8).

Если множество содержит менее 256 элементов, то некоторые биты должны быть все время равны нулю, и поэтому они не требуют хранения. Отсюда следует, что в смысле экономии памяти лучшим способом хранения переменной-множества заданного типа было бы "отсечение" всех незначащих битов и циклический сдвиг всех оставшихся, так чтобы первому элементу множества соответствовал первый бит первого байта. Однако такие операции циклического сдвига являются весьма медленными, и Турбо-Паскаль предлагает поэтому следующий компромисс: не подлежат хранению только те байты, которые статически равны нулю, т.е. те байты, из которых не используется ни одного бита. Такой метод сжатия работает очень быстро и в большинстве случаев оказывается столь же экономным в отношении памяти, как и метод циклического сдвига.

Число байтов памяти, занимаемых переменной-множеством, вычисляется следующим образом:

(Max div 8) - (Min div 8) + 1

где Max и Min - это верхняя и нижняя границы базового типа данного множества. Адрес памяти для конкретного элемента E вычисляется следующим образом:

MemAddress = @ + (E div 8) - (Min div 8)

Адрес внутри байта по адресу MemAddress вычисляется так:

BitAddress = E mod 8

где E обозначает порядковое значение данного элемента множества.

**Интерфейсные блоки файлов**

Ниже показан формат интерфейсного блока файла (FIB):

@ Байт флагов

@+1 Буфер символа

@+2 Указатель буфера сектора (младший байт)

@+3 Указатель буфера сектора (старший байт)

@+4 Число записей (младший байт)

@+5 Число записей (старший байт)

@+6 Длина записи (младший байт)

@+7 Длина записи (старший байт)

@+8 Текущая запись (младший байт)

@+9 Текущая запись (старший байт)

@+10 Не используется

@+11 Не используется

@+12 Первый байт FCB CP/M

........................

@+47 Последний байт FCB CP/M

@+48 Первый байт буфера сектора

........................

@+175 Последний байт буфера сектора

Байт флагов по адресу @+0 имеет следующий формат:

биты 0..3 Тип файла

бит 4 Семафор чтения

бит 5 Семафор записи

бит 6 Флаг вывода

бит 7 Флаг ввода

Тип файла 0 обозначает дисковый файл, а типы файла от 1 до 5 обозначают логические устройства Турбо-Паскаля (CON:, KBD:, LST: , AUX: и USR:). Для файлов, обладающих типом (в смысле элементов файла) бит 4 устанавливается, если содержимое буфера сектора неопределенно, а бит 5 устанавливается, если в буфер сектора были записаны данные. Для текстовых файлов бит 5 устанавливается, если буфер символа содержит предварительно считанный символ. Бит 6 устанавливается, если разрешен вывод, а бит 7 устанавливается, если разрешен ввод.

Указатель буфера сектора содержит смещение (0..127) внутри буфера сектора, расположенного по адресу @+48. Как для файлов с типом, так и для файлов без типа три слова с @+4 до @+9 содержат число записей в файле, длину записи в байтах и номер текущей записи. В интерфейсном блоке для файла без типа нет буфера сектора, поэтому указатель буфера сектора в этом случае не используется.

Когда текстовый файл назначается на логическое устройство, в интерфейсном блоке файла используются только байт флагов и буфер символа.

**Указатели**

Указатель состоит из двух байтов, содержащих 16-разрядный адрес памяти, и хранится в памяти в побайтном "реверсированном" формате, когда младший по значению байт располагается в памяти первым. Значение NIL представляется нулевым словом.

**Структуры данных**

Структуры данных строятся из базовых типов данных с использованием различных методов композиции. Имеются три таких метода композиции: массивы, записи и дисковые файлы. Структурирование данных никоим образом не влияет на внутреннее представление входящих в них базовых типов данных.

**Массивы**

Компоненты массива с наименьшими значениями индекса занимают в памяти наименьшие адреса. Многомерные массивы хранятся так, что самый правый индекс возрастает быстрее всего. Например, массив

Board: array [1..8,1..8] of Square

будет храниться в памяти следующим образом:

наименьший адрес: Board[1,1]

Board[1,2]

..........

Board[1,8]

Board[2,1]

Board[2,2]

..........

наибольший адрес: Board[2,8]

**Записи**

Первое поле записи хранится в памяти с наименьшими адресами. Если запись не содержит вариантных частей, то ее длина определяется как сумма длин отдельных полей. Если же запись содержит варианты, то общее число байтов, занимаемых записью, определяется как длина фиксированной части плюс длина максимального варианта. Каждый вариант начинается с одного и того же адреса памяти.

**Дисковые файлы**

Отличие дисковых файлов от других структур данных состоит в том, что данные хранятся не во внутренней памяти, а в файле на внешнем устройстве. Управление дисковым файлом осуществляется посредством интерфейсного блока файла (FIB), описанного выше. В основном все дисковые файлы делятся на две категории: файлы произвольного доступа и текстовые файлы.

**Файлы произвольного доступа**

Файл произвольного доступа состоит из последовательности записей одинаковой длины и одинакового внутреннего формата. Для оптимального использования внешней памяти записи файла записываются непрерывно. Первые четыре байта первого сектора файла содержат число записей в файле и длину записи в байтах. Первая запись файла хранится, начиная с четвертого байта:

сектор 0, байт 0 Число записей (младший байт)

сектор 0, байт 1 Число записей (старший байт)

сектор 0, байт 2 Длина записи (младший байт)

сектор 0, байт 3 Длина записи (старший байт)

**Текстовые файлы**

Базовыми компонентами текстового файла являются символы, однако в нем существуют и более крупные структурные единицы - ТЕКСТОВЫЕ СТРОКИ (или просто строки). Каждая строка состоит из произвольного числа символов и заканчивается комбинацией CR/LF (в коде ASCII $0D/$0A). Файл заканчивается управляющим символом CTRL-Z (в коде ASCII $1A).

**Параметры**

Передача параметров процедурам и функциям осуществляется через стек Z-80. Обычно все это не представляет интереса для программиста, поскольку машинный код, генерируемый Турбо-Паскалем, автоматически заталкивает параметры в стек перед вызовом подпрограммы и выталкивает их из стека в начале подпрограммы. Однако, если программист собирается использовать ВНЕШНИЕ подпрограммы, то такие подпрограммы должны выталкивать параметры из стека самостоятельно.

При входе во внешнюю подпрограмму верхушка стека всегда содержит адрес возврата (слово). Параметры, если они имеются, располагаются в стеке "под" адресом возврата, т.е. имеют в стеке более высокие адреса. Следовательно, для доступа к параметрам подпрограмма должна сначала вытолкнуть из стека адрес возврата, затем все параметры, и наконец после этого она должна восстановить адрес возврата, втолкнув его обратно в стек (для выталкивания из стека используется команда POP, а для вталкивания - команда PUSH).

**Параметры-переменные**

При передаче параметра-переменной в стек заносится слово, которое определяет абсолютный адрес памяти первого байта, занимаемого фактическим параметром.

**Параметры-значения**

При передаче по значению данные, которые заносятся в стек, зависят от типа фактического параметра. Подробнее об этом Вы узнаете из следующих подразделов.

**Простые типы**

Данные типов INTEGER, BOOLEAN, CHAR и объявленных простых типов передаются в стек в виде одного слова. Если переменная занимает при хранении только один байт, то старший по значению байт параметра заполняется нулем. Обычно слово выталкивается из стека с помощью команды наподобие POP HL.

**Данные типа REAL**

При занесении в стек данных типа REAL используется шесть байтов. Если эти байты выталкиваются из стека с помощью следующей последовательности команд:

POP HL

POP DE

POP BC

то регистр L будет содержать порядок, регистр H – пятый (младший по значению) байт мантиссы, регистр E - четвертый байт, регистр D - третий байт, регистр C - второй байт и регистр B - первый (старший по значению) байт мантиссы.

**Строки - данные типа STRING**

Когда строка находится на верхушке стека, байт, на который указывает указатель стека SP, содержит длину строки. Байты с адресами SP+1,..,SP+n (где n - длина строки) содержат собственно строку, причем первый знак располагается по наименьшему адресу. Ниже показаны команды, с помощью которых можно вытолкнуть строку из стека и поместить ее в StrBuf, при условии, что строка находится на верхушке стека.

LD DE,StrBuf

LD HL,0

LD B,H

ADD HL,SP

LD C,(HL)

INC BC

LDIR

LD SP,HL

**Множества**

Множество всегда занимает в стеке 32 байта (сжатие информации осуществляется только при записи данных в множества и при чтении из множеств). Ниже показана последовательность команд, с помощью которой можно вытолкнуть из стека множество, находящееся на его верхушке, и поместить его в SetBuf.

LD DE,SetBuf

LD HL,0

ADD HL,SP

LD BC,32

LDIR

LD SP,HL

При этом младший по значению байт множества попадет в байт с наименьшим адресом области SetBuf.

**Указатели**

Значение указателя передается в стек в виде слова, которое содержит адрес памяти динамической переменной. Значению NIL соответствует нулевое слово.

**Массивы и записи**

Даже при использовании в качестве параметров-значений массивы и записи в действительности не заносятся в стек. Вместо этого в стек передается слово, содержащее адрес первого байта фактического параметра. После этого вызываемая подпрограмма выталкивает это слово из стека и использует его в качестве адреса операнда-источника в операции копирования блока данных.

**Результаты функций**

Написанные пользователем внешние (EXTERNAL) функции должны возвращать свои результаты в точном соответствии с тем, как изложено ниже:

1) Значения простых типов (кроме REAL) должны возвращаться в паре регистров HL. Если тип результата занимает во внутреннем представлении 1 байт, то этот байт должен возвращаться в регистре L, а регистр H должен содержать нуль;

2) Значения типа REAL должны возвращаться в парах регистров BC, DE и HL. Регистры B, C, D, E и H должны содержать мантиссу (старший по значению байт в регистре B), а регистр L должен содержать порядок;

3) Данные типа STRING и множества должны возвращаться на верхушке стека в формате, принятом для параметров-значений и рассмотренном выше;

4) Значения указателей должны возвращаться в паре регистров HL.

**Хип и стеки**

Как следует из схем распределения памяти, показанных в главе 23, во время выполнения программы исполняющая система Турбо-Паскаля поддерживает три стекоподобных структуры данных:

ХИП, СТЕК ЦП и СТЕК РЕКУРСИИ.

Хип используется для хранения динамических переменных и управляется стандартными процедурами NEW, MARK и RELEASE. В начале выполнения программы указатель хипа HeapPtr устанавливается на адрес "дна" свободной памяти, т.е. на первый байт после объектного кода.

Стек ЦП используется для хранения промежуточных результатов при вычислении выражений и для передачи параметров процедурам и функциям. Активный оператор FOR также пользуется стеком ЦП и занимает в нем одно слово. В начале выполнения программы указатель стека ЦП StackPtr устанавливается на адрес верхней границы свободной памяти.

Стек рекурсии используется только рекурсивными процедурами и функциями, т.е. процедурами и функциями, скомпилированными при пассивном состоянии A-директивы компилятора ( {$A-} ). При входе в рекурсивную подпрограмму осуществляется копирование ее рабочей области в стек рекурсии (на его верхушку), а при выходе из нее вся рабочая область восстанавливается в ее первоначальное состояние. Значение, которое подразумевается по умолчанию для RecurPtr в начале выполнения программы, соответствует адресу, который на 1 Кбайт ($400 байт) ниже указателя стека ЦП.

Вследствие такой техники работы переменные, локальные для подпрограммы, не должны использоваться в качестве VAR-параметров в рекурсивных вызовах.

