Отличия BDS-C от UNIX-C

Fix Me!

BDS C

Документ был подготовлен в ИНЭУМ

```
    Отличия BDS-C от UNIX-C
        ======================
   (см.Керниган,Ритчи. Язык программирования Си.)

    Замечания к Приложению "Справочное руководство по языку Си" в книге
Кернигана, Ритчи "Язык программирования Си".

А.Введение.

    Краткое перечисление отличий BDS-C от UNIX-C:

  а) Исходный файл грузится в память целиком,а не просматривается через
"окно". Это накладывает ограничения на размер исходного файла.
  б) Компиляция производится сразу в код процессора 8080,и никакой про-
межуточный результат не выводится.
  в) BDS-C компилятор написан на ассемблере 8080,а не на Си. Это сделано
для повышения эффективности компилятора.
  г) Типы  short int,long int,float и double не поддерживаются.
  д) Не существует объявления классов памяти. Классы static  и  register
не поддерживаются компилятором; все переменные принадлежат к классам
external или automatic,в зависимости от контекста,в котором они опреде-
лены.
  е) Синтаксис определений переменных сильно отличается от стандарта.
  ж) Не поддерживается начальная инициализация переменных.
  з) Распределение памяти под строки должно производиться явно. Не про-
изводится автоматического управления памятью.

------------------------------------------------------------------------
Б.Замечания к Приложению.

    Номера главок в дальнейшем изложении соответствуют номерам главок в
книге Кернигана,Ритчи.

1. Введение

       BDS-C написан для микрокомпьютеров с процессором 8080 (или совмес-
тимых),с операционной системой CP/M (или совместимой) и генерирует код
микропроцессора 8080.

2.1. Комментарии.

    Комментарии являются вложенными по умолчанию; чтобы вернуться к
стандарту, укажите опцию -c в команде CC при компиляции.

2.2. Идентификаторы(имена).

    В именах переменых малые и большие буквы различаются, а в именах
функций считаются идентичными. Поэтому рекомендуется писать имена фун-
кций все время в одном виде,чтобы избежать путаницы. Например,объявление

                    char foo,Foo,FoO;

   определяет три разных переменных, но два выражения

                    printf("This is a test");
```

и

```
                    prINTf("This is a test");
```

эквивалентны.

2.3. Служебные слова.

Ключевые слова BDS-C:

```
int                     else
char                    for
struct                  do
union                   while
unsigned                switch
goto                    case
return                  default
break                   sizeof
continue                begin
if                      end
register                void
```

Большие и малые буквы не различаются в служебных словах,т.е. слова WHILE и while одинаковы.
Идентификаторы с теми же именами,что и ключевые слова,не допускаются,хотя те могут входить как часть в имена(например, charflag)
Вместо скобок { и } в программах допускается использование слов begin и end.

---

4. Что такое имя?

В BDS-C существует только два класса памяти,external и automatic, но они не объявляются явно. Контекст,в котором встречаются переменные,позволяет однозначно определить,к какому классу они принадлежат: переменные,описание которых встретилось внутри какой-либо функции,считаются automatic,а все другие - external.
Область действия автоматических переменных распространяется на ту функцию,в которой они описаны. Имена переменных,объявленных в одном модуле,не должны совпадать между собой. Это означает,что не должно быть элементов записей и объединений (struct и union),совпадающих по именам с другими переменными.(см.исключения в п.11.1.)
В BDS-C нет концепции блоков внутри функций. Хотя переменные и могут описываться внутри составных операторов,но их имена не должны совпадать с именами уже существующих переменных,и область действия данной переменной распространяется до конца функции,в которой описана эта переменная.
Если программа состоит из нескольких файлов и существуют общие внешние переменные,то необходимо завести файл с расширением .H и описать в нем все внешние переменные,а затем командой препроцессора #include использовать в каждом модуле,где употребляются общие внешние переменные.

6.1. Символы и целые числа.

