1. **Стандартная библиотека BDS-C В CP/М.**

 В пакете bds-c файлы deff.crl и deff2.crl содержат объектные коды стандартной библиотеки. Она является собранием полезных с-функций в crl-формате, доступных во всех с-программах. clink автоматически ищет библиотечные файлы после поиска всех других crl-файлов, явно указанных в командной строке. Поэтому, функции, явно определенные во входном файле и названные так же, как и библиотечные функции, перекрывают библиотечные версии, т.к. clink находит ваши версии перед просмотром библиотек.

В приводимом перечне всех основных функций deff.crl и deff2.crl каждая функция описано словесно и в с-подобной нотации, использованной для иллюстрации того, как описания этих функций должны появляться в с-программах. Такая нотация обеспечивает, с одного взгляда, информацией возвращает ли функция значение (и если да, то какого типа) и какие типы имеют параметры функции.

Есть несколько правил: если функция приведена без указания типа, то она не возвращает значение (для примера, exit и poke не возвращают значений). Формальные параметры, отсутствующие в описании, подразумеваются имеющими тип int, хотя во многих случаях используется только младший байт параметра и функции может передаваться значение типа char.

Замечали, что не всегда легко описать тип формального параметра: указатель памяти как без знаковое или как символьное ? не пытайтесь передавать символьную переменную вместо параметра, являющегося 16-битным указателем памяти, думая, что это будет правильно работать, без учета описания типа в вызывающей программе.

Есть только несколько случаев, когда необходимо описывать библиотечные функции перед их использованием в с-программе. один случай, когда функция имеет тип, отличный от int, и вызов функции расположен в выражении, в котором тип возвращаемого значения должен отличаться от int при обычной обработке выражения (для примера, адресная арифметика). небольшой опыт поможет уяснять, когда действительно необходимо описывать функции; многие из этих решений имеют значение для стиля и портативности.

 deff.crl содержит все с-функции из stdlib1.c и stdlib2.c. deff2.crl

содержит все ассемблерные функции из deff2a.csm, deff2b.csm и deff2c.csm

(ассемблированные с помощью casm).

1. **Краткое описание функций.**

Здесь приводится краткое описание всех основных функций, имеющихся в deff.crl и deff2.crl.

**Функции общего назначения:**

**char csw()**

Возвращает байтовое значение (0-255) регистра переключения консоли (порт 0xff в некоторых системах)

**еxit()**

Закрывает любые открытые файлы и выходит из выполняющейся программы, выполняет горячий старт cp/m. не делает автоматического вызова flush с файлами, открытыми для буферированного вывода.

**int bdos(c,de)**

Вызывает точку входа стандартной bdos-системы (адрес 0005h в большинстве систем), устанавливая регистр с на значение с и регистровую пару de на значение de. Возвращается 16-битное значение, возвращаемое bdos в регистрах hl. Для cp/m систем младший байт является значением, возвращаемым bdos в регистре А, a старший байт - значением из регистра В (или 0 для 8-битных значений). Смотрите приложение "различные замечания" для уточнения несовместимости с не cp/m системами (например, sdos).

**char bios(n,c)**

Вызывает n-й вход таблицы вектора переходов bios, где n=0 для первого входа (boot), n=1 для второго входа (wboot), n=3 для третьего входа (const) и т.д., устанавливая регистровую пару BC на значение С. Результатом является значение, возвращенное bios в регистре А. Заметим, что функция холодного старта (n=0) фактически никогда не может быть использована, поскольку при этом входе будет задет процессор консольных команд и возможно разрушение системы. Имеются вызовы bios, запрашивающие парaметры в регистровой паре DE и возвращающие результат в регистрах HL. для этих вызовов используется функция biosh.

**unsigned biosh(n,bc,de)**

Вызывает n-й вход таблицы вектора переходов bios, устанавливая регистры BC на значение BC и регистры DE на значение DE. Результатом является значение, возвращенное bios в регистрах HL.