Следующие стандартные переменные позволяют программисту контролировать состояние хипа и стеков:

HeapPtr указатель хипа

RecurPtr указатель стека рекурсии

StackPtr указатель стека ЦП

Эти переменные имеют тип INTEGER. Обратите внимание на то, что переменные HeapPtr и RecurPtr могут использоваться в тех же контекстах, что и любая другая переменная типа INTEGER. Что же касается переменной StackPtr, то ее можно употреблять только в присваиваниях и выражениях.

При манипулировании этими переменными всегда должны выполняться следующие условия:

1) Они должны указывать на адреса в пределах свободной памяти;

2) Должно выполняться соотношение:

HerapPtr < RecurPtr < StackPtr.

Невыполнение этих условий может привести к непредсказуемым, а возможно и к фатальным результатам.

Нет необходимости говорить о том, что никакие присваивания значений указателя хипа и стека не должны выполняться, если стеки и хип находятся в работе.

При каждом вызове процедуры NEW и при входе в рекурсивную процедуру или функцию система контролирует наличие коллизии между хипом и стеком рекурсии, т.е. проверяет, не превысил ли (или не сравнялся ли) HeapPtr значения RecurPtr. Если это так, то имеет место коллизия, что приводит к ошибке времени выполнения.

Отметим, что никогда не выполняется никаких проверок пересечения областей стека ЦП и стека рекурсии. Для того, чтобы такое произошло, рекурсивная подпрограмма должна вызвать сама себя примерно 300-400 раз; предполагается, что такая ситуация является весьма редкой. Если, тем не менее, в программе необходима именно такая вложенность, то с помощью показанного ниже оператора, выполненного в начале блока программы, можно сдвинуть указатель стека рекурсии вниз для увеличения размера стека ЦП:

RecurPtr := StackPtr - 2 \* MaxDepth - 512;

где MaxDepth - это максимальная требуемая глубина вложенности вызовов рекурсивных подпрограмм. Примерно 512 дополнительных байтов нужны в качестве резерва для передачи параметров и для промежуточных результатов при вычислении выражений.

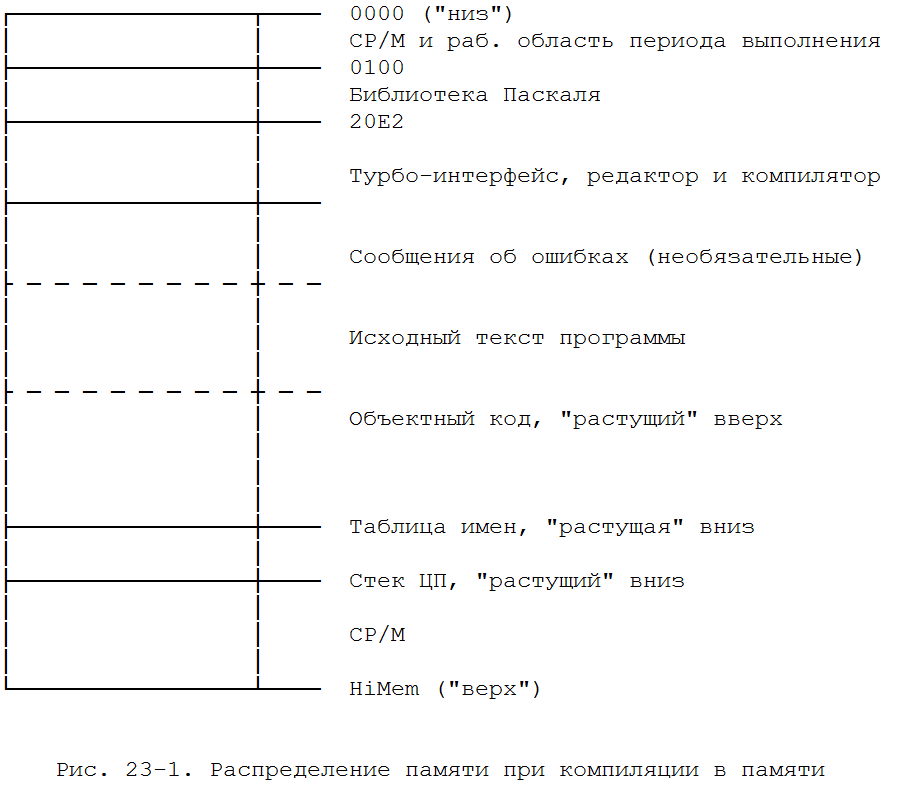
**ГЛАВА 23.**

**РАСПРЕДЕЛЕНИЕ ПАМЯТИ**

Эта глава описывает содержимое оперативной памяти на различных стадиях работы Турбо-системы. На следующих рисунках сплошными линиями обозначены фиксированные границы (т.е. границы, определяемые объемом оперативной памяти, размером операционной системы CP/M, версией Турбо-системы и т.д.), а штриховыми линиями обозначены те границы, которые определяются во время выполнения программы (например, границы, зависящие от размера исходного текста, или от возможной манипуляции пользователя различными указателями и т.д.). Размеры сегментов на рисунках необязательно отражают реально используемые объемы памяти.

**Компиляция в памяти**

Во время компиляции в память (по команде M из меню опций) распределение памяти соответствует рис. 23-1:

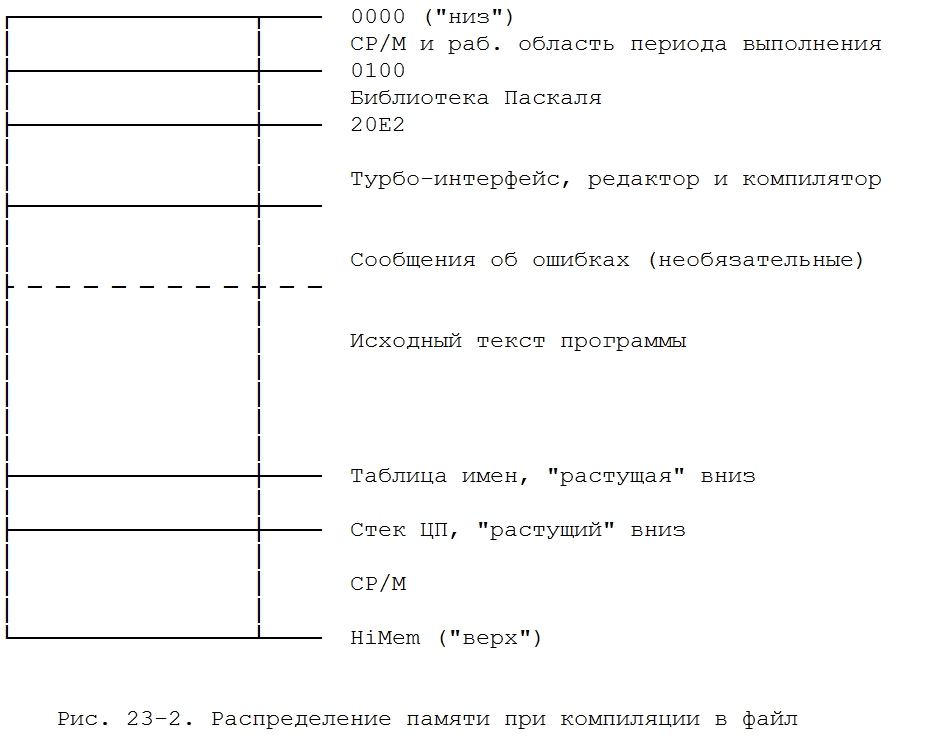


Если при запуске Турбо-системы не загружается файл сообщений об ошибках, то исходный текст располагается в памяти на соответствующее число байтов ниже (т.е. ближе к началу памяти).

После запуска компилятора последний начинает формировать объектный код, начиная с конца области, занимаемой исходным текстом. Стек ЦП заполняется "вниз", начиная с логической верхней границы памяти, а таблица имен компилятора заполняется также "вниз", но начиная с адреса, на 1 Кбайт ($400 байт) меньшего логической верхней границы памяти.

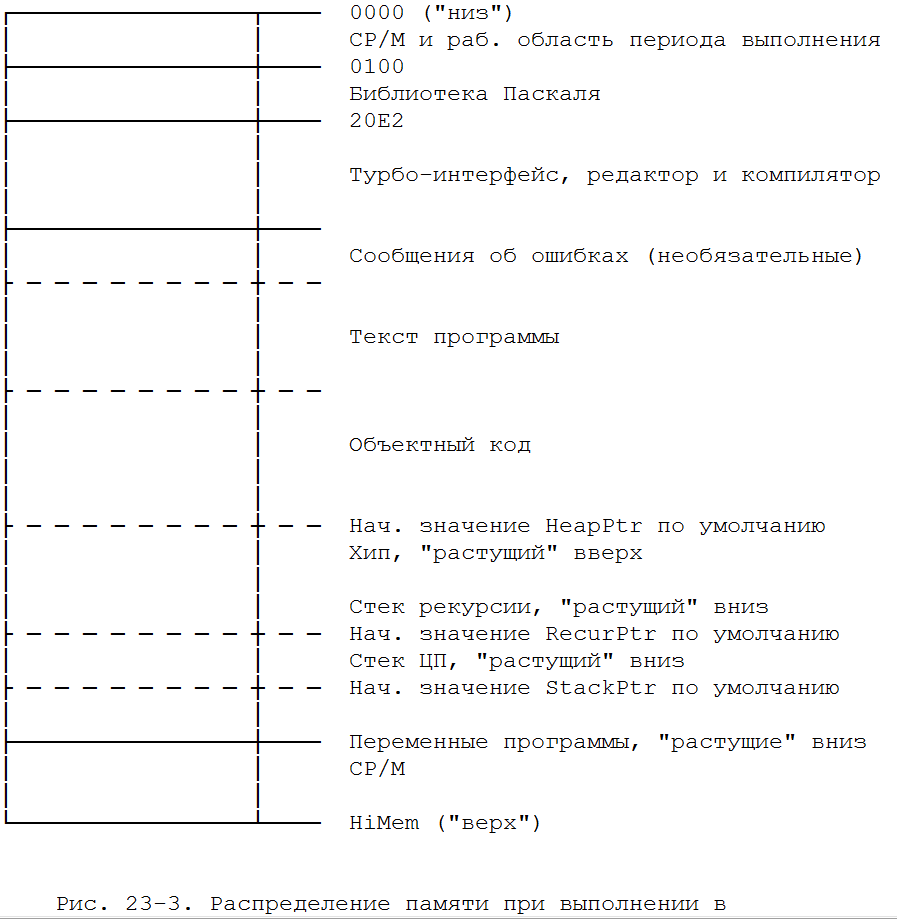
**Компиляция на диск**

Когда результатом компиляции является .COM- или .CHN-файл (по командам C или H из меню опций), карта распределения памяти в основном похожа на распределение при компиляции в памяти, за исключением того, что порождаемый объектный код пишется не в память, а на диск - в файл. Кроме того, код начинается с более высокого адреса (сразу после библиотеки Паскаля, а не после исходного текста). Следовательно, в этом режиме возможна компиляция значительно больших по размеру программ. Карта распределения памяти для этого случая показана на рис. 23-2.