Распространение знака при переходе от char к int не происходит. Все символьные переменные интерпретируются как положительные числа от 0 до 255. Будьте осторожны поэтому,проверяя результат,возвращаемый некоторыми функциями!
Большинство арифметических операций преобразуют символьные переменные к целому виду. В некоторых неарифметических операциях BDS-C оптимизирует код при работе с char-переменными. Поэтому рекомендуется объявлять все переменные,значения которых лежат в промежутке 0-255,как char.

---

7. Выражения.

Деление на 0 и взятие остатка от деления на 0 выдают в качестве результата 0;никакой ошибки не вырабатывается.

7.1. Первичные выражения.

Порядок вычисления аргументов функции - обратный(т.е. последний аргумент вычисляется первым,а первый - последним). Это сделано для функций с переменным числом параметров.

7.2. Унарные операции.

Операции

(<имя типа>) <выражение>
sizeof (<имя типа>)

не поддерживаются. Оператор sizeof может использоваться в виде

sizeof <выражение>

причем <выражение> не должно быть массивом. Для получения размера массива можно объявить массив структурой (struct) и взять sizeof от структуры. Другой способ получения размера состоит в вычислении размера элемента массива и умножении его на количество элементов.

7.5. Операции сдвига.

Операция >> всегда логическая (заполнение нулями).

7.11, 7.12. Логические операции И и ИЛИ.

Операции && и || имеют одинаковый приоритет в BDS-C. Рекомендуется использовать скобки в выражениях с логическими операциями.

---

8. Описания.

<описание> ::= <спецификация типа> <список описаний>

8.1. Спецификация класса памяти.

Не поддерживается.

8.2. Спецификация типа.

```
<спецификация типа> ::=  char           |
                          int            |
                          unsigned       |
                          register       |
             < спецификация записи или смеси >
```

Тип register используется как синоним имени int. Ключевое слово void используется как синоним типа int,для обозначения того факта,что функция не возвращает осмысленного значения.

8.3. Описатели.

Не поддерживается инициализация. Синтаксис описателя:

```
<список описателей> ::= <описатель>   |
              <описатель>,<список описателей>
```

8.4. Смысл описателей.

UNIX-C допускает весьма сложные комбинации в описании типов,например:

```
struct foo * ( * ( *bar[3][3][3] ) () ) ();
```

которое объявляет bar как массив 3x3x3 ссылок на функции,возвращающих ссылки на функции,которые возвращают ссылки на записи типа foo. К сожалению,BDS-C не допускает подобных описаний.

Синтаксис описателя:

```
<простой тип> ::= char       |
                  int        |
                  unsigned   |
                  struct     |
                  union

<скалярный тип> ::= <простой тип>   |
          * <скалярный тип> |
          * <имя функции> ()
```

Внимание! Указатели на указатели на функции не поддерживаются в BDS-C.

Примеры описателей скалярного типа:

```
int x,y;
char *x;
unsigned *fraz;
char **argv;
struct foobar *zot,bar;
int *(*ihtfp)();        ( Описывает ihtfp как ссылку на функцию,
            возвращающую ссылку на целое)
```

Определим массив как одно- или двумерное объединение объектов скалярного типа (включая ссылки на функции). Теперь мы имеем конструкции вида:

```
char *x[5][10];
int **foo[10];
struct steph bar[20][8];
union joyce *ohboy[747];
int * (foobar[10]) ();  ( Описывает foobar как массив из десяти
         ссылок на функции,возвращающие целые)
```

Функции должны возвращать результат любого скалярного типа,за исключением ссылок на функции,записей и смесей.

Несколько примеров:

```
char *bar();            Функция,возвращающая ссылку на символ
char *( *bar)();        Ссылка на функцию,
            возвращающую ссылку на символ
char *( *bar[3][2])();  Массив 3x2,содержащий ссылки на функции,
            возвращающие ссылку на символ
struct foo *zot();      Функция,возвращающая ссылку на запись
```

Внимание! Из-за "простоты" компилятора не допускается описание ссылок на массивы. Например,описание:

```
char ( *foo )[5];       Ссылка на массив символов
```

будет эквивалентно описанию

```
char *foo[5];           Массив ссылок на символ
```

8.5. Описание записей и смесей.