**char peek(n)**

Возвращает содержимое памяти с адресом n. заметим, что в применениях, где требуется рассмотрение последовательности байтов, более эффективно использовать указатель символов, чем peek. Эта функция предназначена для редких применений, когда затруднительно описывать указатель, присваивать ему адрес и использовать косвенный доступ к байту.

**poke(n,b)**

Заносит младшие 8 бит из В в ячейку памяти с адресом n. Это можно выполнить более эффективно, используя указатели, как в \*n=b; (где n является указателем на символ)

**inp(n)**

Возвращает 8-битное значение, присутствующее во входном порте n. Для ввода по карте памяти используется функция peek.

**outp(n,b)**

В выходной порт n выводится 8-битное значение b. Для вывода по карте памяти используется функция pook.

**pause()**

Ожидает в цикле, пока с консоли не будет введен любой символ. Сам символ не забирается; перед повторным использованием pause необходимо вызвать функцию getchar для сброса статуса консоли. Возвращаемого значения нет.

**sleep(n)**

Спит (работает вхолостую) n/20 секунд при частоте 4 мгц, или n/10 секунд при частоте 2 мгц. Единственный способ прерваться раньше времени – ввести <ctrl>c, который снимет программу и вернет на командный уровень. Возвращаемого значения нет.

**call(addr,a,h,b,d)**

Вызывает машинную подпрограмму по адресу addr, устанавливая регистры цп следующим образом:

 HL <-- h

 A <-- a

 BC <-- b

 DE <-- d

 Возвращается значение, устанавливаемое подпрограммой в регистрах HL. Подпрограмма обязана поддерживать стековую дисциплину.

**char calla(addr,a,h,b,d)**

Подобна функции call, за исключением результата, который возвращается подпрограммой в регистре А (вместо регистров HL).

**int abs(n)**

Возвращается абсолютное значение n.

**int max(n1,n2)**

Возвращается большее из двух чисел.

**int min(n1,n2)**

Возвращается меньшее из двух чисел.

**srand(n)**

Если n отлично от нуля, функция инициализирует генератор псевдослучайных чисел, устанавливая внутренний счетчик на значение n. Если n=0, то функция выводит сообщение, просящее пользователя ввести <et>, и начинает внутренний счет. когда пользователь нажмет клавишу, текущее значение счетчика используется для инициализации генератора. Символ, введенный пользователем, теряется; готовность консоли сбрасывается.

**srandl(string)**

**char \*string;**

Как srand(0), кроме сообщения "hit return after a few seconds", указанная строка используется как подсказка. В отличие от srand введенный символ не выбирается; нужно выполнять getchar для взятия символа и/или сброса готовности консоли.

**int rand()**

Возвращает следующее значение ( диапазон: 0 < rand() < 32768 ) из последовательности псевдослучайных чисел, инициализированной функциями srand или srandl.

**nrand(-1,s1,s2,s3)**

**nrand(0,prompt-string)**

**int nrand(1)**

Новый, лучшего качества, генератор случайных чисел, написанный профессором Паулем Гансом для эмуляции генератора cdc 6600. Механизм инициализации был добавлен позднее для совместимости с соглашениями srand и srandl. Первая форма устанавливает внутренний 48-битный счетчик равным 48 битам данных, указанных как s1, s2 и s3 (целых или беззнаковых). Вторая форма используется как srandl: строка promt-string выводится на консоль, затем машина ждет ввода пользователя, постоянно наращивая внутренний 16-битный счетчик. После ввода символа значение счетчика заносится в 48-битный регистр. Заметим, что консольный ввод не очищается (требуется getchar). Последняя форма возвращает очередное случайное число в диапазоне 0 < nrand(1) < 32768. заметим, что внутренний счетчик nrand отделен от счетчика, используемого srand, srandl и rand, который использует первые 32 бита области, именуемой в пакете выполнения идентификатором rseed. nrand пользуется своим внутренним недоступным счетчиком.