**Выполнение программы, хранимой в памяти**

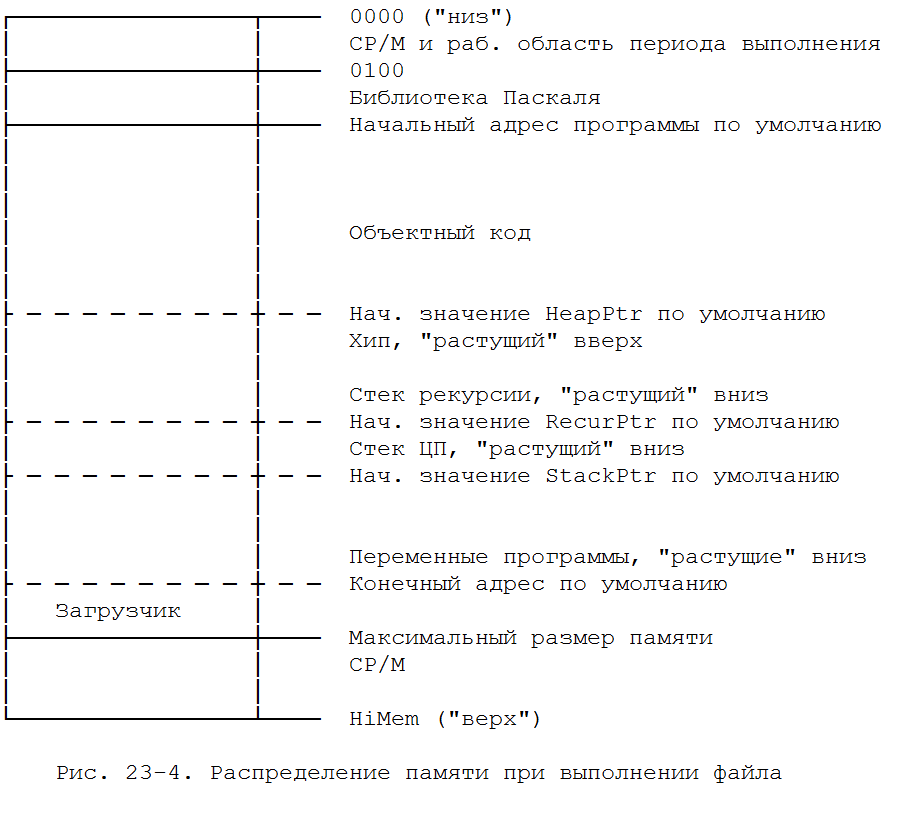
Когда программа выполняется в непосредственном режиме, т.е. в памяти (после компиляции с опцией M из меню опций), карта распределения памяти выглядит так, как показано на рис. 23-3.



Когда программа скомпилирована, становится известным конец объектного кода. По умолчанию указатель хипа HeapPtr устанавливается на это значение, и хип "растет" в памяти, начиная с этой точки по направлению к стеку рекурсии. максимальный размер памяти определяется как BDOS минус один (показано в меню опций компилятора). Программные переменные хранятся в памяти, начиная с этого адреса и "ниже". Конец области, отведенной переменным, совпадает с точкой "верхней границы свободной памяти", которая представляет собой начальное значение указателя стека ЦП StackPtr. Стек ЦП "растет" вниз от этой точки по направлению к позиции указателя стека рекурсии RecurPtr - на $400 байтов ниже, чем StackPtr. Стек рекурсии "растет" от этой точки вниз по направлению к хипу.

**Выполнение программы, хранимой на диске**

При выполнении программы, существующей в виде дискового файла (либо по команде Run в режиме Memory, выбранном из меню опций, либо по команде eXecute, либо непосредственно из CP/M) память распределяется так, как показано на рис. 23-4.



Эта карта распределения памяти напоминает предыдущую, за исключением того, что в ней отсутствуют интерфейс Турбо-системы, редактор и компилятор (и сообщения об ошибках), а также исходный текст программы. ПОДРАЗУМЕВАЕМЫЙ ПО УМОЛЧАНИЮ НАЧАЛЬНЫЙ АДРЕС ПРОГРАММЫ (показанный в меню опций компилятора) есть первый свободный байт после библиотеки периода выполнения Паскаля. Этим значением можно манипулировать посредством команды S из меню опций компилятора. Можно, например, зарезервировать место для абсолютных переменных (с атрибутом ABSOLUTE) и/или для внешних процедур между библиотекой и кодом. МАКСИМАЛЬНЫЙ РАЗМЕР ПАМЯТИ равен BDOS минус единица, и подразумеваемое по умолчанию значение определяется ячейкой BDOS для используемого компьютера.

В программах, предназначенных для перевода на другие системы, нужно проявлять особую осторожность, чтобы избежать коллизий с BDOS. Максимальным размером памяти можно манипулировать с помощью команды E (End address - конечный адрес) из меню опций компилятора. заметим, что по умолчанию подразумевается конечный адрес, на 700-1000 байтов меньший, чем максимально доступный объем памяти. Это связано с необходимостью оставить место для загрузчика, который располагается как раз под BDOS, когда .COM-файлы активизируются по командам R или X из Турбо-системы. По окончании работы программы этот загрузчик восстанавливает в памяти Турбо-редактор, компилятор и сообщения об ошибках (если они загружались) и тем самым возвращает управление Турбо-системе.

**ПРИЛОЖЕНИЕ A.**

**СТАНДАРТНЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ**

В этом приложении даны все стандартные процедуры и функции, которые имеются в Турбо-Паскале CP/M-80, с описанием их синтаксиса, параметров и типа. для обозначения элементов различных типов используются следующие символы:

TYPE произвольный тип

STRING произвольный строковый тип

FILE произвольный тип файла

SCALAR произвольный простой тип

POINTER произвольный тип указателя

Отсутствие спецификации типа параметра означает, что процедура или функция принимает параметры-переменные любого типа.

**Процедуры и функции ввода/вывода**

Следующие процедуры используют в своих списках параметров нестандартный синтаксис:

procedure

Read(var F: file of TYPE; var V: TYPE);

Read(var F: text; var I: Integer);

Read(var F: text; var R: Real);

Read(var F: text; var C: Char);

Read(var F: text; var S: STRING);

ReadLn(var F: text);

Write(var F: file of TYPE; var V: TYPE);

Write(var F: text; I: Integer);

Write(var F: text; R: Real);

Write(var F: text; B: Boolean);

Write(var F: text; C: Char);

Write(var F: text; S: STRING);

WriteLn(var F: text);

**Арифметические функции**

function

Abs(I: Integer): Integer;

Abs(R: Real): Real;

ArcTan(R: Real): Real;

Cos(R: Real): Real;

Exp(R: Real): Real;

Frac(R: Real): Real;

Int(R: Real): Real;

Ln(R: Real): Real;

Sin(R: Real): Real;

Sqr(I: Integer): Integer;

Sqr(R: Real): Real;

Sqrt(R: Real): Real;

**Функции для работы с простыми типами**

function

Odd(I: Integer): Boolean;

Pred(X: SCALAR): SCALAR;

Succ(X: SCALAR): SCALAR;

**Функции преобразования типов**

function

Chr(I: Integer): Char;

Ord(X: Scalar): Integer;

Round(R: Real): Integer;

Trunc(R: Real): Integer;

**Процедуры и функции для работы со строками**

При записи числового параметра процедуры STR используется нестандартный синтаксис.

procedure

Delete(var S: STRING; Pos,Len: Integer);

Insert(S: STRING; var D: STRING; Pos: Integer);

Str(I: Integer; var S: STRING);

Str(R: Real; var S: STRING);

Val(S: STRING; var R: Real; var P: Integer);

Val(S: STRING; var I,P: INTEGER);

function

Concat(S1,S2,..,Cn: STRING);

Copy(S: STRING; Pos,Len: Integer): STRING;

Length(S: STRING): Integer;

Pos(Pattern,Source: STRING): Integer;

**Подпрограммы для работы с файлами**

procedure

Append(var F: FILE; Name: STRING);

Assign(var F: FILE; Name: STRING);

BlockRead (var F: file; var Dest: TYPE; Num: Integer);

BlockWrite(var F: file; var Sour: TYPE; Num: Integer);

Chain(var F: file);

Close(var F: FILE);

Erase(var F: file);

Execute(var F: file);

Rename(var F: file; Name: STRING);

Reset(var F: FILE);

Rewrite(var F: FILE);

Seek(var F: file of TYPE; Pos: Integer);

function

Eof(var F: FILE): Boolean;

Eoln(var F: text): Boolean;

FilePos(var F: file of TYPE): Integer;

FilePos(var F: file): Integer;

FileSize(var F: file of TYPE): Integer;

FileSize(var F: file): Integer;

SeekEof(var F: text): Boolean;

SeekEoln(var F: text): Boolean;

**Процедуры и функции для управления хипом**

procedure

Dispose(var P: POINTER);

FreeMem(var P: POINTER; I: Integer);

GetMem(var P: POINTER; I: Integer);

Mark(var P: POINTER);

New(var P: POINTER);

Release(var P: POINTER);

function

MaxAvail: Integer;

MemAvail: Integer;

Ord(P: POINTER): Integer;

Ptr(I: Integer): POINTER;

**Процедуры и функции для работы с экраном**

procedure

CrtExit;

CrtInit;

ClrEol;

ClrScr;

DelLine;

GotoXY(X,Y: Integer);

InsLine;

LowVideo;

NormVideo;

**Прочие процедуры и функции**

procedure

Bdos(Func,Param: Integer);

Bios(Func,Param: Integer);

Delay(mS: Integer);

Exit;

FillChar(var Dest; Length: Integer; Data: Char);

FillChar(var Dest; Length: Integer; Data: Byte);

Halt;

Move(var Source,Dest: TYPE; Length: Integer);

Randomize;

function

Addr(var Variable): Integer;

Addr(<function identifier>): Integer;

Addr(<procedure identifier>): Integer;

Bdos(Func,Param: Integer): Byte;

BdosHL(Func,Param: Integer): Integer;

Bios(Func,Param: Integer): Byte;

BiosHL(Func,Param: Integer): Integer;

Hi(I: Integer): Integer;

IOresult: Integer;

KeyPressed: Boolean;

Lo(I: Integer): Integer;

ParamCount: Integer;

ParamStr(N: Integer): STRING;

Random(Range: Integer): Integer;

Random: Real;

SizeOf(var Variable): Integer;

SizeOf(<type identifier>): Integer;

Swap(I: Integer): Integer;

UpCase(Ch: Char): Char;

**ПРИЛОЖЕНИЕ B.**

**ОПЕРАЦИИ**

В настоящем приложении сведены все операции, которые имеются в Турбо-Паскале. Операции располагаются здесь в порядке убывания приоритета. В тех случаях, где тип операндов указан как INTEGER, REAL, тип результата определяется следующим образом:

операнд результат

Integer,Integer Integer

Real,Real Real

Real,Integer Real

Таблица B-1

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Действие | Типы операндов | Тип результата |
| Унарный + | Сохранение знака | Integer, Real | Совпадает с типом операнда |
| Унарный - | Инверсия знака | Integer,Real | Совпадает с типом операнда |
| NOT | Отрицание | Integer, Boolean | Совпадает с типом операнда |
| \* | Произведение Пересечение множеств | Integer,Real Произвольный тип множества | Integer,Real  Совпадает с типом операнда |
| / | Деление | Integer,Real | Real |
| DIV | Деление нацело | Integer | Integer |
| MOD | Остаток от деления | Integer | Integer |
| AND | Арифметическое "И"  Логическое "И" | Integer  Boolean | Integer  Boolean |
| SHL | Сдвиг влево | Integer | Integer |
| SHR | Сдвиг вправо | Integer | Integer |
| + | Сложение  Конкатенация Объединение множеств | Integer, Real,  String Произвольный тип множества | Integer, Real, String  Совпадает с типом операнда |
| - | Вычитание  Разность множеств | Integer,Real Произвольный тип множества | Integer,Real  Совпадает с типом операнда |
| OR | Арифметическое "ИЛИ"  Логическое "ИЛИ" | Integer  Boolean | Integer  Boolean |
| XOR | Арифметическое "Исключающее ИЛИ"  Логическое "Исключающее ИЛИ" | Integer  Boolean | Integer  Boolean |
| = | Равенство  Равенство  Равенство  Равенство | Произвольный  простой тип  String  Произвольный тип множества  Произвольный тип указателя | Boolean  Boolean  Boolean  Boolean |
| <> | Неравенство  Неравенство  Неравенство  Неравенство | Произвольный  простой тип  String  Произвольный тип множества  Произвольный тип указателя | Boolean |
| >= | Больше или равно  Больше или равно  Включение множеств | Произвольный  простой тип  String  Произвольный тип множества | Boolean |
| <= | Меньше или равно  Меньше или равно  Включение множеств | Произвольный  простой тип  String  Произвольный тип множества | Boolean  Boolean  Boolean |
| > | Больше  Больше | Произвольный  простой тип  String | Boolean  Boolean |
| < | Меньше  Меньше | Произвольный  простой тип  String | Boolean  Boolean |
| IN | Принадлежность к  множеству | Смотри ниже | Boolean |

Примечание: Первый операнд в операции IN может принадлежать к любому простому типу, в то время как второй операнд этой операции должен быть множеством элементов этого простого типа.