Битовые поля не поддерживаются; поэтому синтаксис описателя следующий:

```
 <спецификатор записи или смеси> ::=
    <запись или смесь> { <список описаний записи> }          |
    <идентификатор записи или смеси> { <список описаний записи> }  |
    <идентификатор записи или смеси>

 <запись или смесь> ::=  struct  |  union

 <список описаний записи> ::=
           <описание записи>                    |
           <описание записи> <список описаний записи>

-------------------------------------------------------------------------
 <описание записи> ::= <спецификатор типа> <список описателей записи>;

 <список описателей записи> ::= <описатель>                |
                  <описатель>,<список описателей записи>
```

Имена элементов и полей записей и смесей не должны совпадать с другими локальными переменными (об исключениях см. 11.1).

8.6. Инициализация.

К сожалению,инициаторы не поддерживаются. Все внешние переменные инициируются нулями перед выполнением.

8.7 , 8.8. Имена типов.

Не поддерживаются в BDS-C. Ключевое слово typedef не задействовано.

9.2. Блоки.

Понятие "блок" в BDS-C не определено. Переменные не могут быть объявлены как локальные в данном блоке; объявление,появляющееся где-либо в функции,действует до ее конца.

9.6. Оператор for.

В книге Кернигана,Ритчи содержится неточность: оператор for не полностью эквивалентен оператору while,как показывается,в следующем смысле: если при вычислении <оператора> встречается оператор continue, то управление передается не <выражение-3>. В представлении через while управление передается на <выражение-2>,без приращения.

Это просто неточность документации: и UNIX-C,и BDS-C обрабатывают оператор for правильно.

9.7. Оператор switch.

На одну конструкцию switch не может быть больше 200 вариантов case. Многократные варианты не считаются за один; так,в операторе

```
        case 'a': case 'b': case 'c': printf("a or b or c");
```

три варианта case.

9.12. Помеченный оператор.

Метки сразу за вариантами case или default не поддерживаются. Метка может появляться до ключевого слова case или default. Например,оператор

```
        case 'x': mumble: zap=frotz;
```

неверен,и должен быть преобразован в

```
        mumble: case 'x': zap=frotz;
```

10. Внешние определения.

Спецификаторы типа должны указываться явно для всех объектов,кроме функций: для них тип по умолчанию есть int.

11.1. Лексические области действия.

Элементы и поля записей и смесей не долны иметь имена,совпадающие с именами других локальных в данном модуле объектов. Таким образом, нельзя объявить подобный объект:

```
        struct foo {       /* определяет запись типа foo */
           int a;
           char b;
        } foo[10];
         /* определяет массив foo из десяти записей типа foo */
```

хотя UNIX-C допускает подобные вещи.
Одно исключение из этого правила делается для элементов записей,имеющих: а) одинаковый тип и б) одинаковое смещение с другим элементом другой записи или смеси. Например:

```
        struct foo {
           int a;
           char b;
           char *cptr;          /* тип: *char, смещение: 3 */
        };

        struct bar {
           unsigned aa;
           char xyz;
           char *cptr;          /* тип: *char, смещение: 3 */
        };
```

11.2. Область действия внешних имен.

Ключевого слова extern не существует; все внешние переменные должны быть определены точно в том же порядке,что и в других файлах. Кроме того,все внешние переменные,используемые в данной программе,должны быть описаны в файле,содержащем функцию main. Вообще говоря,реализация внешних переменных в BDS-C напоминает непомеченный общий блок в Fortran'е.
Кстати: если вы используете библиотечные функции alloc и free,включите файл BDSCIO.H в программу; в нем находятся описания некоторых внешних переменных.