**setmem(addr,count,byte)**

**char byte, \*addr;**

Заносит в count байтов памяти, начинающихся с адреса addr, значение byte. Это эффективно для быстрой инициализации массивов и буферных областей.

**movmem(source,dest,count)**

**char \*source,\*dest;**

Пересылает блок памяти длиной count байтов из source в dest. Эта будет правильно работать при любом расположении исходной и принимающей областей, автоматически определяя, как выполнять пересылку блока: с начала в начало или с конца в конец. при выполнении на z-80 используется команда "пересылка блока". при выполнении на 8080 или 8085 используются обычные команды. Все это происходит автоматически.

**qsort(base,nel,width,compar)**

**char \*base;**

**int (\*compar)();**

Производится сортировка шелла для данных, начинающихся с base, содержащих nel элементов длиной по width байтов. compar должен быть указателем функции от двух указателей (например, x и y), которая возвращает

 1 если \*x > \*y

 -1 если \*x < \*y

 0 если \*x == \*y.

 элементы сортируются в возрастающем порядке.

**int exec(prog)**

**char \*prog;**

Связывает (загружает и выполняет) программу prog.com. prog должен быть указателем строки, содержащей имя вызываемого файла и заканчивающейся нулем (в имени не должно присутствовать .com). Строковая константа (такая как "foo") вполне приемлема, поскольку она превращается в указатель. Если выполняемая программа сгенерирована с-компилятором и разделяет внешние переменные с вызывающей программой, то она должна редактироваться с опцией clink -е для размещения общих внешних данных на одинаковый адрес. При выполнении exec файл может быть не открыт. допустимыми совместными ресурсами являются только внешние данные. При ошибках возвращается -1, однако потом, когда заканчивается все, будет ошибка.

**int execl(prog,arg1,arg2,...,0)**

**char \*prog,\*arg1,\*arg2,...;**

Позволяет связывать две с-программы с передачей параметров через механизм argc и argv. prog должен быть указателем законченной нулем строки, именующей вызываемый com-файл (.com не должно быть в имени), и каждый аргумент также должен быть указателем строки, завершенной нулем. Последний аргумент должен быть нулем. Еxecl работает создавая командную строку с заданными параметрами, и работает так, как будто бы пользователь ввел командную строку в процессор команд cp/м. например,

 execl("foo", "bar", "zot", 0)

 будет иметь такой же эффект, как и командная строка cp/m

 a>foo bar zot <et>

 введенная непосредственно. встроенные в cp/m команды (такие как dir, era и т.д.) не могут быть вызваны по execl. Общая длина командной строки, созданной из заданных строк аргументов, не может превышать 80 символов. если созданная командная строка превышает эту длину, на консоль выведется сообщение и программа будет снята. При ошибках возвращается -1.

**execv(filename,argvector)**

**char \*filename;**

**char \*argvector[];**

Эта функция разрешает выполнять связывание с переменным числом аргументов, подобно execl, за исключением того, что текст параметров задается в массиве, а не явно в вызове. параметр argvector должен быть указателем массива из указателей строк, в котором каждый указатель строки указывает на очередной аргумент, а последний указатель имеет нулевое значение (отличающееся от указателя к нулевой строке).

**int swapin(filename,addr)**

**char \*filename;**

Загружает файл, имя которого указывается законченной нулем строкой, в память по адресу addr. Нет проверки, входит ли файл в память; будьте осторожны, загружая файл! Эта функция может использоваться, например, для загрузки оверлейного сегмента с целью последующего выполнения через переменную, указывающую на функцию. При наличии ошибок чтения файла возвращается -1. управление не передается загруженному файлу.

**char \*codend()**

Возвращает указатель на первый байт, следующий после программного кода корневого сегмента. Обычно это будет начало области внешних данных, если не использована опция -е clink для явного указания расположения области внешних данных (смотрите функцию externs).

**char \*externs()**

Возвращает указатель начала области внешних данных. если не использовалась опция -е cc и/или clink, то это значение будет таким же, как и значение функции codend.