**ПРИЛОЖЕНИЕ C.**

**ДИРЕКТИВЫ КОМПИЛЯТОРА**

Ряд функций компилятора системы Турбо-Паскаль управляется посредством директив компилятора. Директива компилятора записывается в виде комментария с особым синтаксисом, что в частности означает, что директива компилятора допустима везде, где допустим комментарий.

Директива компилятора начинается открывающей фигурной скобкой, за которой непосредственно следует знак доллара и одна директива или несколько директив, разделенных запятыми, и завершается закрывающей фигурной скобкой. Каждая директива задается одной буквой.

Примеры:

{$I-}

{$I INCLUDE.FIL}

{$B-,R+,V-}

(\*$U+\*)

Отметим, что перед знаком доллара и после него пробелы не допускаются. Знак + после директивы означает, что соответствующая директива устанавливается в активное состояние, т.е. "включается", а знак - означает пассивное состояние директивы, т.е. выключенное.

ВАЖНОЕ ЗАМЕЧАНИЕ. Все директивы компилятора имеют значения, подразумеваемые по умолчанию. Эти значения подобраны таким образом, чтобы оптимизировать скорость исполнения и минимизировать размер кода программы. Так, например, по умолчанию "выключены" такие возможности, как генерация кода для рекурсивных процедур и проверка индексов массивов. Поэтому программист должен удостовериться в том, что в его программу включены необходимые установки директив. При этом материал настоящего приложения может служить в качестве справочника.

**A - абсолютный код**

Значение, подразумеваемое по умолчанию: A+

Директива A управляет порождением абсолютного, т.е. нерекурсивного кода. При активном ее состоянии: {$A+} порождается абсолютный код, а при пассивном: {$A-} компилятор порождает код, который допускает рекурсивные вызовы. Такой код требует больше памяти и медленнее выполняется.

**B - выбор режима ввода/вывода**

Значение, подразумеваемое по умолчанию: B+

Директива B управляет выбором режима ввода/вывода. При активном состоянии этой директивы: {$B+} устройство CON: ассоциируется со стандартными файлами INPUT и OUTPUT, т.е. с подразумеваемыми по умолчанию каналами ввода/вывода. При пассивном состоянии этой директивы: {$B-} используется устройство TRM: . ЭТА ДИРЕКТИВА ЯВЛЯЕТСЯ ГЛОБАЛЬНОЙ ПО ОТНОШЕНИЮ КО ВСЕМУ БЛОКУ ПРОГРАММЫ, и не может быть переопределена в каком либо другом месте программы. Подробнее этот вопрос рассматривается в подразделе "Стандартные файлы" главы 14.

**C - интерпретация CTRL-C и CTRL-S**

Значение, подразумеваемое по умолчанию: C+

Эта директива управляет интерпретацией управляющих символов при вводе/выводе с консоли. При активном состоянии этой директивы: {$C+} комбинация CTRL-C, введенная в ответ на оператор READ или READLN, прерывает выполнение программы, а попеременное нажатие комбинации CTRL-S осуществляет запрет/разрешение вывода на экран. При пассивном состоянии C-директивы: {$C-} управляющие комбинации никак не интерпретируются. активное состояние директивы несколько замедляет вывод на экран, так что в тех случаях, когда скорость вывода на экран является решающей, нужно устанавливать эту директиву в пассивное состояние. ЭТА ДИРЕКТИВА ЯВЛЯЕТСЯ ГЛОБАЛЬНОЙ ПО ОТНОШЕНИЮ КО ВСЕМУ БЛОКУ ПРОГРАММЫ и не может быть переопределена в каком-либо другом месте программы.

**I - обработка ошибок ввода/вывода**

Значение, подразумеваемое по умолчанию: I+

Директива I управляет обработкой ошибок ввода/вывода. При активном состоянии этой директивы: {$I+} все операции ввода/вывода контролируются на наличие ошибок. Если же I-директива установлена в пассивное состояние, то ответственность за обнаружение ошибок ввода/вывода возлагается на программиста, который должен использовать для этого стандартную функцию IORESULT. Подробнее этот вопрос рассматривается в разделе "Контроль вво да/вывода" главы 14.

**I - включение файлов**

Директива I, которая сопровождается именем файла, предписывает компилятору включить файл с указанным именем в данную компиляцию. Включение файлов в компиляцию подробно рассматривается в главе 17.

**R - контроль границ индексов**

Значение, подразумеваемое по умолчанию: R-

Директива R управляет проверкой границ индексов во время выполнения программы. При активном состоянии этой директивы {$R+} все операции индексирования массивов контролируются на принадлежность заданным границам значений, а все присваивания переменным простых типов и отрезков типов контролируются на принадлежность соответствующим диапазонам значений. При пассивном состоянии директивы {$R-} никаких проверок не выполняется, и ошибки индексирования вполне могут стать причиной того, что программа "пойдет вразнос". Установку этой директивы в активное состояние можно считать хорошей практикой, пока программа еще не до конца отлажена. После того, как отладка программы закончена, перевод R-директивы в пассивное состояние увеличит скорость выполнения программы (пассивное состояние подразумевается по умолчанию).

**V - контроль типа для VAR-параметров**

Значение, подразумеваемое по умолчанию: V+

Директива V управляет проверкой типа строк, передаваемых в качестве VAR-параметров. При активном состоянии этой директивы:

{$V+} выполняется строгая проверка типов, т.е. длины фактического и формального параметров должны совпадать.

При пассивном же состоянии директивы:

{$V-} компилятор разрешает передавать такие фактические параметры, длина которых не совпадает с длиной формального параметра. Подробнее этот вопрос рассматривается в разделе "Параметры" главы 16.

**U - прерывание программы пользователем**

Значение, подразумеваемое по умолчанию: U-

Директива U управляет возможностью прерывания выполнения программы пользователем. При активном состоянии этой директивы:

{$U+} пользователь может в любой момент прервать выполнение программы посредством ввода комбинации CTRL-C.

Пассивное состояние:

{$U-} не приводит к такому эффекту. Активизация этой директивы значительно замедляет скорость выполнения программы.

**W - вложенность операторов WITH**

Значение, подразумеваемое по умолчанию: W2

Директива W управляет уровнем вложенности операторов WITH, т.е. определяет максимальное количество записей, которые могут быть "открыты" внутри одного блока. Сразу после буквы W должна следовать цифра от 1 до 9. Подробнее этот вопрос рассматривается в разделе "Оператор WITH" главы 11.

**X - оптимизация массивов**

Значение, подразумеваемое по умолчанию: X+

Директива X управляет оптимизацией кода, реализующего работу с массивами. При активном состоянии этой директивы: {$X+} код, реализующий работу с массивами, оптимизируется в смысле скорости работы, а при пассивном состоянии: {$X-} - в смысле объема занимаемой памяти. Подробнее этот вопрос рассматривается в разделе "Оптимизация индексов массивов" главы 10.

**ПРИЛОЖЕНИЕ D.**

**СРАВНЕНИЕ ТУРБО-ПАСКАЛЯ СО СТАНДАРТНЫМ ПАСКАЛЕМ**

Язык Турбо-Паскаль следует определению Стандартного Паскаля, изложенному в книге Иенсена и Вирта "Паскаль. Сообщение и руководство пользователя", и отличается лишь некоторыми второстепенными деталями, введенными исключительно в целях эффективности. Настоящее приложение посвящено описанию этих отличий. Отметим, что РАСШИРЕНИЯ, предлагаемые Турбо-Паскалем, здесь не обсуждаются.

**Динамические переменные**

Процедура NEW не принимает спецификации записей с вариантами. Однако это ограничение можно легко обойти, используя стандартную процедуру GETMEM.

**Рекурсия**

Из-за характера работы с локальными переменными при рекурсии переменная, локальная по отношению к подпрограмме, не может передаваться в качестве VAR-параметра в рекурсивном вызове.

**Процедуры GET и PUT**

В Турбо-Паскале не реализованы стандартные процедуры GET и PUT. Вместо этого процедуры READ и WRITE расширены таким образом, что они обслуживают все необходимые потребности ввода/вывода. Это сделано по следующим причинам. Во-первых, READ и WRITE обеспечивают гораздо более быстрый ввод/вывод; во-вторых, уменьшаются накладные расходы на память для переменных, поскольку не требуются буферные переменные файлов, и, наконец, в-третьих, процедуры READ и WRITE являются гораздо более гибкими и доступными для понимания, чем GET и PUT.

**Операторы GOTO**

Операторы GOTO не должны передавать управление за пределы текущего блока.

**Процедура PAGE**

Стандартная процедура PAGE в Турбо-Паскале не реализована, поскольку в операционной системе CP/M не определен символ перевода формата.

**Упакованные переменные**

Зарезервированное слово PACKED не производит в Турбо-Паскале никакого действия, но использовать его допустимо. Это связано с тем, что упаковка выполняется автоматически везде, где это возможно. По этой же причине не реализованы стандартные процедуры PACK и UNPACK.

**Параметры-процедуры**

В Турбо-Паскале процедуры и функции не могут передаваться в качестве параметров в подпрограммы.

**ПРИЛОЖЕНИЕ E.**

**СООБЩЕНИЯ КОМПИЛЯТОРА ОБ ОШИБКАХ**

Ниже приводится перечень сообщений об ошибках, которые может выдавать компилятор. Обнаружив ошибку, компилятор всегда выводит на экране ее номер. Пояснительный текст выводится лишь в том случае, если вы включили в систему сообщения об ошибках (т.е. ответили "Y" на первый вопрос при запуске Турбо-системы).