12.2. Включение файлов.

Если имя файла в команде #include стоит в кавычках,то файл ищется не только в текущей директории,но и,в случае его отсутствия там,на диске по умолчанию, и в случае неудачи программа прерывается. Если же имя файла указано в угловых скобках < и >,то поиск производится только на текущем диске.
Запомните,что включение файлов производится вне зависимости от места,в котором появляется команда #include. Даже если она встретится внутри блока условной компиляции,файл вставляется в программу и в том случае,если условие не выполнено.
Вложение файлов допустимо на любую глубину.

12.3. Условная трансляция.

Выражение <константное выражение> ограничивается только следующими случаями:

```
<выражение> ::= <выражение2>           |
          <выражение2> && <выражение>  |
          <выражение2> || <выражение>

<выражение2> ::= <десятичная константа>     |
          ! <выражение2>          |
          ( <выражение> )
```

<десятичная константа> воспринимается транслятором как логическое выражение,т.е. значение 0 представляет собой FALSE,остальные - TRUE.
Вложения операторов условной трансляции не полностью поддерживается компилятором.

12.4. Управление строками.

Не подерживается.

15. Константные выражения.

BDS-C упрощает константные выражения во время трансляции,но только в некоторых случаях: когда оно следует за
а) левой квадратной скобкой,
б) ключевым словом case,
в) оператором присваивания,
г) запятой,
д) левой скобкой,
е) ключевым словом return.
В остальных случаях гарантируется,что выражение вычисляться не будет.
Обычно,чтобы гарантировать упрощение константных выражений,их заключают в скобки.Например,выражение

```
x = x + y + 15 * 10;
```

не будет упрощаться (т.е.,это заставит компилятор вставить в программу операцию умножения 10 на 15). Если же переписать его так:

```
x = x + y + (15 * 10);
```

то это приведет к вычислению 10*15 во время компиляции,и тем самым к упрощению кода.

18.1. Выражения.

Унарные операции:

```
* & - ! ~ ++ -- sizeof
```

Бинарные операторы && и || имеют одинаковый приоритет.
Оператор sizeof не может считать размер массива.

18.2. Описания.

Полный синтаксис описания:

```
<описание> ::= <спецификатор типа> <список описателей> ;
<спецификатор типа> ::= char                              |
                        int                                   |
```

```
                              register (то же,что и int)       |
                              unsigned                           |
            <спецификация записи или смеси>
   <список описателей> ::= <описатель>  |
            <описатель> , <список описателей>
   <описатель> ::= <идентификатор>              |
          ( <описатель> )             |
          * <описатель>               |
          <описатель> ()              |
          <описатель> [ <константное выражение> ]
   <спецификация записи или смеси> ::=
                      struct { <список описателей> }         |
                      struct <идентификатор> { <список описателей> } |
                      struct <идентификатор>              |
                      union  { <список описателей> }         |
                      union  <идентификатор> { <список описателей> } |
                      union  <идентификатор>
```

18.4. Внешние определения.

```
   <определение данных> ::= <спецификатор типа> <список описаний> ;
```

18.5. Препроцессор.

Поддерживаются следующие директивы:

```
   #define <идентификатор> <подстановка>
   #include " <имя файла> "
   #include < <имя файла> >  (внешние скобки-элемент языка)
   #if <выражение>
   #ifdef <идентификатор>
   #ifndef <идентификатор>
   #else
   #endif
   #undef <идентификатор>
```

Оператор `#if` поддерживается,но допустимые <выражения> ограничены.
Оператор `#include` не должен появляться внутри блоков условной трансляции.

Элементы компилятора языка Си.

1. Пример типичной компиляции.