**char \*endext()**

Возвращает указатель первого байта, следующего после конца области внешних данных. Он является началом области, из которой функция sbrk берeт свободную память.

**char \*topofmem()**

Возвращает указатель последнего байта памяти пользователя. Обычно это начало стека, который непосредственно примыкает к bdos (если для clink не указывалась опция -n), или непосредственно примыкает к процессору команд (если при редактировании использована опция -n). Возвращаемое функцией значение не поражается использованием опции –т при редактировании.

**char \*alloc(n)**

Возвращает указатель на блок памяти длиной n байтов, либо нуль (нет памяти размером n байтов). Это грубая функция выделения памяти из части 8 "языка программирования c", упрощенная отсутствием ограничений выравнивания типа. Стандартный заголовочный файл bdscio.h должен включаться во все файлы программы, использующей пару alloc и free, поскольку там содержатся описания их некоторых критических внешних данных.

**free(allocptr)**

**char \*allocptr;**

Освобождает блок памяти, выделенный функцией alloc. allocptr является значением, возвращенным предшествующим вызовом alloc. free не должны вызываться в порядке, обратном к вызовам предшествующих alloc, поскольку структура данных связанного списка может допускать любой порядок выделения/освобождения. Нельзя вызывать free с аргументом, не возвращенным ранее функцией alloc.

**char \*sbrk(n)**

Это низкоуровневая функция распределения памяти, используемая alloc для получения куска памяти. она возвращает указатель на n байтов памяти, либо -1, когда памяти нет. Первый вызов sbrk возвращает указатель памяти, следующей непосредственно за областью внешних данных, каждый последующий вызов возвращает блок, смежный с предыдущим, до тех пор, пока sbrk не обнаружит, что адрес выделяемой памяти угрожающе близок к текущему значению указателя стека. По умолчанию, "угрожающая близость" определена в 1000 байтов. Для изменения этого умолчания смотрите следующую функцию. Если вы планируете использовать в программе функции alloc и free, но собираетесь также защитить некоторую память от распределения, используйте для запроса требуемой памяти sbrk вместо alloc). вызов sbrk может быть выполнен в любое время (независимо от произведенных вызовов alloc и free).

**rsvstk(n)**

Эта функция заставляет функции распределения памяти отвергать вызовы выделения, которые оставят менее чем n байтов между концом выделяемой области и текущим значением указателя стека (стек растет в памяти сверху вниз). Если требуется, rsvstk movet вызываться после произведенных вызовов sbrk или alloc. Если rsvstk не использовано, то выделение памяти автоматически превращается при приближении к стеку ближе чем на 1000 байтов (как будто был вызов rsvstk(1000)).

**int setjmp(buffer)**

**char buffer[jbufsize]**

**longjmp(buffer,val)**

**char buffer[jbufsize]**

При вызове setjump текущее состояние процессора запоминается в заданном буфере (символическая константа jbufsize определена в bdscio.h) и возвращается нулевое значение. когда последующий вызов longjmp будет выполнен где-либо в текущей или подчиненной функции, тогда восстановится состояние процессора, запомненное при предшествующем вызове setjmp с тем же буфером в качестве параметра. Программа возобновляет выполнение "возвратом" к первоначальному вызову setjmp, возвращается значение val, переданное в longjmp. Для различения программами вызовов setjmp и передач управления значения, передаваемые в longjmp, должны отличаться от нуля. Типичными применениями setjmp/longjmp являются выход из нескольких уровней функций, следующий без возврата из каждого уровня; обязательное выполнение особых выходов из программы (скажем, dioflush из пакета dio.c).