Смысл многих сообщений об ошибках совершенно ясен из текста этих сообщений, и лишь немногие требуют дополнительных пояснений. Эти пояснения Вы также найдете ниже.

|  |  |
| --- | --- |
| Код и текст сообщения | Перевод текста и комментарии |
| 01 ';' expected | Должна стоять точка с запятой. |
| 02 ':' expected | Должно стоять двоеточие. |
| 03 ',' expected | Должна стоять запятая. |
| 04 '(' expected | Должна стоять открывающая |
|  | круглая скобка. |
| 05 ')' expected | Должна стоять закрывающая |
|  | круглая скобка. |
| 06 '=' expected | Должен стоять знак равенства |
| 07 ':=' expected | Должен стоять символ |
|  | присваивания. |
| 08 '[' expected | Должна стоять открывающая |
|  | квадратная скобка. |
| 09 ']' expected | Должна стоять закрывающая |
|  | квадартная скобка. |
| 10 '.' expected | Должна стоять точка. |
| 11 '..' expected | Должен стоять символ отрезка типа. |
| 12 BEGIN expected | Должно стоять слово BEGIN. |
| 13 DO expected | Должно стоять слово DO. |
| 14 END expected | Должно стоять слово END. |
| 15 OF expected | Должно стоять слово OF. |
| 16 PROCEDURE or FUNCTION | Должно стоять слово |
| expected | PROCEDURE или FUNCTION. |
| 17 THEN expected | Должно стоять слово THEN. |
| 18 TO or DOWNTO expected | Должно стоять слово TO или DOWNTO. |
| 20 Boolean expression expected | Должно стоять выражение типа BOOLEAN. |
| 21 File variable expected | Здесь должна быть переменная-файл. |
| 22 Integer constsnt expected | Здесь должна быть константа INTEGER. |
| 23 Integer expression expected | Здесь должно быть выражение типа INTEGER. |
| 24 Integer variable expected | Здесь должна быть переменная типа INTEGER. |
| 25 Integer or real constant expected | Здесь должна быть константа INTEGER или REAL. |
| 26 Integer or real expression expected | Здесь должно быть выражение INTEGER или REAL. |
| 27 Integer or real variable expected | Здесь должна быть переменная типа INTEGER или REAL. |
| 28 Pointer variable expected | Здесь должна быть переменная-указатель. |
| 29 Record variable expected | Здесь должна быть переменная-запись. |
| 30 Simple type expected | Здесь должен быть любой простой тип, кроме REAL. |
| 31 Simple expression expected | Здесь должно быть выражение простого типа, кроме REAL. |
| 32 String constant expected | Здесь должна быть строковая константа. |
| 33 String expression expected | Здесь должно быть строковое выражение. |
| 34 String variable expected | Здесь должна быть строковая переменная. |
| 35 Textfile expected | Здесь должен быть текстовый файл. |
| 36 Type identifier expected | Здесь должен быть идентификатор типа. |
| 37 Untyped file expected | Здесь должен быть файл без типа. |
| 40 Undefined label | Неопределенная метка. Указанный оператор ссылается на неопределенную метку. |
| 41 Unknown identifier or syntax error | Неопределенный идентификатор или синтаксическая ошибка. Компилятор встретил неопределенный идентификатор метки, константы, типа, переменной или поля, либо в операторе имеется синтаксическая ошибка. |
| 42 Undefined pointer type in preceding type definitions | Неизвестный тип указателя в предшествующих определениях типа. Предшествующее определение типа указателя содержит ссылку на неизвестный тип. |
| 43 Duplicate identifier or label | Этот идентификатор или метка уже использовался внутри текущего блока. |
| 44 Type mismatch | Несоответствие типов. 1) Несовместимость типов переменной и выражения в операторе присваивания; 2) Несовместимость типов фактического и формального параметров в вызове подпрограммы; 3) Несовместимость типов выражения и индекса при обращении к массиву; 4) Несовместимость типов операндов в выражении. |
| 45 Constant out of range | Константа выходит за допустимые границы значений. |
| 46 Constant and CASE selector type does not match | Константа не соответствует типу селектора в операторе CASE. |
| 47 Operand type(s) does not match operator | Типы операндов не соответствуют операции, например, 'A' DIV '2'. |
| 48 Invalid result type | Недопустимый тип результата. Допустимыми являются все простые типы, включая REAL, строковые типы и типы указателей. |
| 49 Invalid string length | Недопустимая длина строки. Длина строки должна укладываться в диапазон 0..255. |
| 50 String constant length does not match type | Длина строковой константы не соответствует типу. |
| 51 Invalid subrange base type | Недопустимый базовый тип для отрезка типа. |
| 52 Lower bound > upper bound | Нижняя граница больше верхней. Порядковое значение верхней границы должно быть больше или равно порядковому значению нижней границы. |
| 53 Reserved word | Зарезервированное слово. |
| 54 Illegal assignment | Недопустимое присваивание. |
| 55 String constant exceeds line | Строковая константа выходит за пределы строки программы. Строковые константы не должны переходить с одной строки текста на другую. |
| 56 Error in integer constant | Ошибка в целой константе. Константа типа INTEGER не соответствует синтаксису, определенному в главе 4, или выходит за пределы диапазона -32768..32767. Числа типа REAL должны оканчиваться десятичной точкой и нулем, например 123456789.0 |
| 57 Error in real constant | Ошибка в вещественной константе. Синтаксис констант типа REAL определен в главе 4. |
| 58 Illegal character in identifier | Недопустимый символ в идентификаторе. |
| 60 Constants are not allowed here | Константы здесь недопустимы. |
| 61 Files and pointers are not allowed here | В этом месте недопустимы файлы и указатели. |
| 62 Structured variables | В этом месте недопустимы |
| are not allowed here | структурные переменные. |
| 63 Textfiles are not allowed here | В этом месте недопустимы текстовые файлы. |
| 64 Textfiles and untyped files are not allowed here | В этом месте недопустимы текстовые файлы и файлы без типа |
| 65 Untyped files are not allowed here | В этом месте недопустимы файлы без типа. |
| 66 I/O not allowed here | В этом месте недопустим ввод/вывод. |
| 67 Files must be VAR parameters | Файлы должны быть VAR-параметрами. |
| 68 File components may not | Компоненты файлов не могут |
| be files | быть файлами. |
| 69 Invalid ordering of fields | Недопустимый порядок полей. |
| 70 Set base type out of range | Базовый тип множества выходит за допустимые границы. Базовым типом множества может быть простой тип с числом возможных значений не более 256 или отрезок типа, обе границы которого не вы ходят за пределы диапазона 0..255. |
| 71 Invalid GOTO | Недопустимый переход по GOTO. Этот оператор не должен ссылаться на метку внутри цикла FOR извне этого цикла. |
| 72 Label not within current block | Метка вне пределов текущего блока. Оператор GOTO не должен ссылаться на метку вне текущего блока. |
| 73 Undefined FORWARD procedure(s) | Неопределенная FORWARD-процедура. |
| 74 INLINE error | Ошибка оператора INLINE. |
| 75 Illegal use of absolute | Недопустимое употребление ABSOLUTE. 1) В объявлении абсолютной переменной перед двоеточием может стоять только один идентификатор; 2) Атрибут ABSOLUTE нельзя использовать в записи. |
| 76 Overlays can not be forwarded | Запрещается предварительно объявлять оверлеи. Для оверлейных процедур недопустимо указывать спецификацию FORWARD. |
| 77 Overlays not allowed in | Оверлеи недопустимы в режиме |
| direct mode | непосредственного исполнения (Memory). Оверлеи можно использовать только в программах, скомпилированных в файл. |
| 90 File not found | Файл не найден. Указанный INCLUDE-файл не существует на диске. |
| 91 Unexpected end of source | Неожиданный конец исходного текста. Программа оканчивается недопустимым способом. Возможно, в программе слов BEGIN больше, чем слов END. |
| 92 Unable to create overlay file | Невозможно создать оверлейный файл. |
| 93 Invalid compiler directive | Недопустимая директива компилятора. |
| 96 Nested INCLUDE are not allowed | Вложенные INCLUDE-директивы запрещены. |
| 97 Too many nested WITHs | Слишком много вложенных операторов WITH. Воспользуйтесь W-директивой компилятора для увеличения максимального числа вложенных операторов WITH. По умолчанию подразумевается глубина 2. |
| 98 Memory overflow | Переполнение памяти. В Вашей программе делается попытка выделить для переменных больше памяти, чем доступно системе. |
| 99 Compiler overflow | Переполнение компилятора. Не хватает памяти для компиляции программы. Такая ошибка может возникнуть даже в том случае, если Вам кажется, что свободная память есть. Однако, эта свободная память используется для стека и таблиц имен во время компиляции. Вам следует разбить исходный текст на фрагменты меньшего размера и воспользоваться INCLUDE-файлами. |

**ПРИЛОЖЕНИЕ F.**

**СООБЩЕНИЯ ОБ ОШИБКАХ ПЕРИОДА ВЫПОЛНЕНИЯ**

Фатальные ошибки, возникающие во время выполнения программы, приводят к останову программы и выводу следующего сообщения:

┌───────────────────────────────────────────┐

│ Run-time error NN, PC=addr │

│ Program aborted │

└───────────────────────────────────────────┘

В этом сообщении NN - это номер ошибки периода выполнения, а addr - адрес того места в коде программы, где произошла ошиб ка. Ниже приводится перечень ошибок времени выполнения, в котором каждый номер сопровождается пояснительным текстом. Обратите внимание на то, что номера являются 16-ричными.

|  |  |
| --- | --- |
| Код и текст сообщения | Перевод текста и комментарии |
| 01 Floating point overflow | Переполнение с плавающей запятой. |
| 02 Division by zero attempted | Попытка деления на ноль. |
| 03 Sqrt argument error | Ошибочный аргумент SQRT. Аргумент, переданный функции SQRT, оказался отрицательным. |
| 04 Ln argument error | Ошибочный аргумент для Ln. Аргумент, переданный функции Ln, оказался <=0. |
| 10 String length error | Неверная длина строки. 1) Конкатенация строк дает строку, длина которой больше 255; 2) Только строки длиной 1 допускают преобразование в символ. |
| 11 Invalid string index | Недопустимый индекс строки. При вызове процедур COPY, DELETE или INSERT индексное выражение выходит за пределы диапазона 1..255. |
| 90 Index out of range | Индекс вне границ значений. Выход за допустимые границы индексного выражения при вычислении индекса массива. |
| 91 Scalar or subrange | Простой тип или отрезок типа |
| out of range | вне границ значений. Значение, присваиваемое переменной простого типа или отрезка типа, выходит за допустимые границы значений. |
| 92 Out of integer range | Выход за границы целых значений. Вещественное значение, переданное в качестве аргумента для TRUNC или ROUND, выходит за рамки диапазона INTEGER -32768..32767. |
| F0 Overlay file not found | Оверлейный файл не найден. |
| FF Heap/stack collision | Коллизия стека и хипа. При вызове стандартной процедуры NEW или рекурсивной подпрограммы оказалось, что между указателем хипа (HeapPtr) и указателем стека рекурсии (RecurPtr) недостаточно места. |

Примечание: В процессе эксплуатации компилятора TURBO Pascal V3.02A было замечено, что ошибка 92 возникает и в том случае, если вещественный аргумент TRUNC или ROUND имеет значение -32768. Указанную некорректность в работе программы можно устранить, например, следующим образом (R имеет тип REAL, I - тип INTEGER):

if R< -Maxint then I:=Round(R+1)-1

else I:=Round(R);

**ПРИЛОЖЕНИЕ G.**

**СООБЩЕНИЯ ОБ ОШИБКАХ ВВОДА/ВЫВОДА**

Ошибки ввода/вывода порождаются особыми ситуациями, возникающими в операциях ввода/вывода во время выполнения программы. В условиях действия контроля ввода/вывода (при активном состоянии I-директивы компилятора) ошибка ввода/вывода приводит к останову программы и выдаче следующего сообщения об ошибке:

┌───────────────────────────────────────────┐

│ I/O error NN, PC=addr │

│ Program aborted │

└───────────────────────────────────────────┘

Здесь NN - это номер ошибки ввода/вывода, а addr есть адрес того места в коде программы, где возникла ошибка.

При пассивном состоянии контроля ошибок ввода/вывода: {$I-} ошибка ввода/вывода не приводит к останову программы. Вместо этого все дальнейшие операции ввода/вывода должны быть приостановлены до тех пор, пока результат ошибочной операции ввода/вывода не будет проанализирован с помощью стандартной функции IORESULT. Если после возникновения ошибки делается попытка выполнить операцию ввода/вывода до вызова функции IORESULT, то возникнет новая ошибка, которая может "подвесить" программу.