Рассмотрим как пример компиляцию файла `progr.c`,содержащего программу на Си.
Компилятор запускается командой:

```
A>cc progr.c
```

После печати сигнального сообщения `CC` считывает файл `progr.c` с диска и обрабатывает его. Если в программе нет ошибок,`CC` выдает диагностику использования памяти и загружает `CC2.COM`. `CC2` продолжает обработку программы и,если не находит ошибок,записывает файл `progr.crl` на диск.
Следующий шаг - вызов линкера (редактора связей):

```
A>clink foo
```

Если в файле нет неразрешенных внешних ссылок,то генерируется файл `progr.com`,который может быть запущен на исполнение командой

```
A>progr [ аргументы,если есть ]
```

2. Описание элементов компилятора.

2.1. CC - анализатор.

Формат команды:
CC [<имя дисковода>:]<имя>.<расширение> [<опции>]

Допустимо любое имя и любое расширение. По умолчанию расширение считается .C . CC сначала пытается найти указанный файл; если это не удается сделать,и не указано расширение в имени,то компилятор добавляет к имени файла расширение .C и еще раз пытается найти такой файл.

Если указывается имя дисковода,то исходный файл будет считываться с указанного дисковода,и вся выдча компилятора будет помещена на тот же дисковод.

Нажатие Ctrl-C в любой момент прерывает компиляцию.

Вслед за именем файла в командной строке можно указать опции,управляющие компилятором:

-p Вызывает печать текста программы без комментариев,со включенными по директиве #include файлами и произведенными подстановками команд #define. Опция используется для проверки соблюдения вложенности комментариев. Во время выдачи можно нажать Ctrl-P,что вызовет выдачу на устройство LST (принтер).

-a <d> Указывает дисковод,с которого надо загружать CC2.COM. Если в качестве имени дисковода указана буква z,то файл CC2.COM не загружается и промежуточный результат сбрасывается на диск для последующей отдельной компиляции в файл с именем исходного файла и расширением .CCI .

-d <x> Указывает дисковод,на который надо выдавать .CRL-файл. Если используется совместно с опцией -a z ,то <x> указывает дисковод для промежуточного файла с расширением .CCI .

-m xxxx Определяет адрес (шестнадцатиричный) начала библиотеки времени выполнения,т.е. адрес начала .COM-файла (для нестандартных применений). Если Вы используете эту опцию,перекомпилируйте пакет CCC.ASM и используйте опции -l,-e и -t при вызове CLINK. Необходимо,чтобы сразу после CC автоматически загрузился CC2.COM.

-e xxxx Определяет адрес (шестнадцатиричный) начала области внешних данных. Обычно внешние данные располагаются сразу за программой,но Вы можете изменить это. Использование опции -e позволяет компилятору вырабатывать более оптимальный код. Учтите: не рекомендуется использовать эту опцию в неотлаженной программе! Чтобы узнать адрес конца программы,можно поступить следующим образом: скомпилировать программу с опцией -e,оставляя адрес данных на старом месте,затем определить "Last code address" по статистике ,выдаваемой CLINK, и только потом перекомпилировать программу,указывая в опции -e новый адрес. Не располагайте данные и программу очень близко - если вы будете производить модификации программы,то код и данные могут налезть друг на друга. Необходимо,чтобы сразу после CC автоматически загрузился CC2.COM.

-o Указывает,что надо производить код,оптимальный по времени выполнения. Если действует опция -o,то многие стандартные подпрограммы из библиотеки времени выполнения вставляются прямо на место их вызова. Это приводит к более быстрому,но и более длинному коду. Для быстрейшего кода используйте опции -e и -o вместе; для кратчайшего кода - опцию -e без -o. Необходимо,чтобы сразу после CC автоматически загрузился CC2.COM.

-x Требует прекращения .BAT-файла в случае ошибок компиляции.

-r x Отводит x килобайт памяти под таблицу символов. Если в процессе компиляции возникает ошибка "Out of symbol table space",попробуйте перекомпилировать файл,увеличив объем таблицы символов опцией -r. Если же Вы встретите ошибку "Out of memory",попытайтесь уменьшить объем таблицы сиволов. Размер таблицы по умолчанию - 10Кбайт.