**Символьный ввод/вывод:**

Пакет функций cio для прямого обмена с консолью функции getchar и putchar, реализованные в стандартной библиотеке (и описанные ниже), не обеспечивают полного управления консолью. Вместо этого они разработаны так, чтобы быть наиболее полезными для общепринятых применений без необходимости в какой-либо инициализации или специальном действии. Если в ваших применениях важно обладать полным контролем над обменом с системной консолью, тогда ассемблируйте и используйте пакет функций cio, обеспеченный только во входном виде (cio.csm) в дистибуции пакета bds-c v1.50. Пакет Функций cio обеспечивает альтернативные версии функций getchar, putchar и kbhit и новую функцию ttymode, которая поддерживает динамическое изменение операционных характеристик интерфейса консоли.

**int getchar()**

Возвращает очередной символ из стандартного входного потока (ввод с консоли cp/m). Перезагружает cp/m, когда вводится <ctrl>-c. Возврат каретки отражается на выводе консоли как cr-lf и возвращает символ новой строки ('\n'). Для <ctrl>z возвращается значение -1; замечу, что значение, возвращаемое getchar, должно обрабатываться как целое (а не символьное), если возможно значение -1. если вместо этого вы опишете getchar возвращающей символьное значение или присвоите возвращенное ею значение символьной переменной, то значение должно проверяться на 255 вместо <ctrl>z, но замечу, что в этом случае обычное значение 255 будет неотличимо от маркера конца файла (eof).

**char ungetch(c)**

Обеспечивает возврат символа с следующему вызову функции getchar. Только один символ может быть возвращен между последовательными вызовами getchar. Обычно возвращается нуль. Если уже имеется возвращенный символ после последнего вызова getchar, то возвращается значение этого символа.

**int kbhit()**

Возвращает истину (не нуль), если на стандартном вводе присутствует символ (нажата клавиша); Иначе возвращается ложь (нуль). присутствующий символ не вводится, это делается последующим вызовом getchar. Замечу, что kbhit возвращает истину и в том случае, когда после последнего вызова getchar была вызвана функция ungetch для возврата символа.

**putchar(c)**

**char c;**

Записывает символ c в стандартный вывод (вывод консоли cp/m). Символ новой строки ('\n') расширяется при выводе в комбинацию cr-lf. Если при вызове putchar на вводе консоли обнаруживается символ <ctrl>-c, то выполнение программы прекращается и управление возвращается на командный уровень, что позволяет пользователю снимать программу, выполняющую вывод на консоль (через вызовы putchar), посредством ввода <ctrl>-c. Поскольку putchar использует вызовы bdos kak для вывода символа, так и для тестирования ввода консоли, специальный символ cp/m для управления потоком (<ctrl>z) опознается и может использоваться для приостановки вывода, производимого через putchar.

**putch(c)**

**char c;**

Подобна putchar, за исключением того, что при выводе не опрашивает ввод консоли на <ctrl> c, разрешая преждевременный ввод при выводе на консоль в системах с драйверами, работающими по прерываниям. Заметим, что этот вывод может быть направлен на устройство cp/m "list" путем ввода <ctrl>-p перед запуском cc.

**Функции работы с файлами:**

**int seek(fd,offset,code)**

Модифицирует указатель чтения/записи, ассоциированный с файлом fd. Если code=0, to указатель устанавливается на запись offset. Если code=1, to указатель устанавливается на его текущее значение плюс offset (которое может быть отрицательным). Если code=2, to указатель устанавливается на eof плюс offset. для установки на существующую запись файла offset должно быть отрицательным. если code=2 и offset=0, to указатель устанавливается для пополнения файла. Возвращенное значение -1 индицирует, что bdos определила какую-то ошибку при установке относительно eof. Функция errno обеспечит вас подробностями о случившихся ошибках.

**int tell(fd)**

Возвращает значение указателя чтения/записи, ассоциированного с файлом fd. Это число указывает номер следующей записи, подлежащей обмену с файлом (нумерация записей файла начинается с нуля).

**int unlink(filename)**

**char \*filename;**

Удаляет указанный файл из файловой системы. Используйте с осторожностью!

**int rename(old, new)**

**char \*old, \*new;**

Переименовывает файл. При переименовании файл не должен быть открытым. При ошибке возвращает -1.