Ниже приводится перечень сообщений об ошибках с кодовыми номерами и пояснительным текстом. Обратите внимание на то, что номера являются 16-ричными.

|  |  |
| --- | --- |
| Код и текст сообщения | Перевод текста и комментарии |
| 01 File does not exist | Файл не существует. Имя файла в RESET, ERASE, RENAME, EXECUTE или CHAIN не соответствует ни одному из существующих файлов. |
| 02 File not open for input | Файл не открыт для ввода. 1) Попытка читать (посредством READ или READLN) файл без предварительного вызова RESET или REWRITE; 2) Попытка читать текстовый файл, подготовленный с помощью REWRITE, и следовательно пустой; 3) Попытка читать с логического устройства LST: , которое предназначено только для вывода. |
| 03 File not open for output | Файл не открыт для вывода. 1) Попытка писать (посредством WRITE или WRITELN) в файл без предварительного вызова RESET или REWRITE; 2) Попытка писать в текстовый файл, подготовленный к работе с помощью RESET; 3) По пытка писать на логическое устройство KBD:, которое допускает только чтение. |
| 04 File not open | Файл не открыт. Сделана по пытка доступа (с помощью BLOCKREAD или BLOCKWRITE) к файлу без предварительного вызова RESET или REWRITE. |
| 10 Error in numeric format | Ошибка в формате числа. Строка, прочитанная из текстового файла в числовую переменную, не удовлетворяет соответствующему числовому формату (см. раздел "Числа" главы 4). |
| 20 Operation not allowed | Эта операция недопустима для |
| on a logical device | логического устройства. Делается попытка вызова процедур ERASE, RENAME, CHAIN или EXECUTE для файла, назначенного на логическое устройство. |
| 21 Not allowed in direct mode | Недопустимо в режиме непосредственного исполнения. Программа, исполняющаяся в непосредственном режиме (т.е. программа, активируемая командой Run при выбранной M-опции компилятора), не может запускать другие программы с помощью процедур CHAIN и EXECUTE. |
| 22 Assign to std files not allowed | Выполнять ASSIGN для стандартных файлов запрещено. |
| 90 Record length mismatch | Нестыковка длины записи. Длина записи файловой переменной не стыкуется с файлом, который Вы пытаетесь с этой переменной связать. |
| 91 Seek beyond end of file | Поиск за пределами файла. |
| 99 Unexpected end of file | Неожиданный конец файла. 1) При чтении текстового файла |
|  | физический конец файла обнаружен до того, как встретился знак EOF (CTRL-Z); 2) Для определенного файла была сделана попытка чтения после конца файла; 3) Процедура READ или BLOCKREAD не смогла прочитать очередной сектор в определенном файле. Возможно, что-то случилось с файлом, а может быть (в случае BLOCKREAD) Вы пытаетесь читать после физического EOF. |
| F0 Disk write error | Ошибка записи на диск. При попытке расширения файла диск оказался полон. Это может произойти при операциях вывода WRITE, WRITELN, BLOCKWRITE и FLUSH, однако такую ошибку могут вызвать и операции READ, READLN и CLOSE, поскольку они вызывают опорожнение буфера записи. |
| F1 Directory is full | Директорий заполнен. Вы пытаетесь выполнить REWRITE для некоторого файла, однако в дисковом директории нет больше места. |
| F2 File size overflow | Переполнение размера файла. Делается попытка записать (с Помощью WRITE) запись с номером, превышающим 65535, в определенный файл. |
| F3 Too many open files | Слишком много открытых файлов. |
| FF File disappeared | Файл исчез. Была сделана попытка закрыть (с помощью CLOSE) файл, которого уже нет в директории диска. Это может произойти из-за неожиданной смены диска. |

**ПРИЛОЖЕНИЕ H.**

**ПЕРЕВОД СООБЩЕНИЙ ОБ ОШИБКАХ**

Сообщения компилятора об ошибках содержатся в файле TURBO.MSG. Эти сообщения представлены на английском языке, однако их можно легко перевести на любой другой язык, пользуясь приведенными ниже инструкциями.

Первые 24 строки этого файла определяют несколько текстовых констант для последующего включения в строки сообщений об ошибках; этот метод позволяет в огромной степени уменьшить потребности сообщений об ошибках в дисковой и оперативной памяти. Каждая константа идентифицируется управляющим символом, который обозначается в последующем листинге как ^ . Значением константы является все, что располагается следом на той же строке. Все знаки являются значащими, в том числе ведущие и хвостовые пробелы.

Каждая из последующих строк файла содержит по одному сообщению и начинается с кодового номера ошибки, за которым непосредственно следует текст сообщения. Текст сообщения может состоять из любых знаков и может включать в себя ранее определенные идентификаторы констант, т.е. управляющие символы. Полностью полученные в результате сообщения перечислены в приложении E.

При переводе сообщений на другой язык соотношение между константами и сообщениями может оказаться совершенно не таким, каким оно является для показанной здесь английской версии. Поэтому Вам следует начать с того, чтобы записать все сообщения полностью, без использования констант. После того, как все сообщения переведены, следует найти возможно большее число общих участков текста. Теперь нужно определить эти общие участки в качестве констант в начале файла, а в последующие тексты сообщений включить только их идентификаторы. Число констант, которые Вы определите, зависит только от Вас, и ограничением здесь является только число управляющих символов.

В качестве хорошего примера использования констант рассмотрите сообщения 25, 26 и 27. Они определены исключительно с помощью идентификаторов констант (всего 15) и потребовали бы 101 символ текста, будучи записанными явным образом.

Для редактирования файла TURBO.MSG можно воспользоваться редактором системы Турбо. Управляющие символы вводятся с префиксом CTRL-P, т.е. для ввода в файл CTRL-A (^A) нужно нажать клавишу CONTROL (УПР) и, удерживая ее, нажать сначала P, а затем A. Управляющие символы появляются на экране в бледном свечении (если дисплей обладает соответствующими возможностями).

Отметим, что редактор системы Турбо удаляет все хвостовые пробелы. Поэтому в исходных сообщениях хвостовые пробелы не используются.

**Листинг файла сообщений об ошибках**

^A are not allowed

^B can not be

^C constant

^D does not

^E expression

^F identifier

^G file

^H here

^K Integer

^L File

^N Illegal

^O or

^P Undefined

^Q match

^R real

^S String

^T Textfile

^U out of range

^V variable

^W overflow

^X expected

^Y type

^[ Invalid

^] pointer

01';'^X

02':'^X

03','^X

04'('^X

05')'^X

06'='^X

07':='^X

08'['^X

09']'^X

10'.'^X

11'..'^X

12 BEGIN^X

13 DO^X

14 END^X

15 OF^X

17 THEN^X

18 TO^O DOWNTO^X

20 Boolean^E^X

21 ^L^V^X

22 ^K^C^X

23 ^K^E^X

24 ^K^V^X

25 ^K^O^R^C^X

26 ^K^O^R^E^X

27 ^K^O^R^V^X

28 Pointer^V^X

29 Record^V^X

30 Simple^Y^X

31 Simple^E^X

32 ^S^C^X

33 ^S^E^X

34 ^S^V^X

35 ^T^X

36 Type^F^X

37 Untyped^G^X

40 ^P label

41 Unknown^F^O syntax error

42 ^P^]^Y in preceding^Y definitions

43 Duplicate^F^O label

44 Type mismatch

45 ^C^U

46 ^C and CASE selector^Y^D^Q

47 Operand^Y(s)^D^Q operator

48 ^[ result^Y

49 ^[ ^S length

50 ^S^C length^D^Q^Y

51 ^[ subrange base^Y

52 Lower bound > upper bound

53 Reserved word

54 ^N assignment

55 ^S^C exceeds line

56 Error in integer^C

57 Error in^R^C

58 ^N character in^F

60 ^Cs^A^H

61 ^Ls and^]s^A^H

62 Structured^Vs^A^H

63 ^Ts^A^H

64 ^Ts and untyped^Gs^A^H

65 Untyped^Gs^A^H

66 I/O^A

67 ^Ls must be^V parameters

68 ^L components^B^Gs

69 ^[^Odering of fields

70 Set base^Y^U

71 ^[ GOTO

72 Label not within current block

73 ^P FORWARD procedure(s)

74 INLINE error

75 ^N use of ABSOLUTE

90 ^L not found

91 Unexpected end of source

97 Too many nested WITH's

98 Memory^W

99 Compiler^W

**ПРИЛОЖЕНИЕ I.**

**ТАБЛИЦА КОДА ASCII**

В данном приложении приведены две половины кодовой таблицы ASCII для альтернативной кодировки символов. Рядом с изображением каждого символа указан его шестнадцатеричный код.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ^@ 00 | ^P 10 | 20 | 0 30 | @ 40 | P 50 | ` 60 | p 70 |
| ^A 01 | ^Q 11 | ! 21 | 1 31 | A 41 | Q 51 | a 61 | q 71 |
| ^B 02 | ^R 12 | " 22 | 2 32 | B 42 | R 52 | b 62 | r 72 |
| ^C 03 | ^S 13 | # 23 | 3 33 | C 43 | S 53 | c 63 | s 73 |
| ^D 04 | ^T 14 | $ 24 | 4 34 | D 44 | T 54 | d 64 | t 74 |
| ^E 05 | ^U 15 | % 25 | 5 35 | E 45 | U 55 | e 65 | u 75 |
| ^F 06 | ^V 16 | & 26 | 6 36 | F 46 | V 56 | f 66 | v 76 |
| ^G 07 | ^W 17 | ' 27 | 7 37 | G 47 | W 57 | g 67 | w 77 |
| ^H 08 | ^X 18 | ( 28 | 8 38 | H 48 | X 58 | h 68 | x 78 |
| ^I 09 | ^Y 19 | ) 29 | 9 39 | I 49 | Y 59 | i 69 | y 79 |
| ^J 0A | ^Z 1A | \* 2A | : 3A | J 4A | Z 5A | j 6A | z 7A |
| ^K 0B | ^[ 1B | + 2B | ; 3B | K 4B | [ 5B | k 6B | { 7B |
| ^L 0C | ^\ 1C | , 2C | < 3C | L 4C | \ 5C | l 6C | | 7C |
| ^M 0D | ^] 1D | - 2D | = 3D | M 4D | ] 5D | m 6D | } 7D |
| ^N 0E | ^^ 1E | . 2E | > 3E | N 4E | ^ 5E | n 6E | ~ 7E |
| ^O 0F | ^\_ 1F | / 2F | ? 3F | O 4F | \_ 5F | o 6F |  7F |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| А 80 | Р 90 | а A0 | ░ B0 | └ C0 | ╨ D0 | Р E0 | Ё F0 |
| Б 81 | С 91 | б A1 | ▒ B1 | ┴ C1 | ╤ D1 | С E1 | ё F1 |
| В 82 | Т 92 | в A2 | ▓ B2 | ┬ C2 | ╥ D2 | т E2 | Є F2 |
| Г 83 | У 93 | г A3 | │ B3 | ├ C3 | ╙ D3 | у E3 | є F3 |
| Д 84 | Ф 94 | д A4 | ┤ B4 | ─ C4 | ╘ D4 | ф E4 | Ї F4 |
| Е 85 | Х 95 | е A5 | ╡ B5 | ┼ C5 | ╒ D5 | х E5 | ї F5 |
| Ж 86 | Ц 96 | ж A6 | ╢ B6 | ╞ C6 | ╓ D6 | Ц E6 | Ў F6 |
| З 87 | Ч 97 | з A7 | ╖ B7 | ╟ C7 | ╫ D7 | ч E7 | ў F7 |
| И 88 | Ш 98 | и A8 | ╕ B8 | ╚ C8 | ╪ D8 | ш E8 | ° F8 |
| Й 89 | Щ 99 | й A9 | ╣ B9 | ╔ C9 | ┘ D9 | щ E9 | ∙ F9 |
| К 8A | Ъ 9A | к AA | ║ BA | ╩ CA | ┌ DA | ъ EA | · FA |
| Л 8B | Ы 9B | л AB | ╗ BB | ╦ CB | █ DB | ы EB | √ FB |
| М 8C | Ь 9C | м AC | ╝ BC | ╠ CC | ▄ DC | ь EC | № FC |
| Н 8D | Э 9D | н AD | ╜ BD | ═ CD | ▌ DD | э ED | ¤ FD |
| О 8E | Ю 9E | о AE | ╛ BE | ╬ CE | ▐ DE | ю EE | ■ FE |
| П 8F | Я 9F | п AF | ┐ BF | ╧ CF | ▀ DF | я EF | FF |

**ПРИЛОЖЕНИЕ J.**

**ИНСТАЛЛЯЦИЯ**

**Инсталляция терминала**

Перед тем, как начать работать с Турбо-Паскалем, его необходимо инсталлировать для вашего конкретного терминала, т.е. снабдить Турбо-систему информацией об управляющих символах, требуемых для выполнения тех или иных функций. Такая настройка легко выполняется с помощью программы TINST, которая обсуждается в настоящем приложении.

Сделав рабочую копию дистрибутивной дискеты, уберите последнюю в безопасное место и в дальнейшем работайте только с этой рабочей копией.

Теперь можно запустить процедуру инсталляции. Для этого наберите на клавиатуре Вашего терминала TINST. Выберите режим инсталляции экрана (Screen - экран) из основного меню. На экране появляются названия ряда терминалов, и пользователь должен выбрать один из них путем ввода его номера в этом списке.

Если Ваш терминал указан в меню, то Вам достаточно просто ввести соответствующий номер, и инсталляция будет выполнена. Перед выполнением собственно процесса инсталляции система задаст Вам следующий вопрос:

Do you want to modify this definition before installation?

Здесь Вы имеете возможность изменить одно или более значений, которые устанавливаются в соответствии с нижеследующим описанием. Если Вы не собираетесь модифицировать определение терминала, то нажмите клавишу N, и инсталляция завершится вопросом о рабочей частоте Вашего ЦП (смотри последний пункт настоящего приложения).

В случае, если Ваш терминал отсутствует в меню, Вам придется самому определить требуемые значения. Значения эти Вы скорее всего сможете найти в документации по Вашему терминалу.

Введите номер, соответствующий пункту "None of the above" - ни один из вышеперечисленных - и ответьте на вопросы по мере их появления (по одному) на экране.

Ниже каждая команда, которую Вы можете смонтировать, обсуждается в деталях. Возможно, Ваш терминал поддерживает не все команды, которые могут быть смонтированы. Если это так, то для обхода ненужных команд нажимайте клавишу RETURN (ВК) в ответ на соответствующую подсказку. Если команды "Удалить строку", "Вставить строку" или "Стереть до конца строки" не установлены, то эти функции будут эмулироваться программно, и это в некоторой степени замедлит работу экрана.

Команды можно вводить либо путем нажатия соответствующих клавиш, либо путем ввода десятичного или шестнадцатеричного значения (ASCII) команды. Например, если в команде требуются два символа: ESCAPE и "=", то имеются две возможности:

- либо нажать сначала клавишу ESC, а затем клавишу = ; на экран при этом будет выведено соответствующее эхо, т.е.

<ESC>= ;

- либо ввести 10-ные или 16-ные значения, разделенные пробелами; 16-ричным значениям должен предшествовать знак доллара, например, 27 61 или $1B 61 или $1B $3D - все это эквивалентные определения.

Эти два метода определения команд нельзя смешивать, т.е. если введен нечисловой знак, то остальная часть этой команды должна быть определена таким же методом, и наоборот.

Знак "дефис", введенный в качестве самого первого знака команды, используется для удаления команды и дает эхо "Nothing".

**Terminal type:**

Введите название терминала, который Вы собираетесь инсталлировать. После окончания работы программы TINST все значения будут запомнены, и имя терминала появится в исходном списке терминалов. если в последствии Вам понадобится вновь инсталлировать Турбо-Паскаль на этом терминале, то Вы сможете сделать это, выбрав соответствующий пункт из меню.

**Send an initialization string to the terminal?**

Если Вам нужна инициализация Вашего терминала при запуске Турбо-Паскаля (например, для программирования функциональных клавиш), то на этот вопрос нужно ответить "Y" (Yes). В противном случае следует нажать RETURN.

**Send a reset string to the terminal?**

Определите строку, которую нужно передавать на терминал при окончании работы Турбо-Паскаля. Здесь применимо описание команды инициализации (предыдущая команда).

**CURSOR LEAD-IN command:**

Это специальная последовательность знаков, которая сообщает терминалу, что символы, идущие вслед за ней, представляют собой адрес места на экране, куда должен переместиться курсор.

При определении этой команды система задаст Вам несколько дополнительных вопросов.

**CURSOR POSITIONING COMMAND to send between line and column:**

Некоторые терминалы нуждаются в дополнительной команде между двумя числами, определяющими строку и столбец в адресе курсора.

**CURSOR POSITIONING COMMAND to send after line and column:**

Некоторые терминалы нуждаются в дополнительной команде после двух чисел, определяющих строку и колонку в адресе курсора.

**Column first ?**

Большинство терминалов требуют адрес в следующем формате: сначала ROW (строка), затем COLUMN (столбец). Если с Вашим терминалом дело обстоит именно так, то нажмите N. Если же для него сначала должно идти COLUMN, затем ROW, то введите Y.

**OFFSET to add to LINE:**

Введите число, которое должно прибавляться к адресу строки (ROW или LINE).

**OFFSET to add to COLUMN:**

Введите число, которое должно прибавляться к адресу столбца (COLUMN).

**Binary address ?**

Большинству терминалов требуется передавать адрес курсора в двоичной форме. если именно так обстоит дело с Вашим терминалом, то введите ответ Y. если ваш терминал принимает адрес курсора в виде ASCII-цифр, то введите ответ N. При этом Вам будет задан дополнительный вопрос:

**Number of ASCII digits (2 or 3):**

Здесь Вам нужно указать количество цифр в адресе курсора для Вашего терминала.

**CLEAR SCREEN command:**

Введите команду, которая очищает весь экран Вашего терминала, как передний план, так и фон, если эти понятия применимы к Вашему терминалу.

**Does CLEAR SCREEN also HOME cursor ?**

Обычно это имеет место, т.е. команда очистки экрана одновременно переводит курсор в исходное положение. Если для Вашего терминала это не так, то введите ответ N и определите команду перевода курсора в исходное положение (HOME).

**DELETE LINE command:**

Введите команду, которая удаляет всю строку, расположенную в текущей позиции курсора.

**INSERT LINE command:**

Введите команду, которая вставляет строку в текущей позиции курсора.

**ERASE TO END OF LINE command:**

Введите команду, которая удаляет часть строки, начиная с текущей позиции курсора и до конца текущей строки.

**START HIGHLIGHTING command:**

Если Ваш терминал поддерживает различные интенсивности видеоизображения, то здесь Вам нужно определить команду, которая инициирует видео-атрибут яркого высвечивания символов. Если эта команда определена, то Вам будет задан еще один вопрос:

**END HIGHLIGHTING command:**

Определите команду, которая выключает повышенную интенсивность свечения символов на экране.

**Number of rows (lines) on your screen:**

Введите число горизонтальных рядов (строк) на экране Вашего терминала.

**Number of columns on your screen:**

Введите число позиций по вертикали (столбцов) на Вашем экране.

**Delay after CURSOR ADDRESS (0-255 ms):**

**Delay after CLEAR, DELETE and INSERT (0-255 ms):**

**Delay after ERASE TO END OF LINE and HIGHLIGHT (0-255 ms):**

Введите задержку в миллисекундах для всех трех указанных групп функций. Нажатие RETURN означает 0 (отсутствие задержки).

**IS this definition correct ?**

Если в определениях Вы сделали какие-либо ошибки, введите ответ N. Тогда система возвратит Вас к меню выбора терминала. Данные об инсталляции, которые Вы только что ввели, будут включены в файл TINST.DTA и появятся в меню выбора терминала, но инсталляция выполнена НЕ БУДЕТ. Если же в ответ на этот вопрос ввести Y, то будет задан еще один вопрос:

**Operating frequency of your microprocessor in MHz (for delays):**

Поскольку задержки, определенные ранее, зависят от рабочей частоты Вашего микропроцессора, Вам необходимо задать это значение. Кроме того, данный параметр влияет на время выполнения стандартной процедуры DELAY.

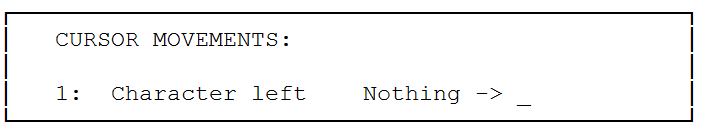
На этом инсталляция заканчивается, соответствующие данные запоминаются в Турбо-Паскале, и система возвращает Вас к основному меню инсталляции.

**Инсталляция команд встроенного редактора**

Встроенный редактор системы Турбо-Паскаль реагирует на целый ряд команд, которые используются для перемещения курсора по экрану, удаления и вставки текста, перемещения текста и т.д. Эти команды установлены фирмой "Борланд" в соответствии со "стандартом", введенным редактором WordStar, однако с помощью программы инсталляции их легко можно переопределить в соответствии с Вашими вкусами и Вашей клавиатурой.

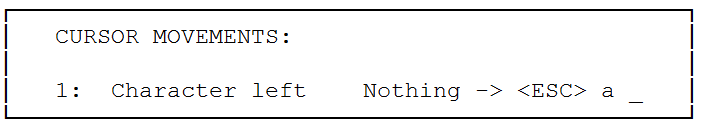
После нажатия C (Command installation - настройка команд)

появляется первая команда:



Это означает, что для перемещения курсора на один знак влево не установлено никакой команды пользователя. Если Вам нужно инсталлировать некоторую команду (в дополнение к стандартной команде CTRL-S, такой же как в WordStar и не показанной здесь), то Вы можете ввести требуемую команду после подсказки – одним из двух показанных ниже способов.

1). Просто нажмите ту клавишу, которой Вы собираетесь пользоваться. Это может быть функциональная клавиша (например, клавиша со стрелкой, направленной влево, если она у Вас имеется) или любая другая клавиша или последовательность клавиш, которую Вы наберете (не более 4). Программа инсталляции реагирует на это выводом мнемоники каждого символа, который она получает с клавиатуры. Если на Вашей клавиатуре имеется клавиша со стрелкой влево, которая передает ESCAPE и вслед за ним букву "a" нижнего регистра, то при нажатии на эту клавишу в вышеописанной ситуации экран будет выглядеть следующим образом:



2). Вместо того, чтобы нажимать ту клавишу, которую Вы собираетесь использовать, можно ввести ASCII-значения для всех символов, которые участвуют в команде. Значения для нескольких символов при вводе отделяются друг от друга пробелами. Десятичные значения вводятся как обычно, например: 27; шестнадцатеричные значения предваряются знаком доллара: $1B. Этим методом можно воспользоваться для инсталляции команд, которые в данный момент недоступны на Вашей клавиатуре, например, при установке значений для нового терминала, в то время, как Вы еще работаете на старом. Эта возможность используется весьма редко, поскольку в самом деле нет большого смысла в том, чтобы определить такую команду, которую невозможно сгенерировать нажатием на некоторую клавишу. Тем не менее, эта функция имеется в системе для тех, кому она нужна.

В обоих случаях закончить ввод следует нажатием клавиши RETURN. Заметим, что эти два метода нельзя смешивать внутри одной команды. Если определение командной последовательности начато нажатием клавиш, то все знаки в этой команде должны быть определены путем нажатия клавиш, и наоборот.

Для вычеркивания команды из списка можно ввести знак "-" (минус), а для возврата на один элемент списка (т.е. к предыдущему элементу) можно нажать клавишу B .