**int fabort(fd)**

Освобождает дескриптор файла fd без закрытия ассоциированного с ним файла. Если файл был открыт только для чтения, то это не будет давать последствий для файла. Если файл был открыт для записи, то все изменения, сделанные в текущем экстенте после его открытия, будут проигнорированы; а изменения в других экстентах запомнятся. не выполняйте fabort для файла открытого для записи, если вы не желаете терять записанные в него данные.

**unsigned cfsize(fd)**

Вычисляет точный размер (в 128-байтовых записях) заданного открытого файла без изменения указателя чтения/записи, ассоциированного с файлом. замечу, что размер, возвращаемый cfsize, в отличие то функции 35 bdos, отражает данные, записанные на новые экстенты и затем закрытые.

**int oflow(fd)**

Возвращает истину (не нуль), если произошло переполнение старшего (3-го) байта поля номера записи в блоке управления файлом указанного открытого файла.

**int errno()**

Возвращает код ошибки, обнаруженной при выполнении последней операции обмена с файлами. смотрите список сообщений об ошибках, ассоциированных с их кодами.

**char \*errmsg(errnum)**

Беря код ошибки, возвращенный функцией errno, эта функция возвращает указатель строки, описывающей указанную ошибку на английском языке. Перечень возможных кодов ошибок вместе с текстами ошибок:

 код ошибки текст

 0 no error has occurred yet

 1 reading unwriten data

 2 disk out of data space

 3 cann't close current extent

 4 seek to unwritten extent

 5 can't create new extent

 6 seek past end of disk

 7 bad file descriptor given

 8 file not open for read

 9 file not open for write

 10 no file descriptor slots left

 11 file not found

 12 bad mode qiven to open

 13 can't create file

 14 seek past 65535th record

**int setfcb(fcbaddr,filename)**

**char fcbaddr[36];**

**char \*filename;**

Инициализирует 36-байтовый блок управления файлом (буф, fcb) cp/m, расположенный по адресу fcb addr, занесением ограниченного нулем имени файла, адресуемого filename. Символы нижнего регистра переводятся в верхний регистр. Неиспользуемые позиции полей имени и расширения заполняются в fcb пробелами. В fcb обнуляется поле номера записи и поле номера экстента. Если в имени файла встречается запрещенный символ, то этот символ и остаток строки имени файла игнорируются.

**char \*fcbaddr(fd)**

Возвращает адрес внутреннего (обычно невидимого) блока управления файлом (буф, fcb), ассоциированного с открытым файлом, имеющим дескриптор fd. Возвращается -1, если fd не является дескриптором открытого файла.

**Функции буферированного обмена с файлами:**

int fopen(filename, iobuf)

char \*filename;

struct \_buf \*iobuf;

int getc(iobuf)

stuct \_buf \*iobuf;

ungetc(c,iobuf)

char c;

struct \_buf \*iobuf;

int getw(iobuf)

struct \_buf \*iobuf;

int fcreat(filename, iobuf)

char \*filename;

struct \_buf \*iobuf;

int putc(c,iobuf)

char c;

struct \_buf \*iobuf;

int putw(w, iobuf)

struct \_buf \*iobuf;

int flush(iobuf)

struct \_buf \*iobuf;

int fclose(iobuf)

struct \_buf \*iobuf;

int fprintf(iobuf,format, arg1, arg2,...)

struct \_buf \*iobuf;

char \*format;

int fscanf(iobuf,format, arg1, arg2,...)

struct \_buf \*iobuf;

char \*format;

char \*fgets(str, iobuf)

char \*str;

struct \_buf \*iobuf;

int fputs(str, iobuf)

char \*str;

struct \_buf \*iobuf;

**Функции работы с экраном:**

setplot(base, xsize, ysize)

clrplot()

plot(x,y,chr)

char chr;

txtplot(string, x, y, ropt)

char \*string;

line(c,x1,y1,x2,y2)