Редактор реагирует в общей сложности на 45 команд, и все они могут быть настроены в соответствии с Вашими спецификациями. Если в процессе инсталляции сделана ошибка, такая как определение одной команды для двух разных целей, то система выдает содержательное сообщение, и пользователь должен скорректировать ошибку перед окончанием процедуры инсталляции.

Ниже показана таблица, которая содержит перечень исходных команд; в этой же таблице можно пометить все команды, установленные лично Вами.

Таблица J-1

|  |  |  |  |
| --- | --- | --- | --- |
| N | Команда | Клавиши | Примечание |
|  | **Команды перемещения курсора** |  |  |
| 1 | На один символ влево | CTRL-S |  |
| 2 | Альтернативная | CTRL-H |  |
| 3 | На один символ вправо | CTRL-D |  |
| 4 | На слово влево | CTRL-A |  |
| 5 | На слово вправо | CTRL-F |  |
| 6 | На строку вверх | CTRL-E |  |
| 7 | На строку вниз | CTRL-X |  |
| 8 | Скроллинг вверх | CTRL-W |  |
| 9 | Скроллинг вниз | CTRL-Z |  |
| 10 | На страницу вверх | CTRL-R |  |
| 11 | На страницу вниз | CTRL-C |  |
| 12 | В начало строки | CTRL-Q CTRL-S |  |
| 13 | В конец строки | CTRL-Q CTRL-D |  |
| 14 | В начало страницы | CTRL-Q CTRL-E |  |
| 15 | В конец страницы | CTRL-Q CTRL-X |  |
| 16 | В начало текста | CTRL-Q CTRL-R |  |
| 17 | В конец текста | CTRL-Q CTRL-C |  |
| 18 | В начало блока | CTRL-Q CTRL-B |  |
| 19 | В конец блока | CTRL-Q CTRL-K |  |
| 20 | В последнюю позицию курсора | CTRL-Q CTRL-P |  |
| 21 | Включение/выключение |  |  |
|  | режима вставки | CTRL-V |  |
| 22 | Вставка строки | CTRL-N |  |
| 23 | Удаление строки | CTRL-Y |  |
| 24 | Удаление до конца строки | CTRL-Q CTRL-Y |  |
| 25 | Удаление слова справа | CTRL-T |  |
| 26 | Удаление символа под курсором | CTRL-G |  |
| 27 | Удаление символа слева | <DEL> |  |
| 28 | Альтернативная | отсутствует |  |
|  | **Блочные команды** |  |  |
| 29 | Пометить начало блока | CTRL-K CTRL-B |  |
| 30 | Пометить конец блока | CTRL-K CTRL-K |  |
| 31 | Пометить одиночное слово | CTRL-K CTRL-T |  |
| 32 | Скрыть/показать блок | CTRL-K CTRL-H |  |
| 33 | Скопировать блок | CTRL-K CTRL-C |  |
| 34 | Переместить блок | CTRL-K CTRL-V |  |
| 35 | Удалить блок | CTRL-K CTRL-Y |  |
| 36 | Считать блок с диска | CTRL-K CTRL-R |  |
| 37 | Записать блок на диск | CTRL-K CTRL-W |  |
|  |  |  |  |
|  | **Прочие команды** |  |  |
| 38 | Закончить редактирование | CTRL-K CTRL-D |  |
| 39 | Табуляция | CTRL-I |  |
| 40 | Включить/выключить автоотступ | CTRL-Q CTRL-I |  |
| 41 | Восстановить строку | CTRL-Q CTRL-L |  |
| 42 | Поиск | CTRL-Q CTRL-F |  |
| 43 | Поиск и замена | CTRL-Q CTRL-A |  |
| 44 | Повторить последний поиск | CTRL-L |  |
| 45 | Префикс управляющего символа | CTRL-P |  |

Позиции 2 и 28 дают пользователю возможность определить альтернативные команды для команд "На один символ влево" и "Удалить символ слева". Обычно <BS> является альтернативой для CTRL-S, а для <DEL> альтернатива не определена. Вы можете инсталлировать свои команды так, чтобы они подходили к Вашей клавиатуре, например, можно воспользоваться клавишей <BS> в качестве альтернативы для <DEL>, если первая расположена более удобным образом. Безусловно, две альтернативные команды должны оставаться недвусмысленными, как и все прочие команды.

**ПРИЛОЖЕНИЕ K.**

**СОВЕТЫ И ОТВЕТЫ НА ВОПРОСЫ**

В этом приложении содержится перечень наиболее часто возникающих вопросов и приводятся соответствующие ответы и рекомендации.

**Вопрос**: Как использовать Турбо-Паскаль?

**Ответ**: Прочтите эту книгу, в особенности главу 1. Если Вам необходимо начать работу немедленно, то сделайте следующее:

- загрузите операционную систему;

- если Ваша Турбо-система не инсталлирована, то вы-

полните процедуру начальной настройки, описанную в главе 1;

- запустите TURBO;

- начинайте программировать !

**Вопрос**: Как поменять дискеты во время работы с Турбо-системой?

**Ответ**: Всякий раз при включении Вашего компьютера он считывает первые две дорожки с CP/M-дискеты и загружает в оперативную память копию операционной системы CP/M. При каждой перезагрузке с диска CP/M также создает список дискового пространства, доступного на каждом дисководе. При попытке записи какого-либо файла на диск CP/M проверяет, не было ли смены дискет. Так, например, если Вы заменили дискету в дисководе A без перезагрузки, то при попытке записи на этот диск CP/M выведет следующее сообщение об ошибке:

BDOS ERROR ON A: R/O

Управление будет передано операционной системе, а Ваша работа НЕ БУДЕТ ЗАПИСАНА НА ДИСК. Все это порождает некоторую путаницу для начинающих при копировании дискет. Указанных неприятностей можно избежать, если после каждой смены дискет делать перезагрузку дисковой операционной системы. Если Вы работаете в Турбо-системе, для этой цели используйте команду L из главного меню (смотри подраздел "Выбор текущего дисковода" в главе 1).

**Вопрос**: Необходима ли Турбо-система для выполнения программ, скомпилированных в среде этой системы?

**Ответ**: Нет, если Вы осуществляете компиляцию в .COM или .CHN-файлы.

**Вопрос**: Как создать .COM-файл ?

**Ответ**: Выберите режим O в главном меню ("Опции компилятора"), а затем введите команды C (COM-файл) и Q (выход из меню опций).

**Вопрос**: Что делать, когда возникает ошибка 99 (переполнение компилятора)?

**Ответ**: У Вас есть две возможности: разбить текст на фрагменты меньшего размера, воспользовавшись директивой компилятора $I (об этом говорится в главе 17) или выполнить компиляцию в .COM-файл.

**Вопрос**: Что делать, если размер объектного кода должен быть больше, чем размер доступной оперативной памяти?

**Ответ**: Вам следует воспользоваться либо цепным вызовом программ (процедурой CHAIN), либо оверлейной структурой.

**Вопрос**: Как выполнить чтение с клавиатуры без нажатия на клавишу RETURN (ВК) подобно функции INKEY$ в Бейсике?

**Ответ**: это делается посредством вызова Read(Kbd,Ch), где Ch имеет тип CHAR.

**Вопрос**: Как направить вывод на принтер?

**Ответ**: Попробуйте сделать так: WriteLn(Lst,...) .

**Вопрос**: Как вывести листинг текста программы на принтер?

**Ответ**: Если на Вашем диске нет программы TLIST.COM, которая распечатывает листинг на принтер с такими "косметическими" украшениями, как подчеркивание или выделение зарезервированных слов, разделение на страницы и раскрытие всех INCLUDE-файлов, то Вы можете воспользоваться показанной ниже простой программой.

program TextFilePrint;

var

TextFile: text;

Scratch: string[128];

begin

write('Имя печатаемого файла: ');

ReadLn(Scratch); { Ввод имени файла }

Assign(TextFile,Scratch); { Подготовить файл }

{$I-}

Reset(TextFile);

{$I+}

if IOresult <> 0 then

writeln('Файл ',Scratch,' не найден!')

else

begin { Печать файла }

while not Eof(TextFile) do

begin

ReadLn(TextFile,Scratch); { Чтение }

WriteLn(Lst,Scratch); { Печать строки }

end;

WriteLn(Lst);{ Опорожнение буфера принтера }

Close(TextFile);

end; { else }

end.

**Вопрос**: Почему мои рекурсивные процедуры не работают?

**Ответ**: Выключите A-директиву компилятора: {$A-} .

**Вопрос**: Как мне воспользоваться функциями EOF и EOLN без файловой переменной в качестве параметра?

**Ответ**: Выключите буферизацию ввода: {$B-}.

**Вопрос**: Как мне выяснить, существует ли данный файл на диске?

**Ответ**: Используйте директиву I в состояниях {$I-} и {$I+}. Ниже показана функция, которая возвращает TRUE, если имя файла, переданное ей в качестве параметра, имеется в каталоге диске; в противном случае возвращается FALSE.

type

Name: string[14];

:

:

function Exist (FileName: name): Boolean;

var

Fil: file;

begin

Assign(Fil,FileName);

{$I-} Reset(Fil); {$I+}

Exist:=(IOresult=0);

{$I-} Close(Fil); {$I+}

end;

**Вопрос**: Как заблокировать действие нажатия CTRL-C?

**Ответ**: С помощью директивы компилятора: {$C-}.

**Вопрос**: У меня возникает ошибка "Несоответствие типов" (Type mismatch) при передаче строки в качестве параметра процедуры или функции.

**Ответ**: Выключите контроль типов для параметров-переменных:{$V-} .

**Вопрос**: При компиляции моей программы у меня возникает ошибка "Файл не найден" - для INCLUDE-файла, несмотря на то, что файл есть в оглавлении.

**Ответ**: При использовании директивы включения файлов I расширение имени файла (тип файла), если оно содержит меньше трех символов, должно отделяться от закрывающей фигурной скобки или символов \*) пробелом:

{$I SAMPLE.F}

В противном случае закрывающая скобка будет интерпретироваться как часть имени файла.

**Вопрос**: Почему моя программа ведет себя неодинаково, когда я запускаю ее подряд несколько раз?

**Ответ**: Если Вы исполняете программы в режиме M (Memory) т.е. в режиме непосредственного исполнения, и пользуетесь константами с заданным типом в качестве инициализированных переменных, то эти константы будут инициализированы только сразу после компиляции, а не при каждом запуске по команде R (Run), поскольку они располагаются в области кода. Для .COM-файлов такой проблемы не существует, но если у Вас все же возникают различные результаты при использовании массивов или множеств, то попробуйте включить контроль границ: {$R+} .

**Вопрос**: При использовании данных типа REAL и INTEGER в одном выражении я не получаю тех результатов, которые, по моему мнению, должен был бы получить.

**Ответ**: Когда выражение типа INTEGER присваивается переменной типа REAL, это ыражение преобразуется в тип REAL. Однако само выражение вычисляется как целое, поэтому следует быть готовым к возможному целому переполнению в выражении. Это может привести к неожиданным результатам. Возьмем, например:

RealVar:=40\*1000;

Сначала 40 умножается на 1000, и получается 40000, что дает целое переполнение. Поскольку целые в таких ситуациях усекаются циклически, это даст в результате -25536, так что переменная RealVar получит значение -25536. Чтобы этого не произошло, можно записать такое выражение:

RealVar:=40.0\*1000;

или

Realvar:=1.0 \* IntVar1 \* IntVar2;

Тем самым будет обеспечено, что выражение будет вычисляться как REAL.

Примечание. Если Вы занимаетесь распространением своих программ, то Вам следует включать подобные инструкции в выпускаемую Вами документацию.