-c Запрещает вложенность комментариев в программе (по стандарту UNIX-C). В этом случае конструкция
```
/* printf("hello");     /* this prints hello */
```
считается законченым комментарием. Если опция -c не используется,компилятор будет ждать другую "скобку" */ ,чтобы закончить комментарий.

Один файл на языке Си не может содержать более 63 определений функций, однако,поскольку программа на Си может находиться в нескольких файлах,то это не оказывает заметного влияния на размер программы.
Если обнаружена ошибка,то процесс компиляции прерывается немедленно,без выдачи промежуточного результата или вызова CC2.

Скорость обработки: около 20 строк/секунда. После загрузки исходного файла диск не требуется.

Примеры:

```
A>cc foobar.c -r12 -ab
```

запускает на компиляцию файл foobar.c,отведя под таблицу символов 12K байт. CC2.COM автоматически загружается с дисковода B:

```
A>cc c:belle.c -p -o
```

запускает на компиляцию файл foobar.c с диска C:. Текст печатается на терминале без комментариев,со включенными по директиве #include файлами и произведенными подстановками команд #define. Код оптимизируется по скорости выполнения.

2.1. CC2 - генератор кода.

Формат команды:
CC2 [<имя дисковода>:]<имя>

Обычно CC2 загружается автоматически вслед за CC. Если загружать его явно,то с диска будет считываться файл <имя>.CCI,содержащий промежуный результат компиляции.
Если не обнаружено ошибок,на диске создается файл <имя>.CRL .
Скорость обработки: около 70 строк/секунда.
Нажатие Ctrl-C в любой момент прерывает компиляцию.

Пример:

```
A>cc2 foobar
```

2.3. CLINK - редактор связей для Си.

Формат команды:
CLINK [<имя дисковода>:]<имя> [<другие имена и опции>]

Файл <имя>.CRL должен содержать функцию main; остальные .CRL-файлы (до опции -f) будут целиком подсоединяться к главному файлу. Если в командной строке появилась опция -f,то все .CRL-файлы,перечисленные за ней,будут просматриваться в оисках нужных функций и подсоединяться будут только нужные функции из этих файлов. После обработки всех указанных .CRL-файлов,начинают обрабатываться библиотеки DEFF*.CRL - в них ищутся

необходимые функции,не найденные в предыдущих файлах. Последовательность поиска в библиотеках: DEFF.CRL,DEFF2.CRL и,если создан пользователем, DEFF3.CRL.

По умолчанию,CLINK ищет все указанные файлы на текущем диске,а все библиотечные файлы - на "диске по умолчанию". Если у главного файла указан префикс <имя дисковода>,то поиск указанных в командной строке файлов ведется на указанном дисководе. Каждый дополнительный .CRL-файл может иметь указатель <имя дисковода> для определения,с какого диска его загружать. Если файл не найден на указанном диске,поиск продолжается на "диске по умолчанию".

Если после обработки всех файлов остались неразрешенные внешние ссылки,то CLINK переходит в интерактивный режим и в диалоге с пользователем определяет другие .CRL-файлы с нужными функциями.

Нажатие Ctrl-C в любой момент прерывает редактирование и возвращает в систему.

В списке файлов могут встречаться опции редактора,перед которыми стоит тире (-). Несколько однобуквенных опций можно объединять,ставя перед этой комбинацией одно тире. Допустимые опции:

-s Печатает карту памяти и статистику на консоли.

-f Требует просматривать следующие за ней файлы в поисках нужных функций; файлы,указанные в командной строке после -f,загружаются в память не полностью,а из них выбираются только требуемые функции.

-e xxxx Устанавливает адрес начала области внешних данных на (шестнадцатиричный) адрес xxxx. Обычно внешние данные начинаются сразу за кодом программы,но данная опция позволяет изменить это положение. Возможность используется в случае организации "цепей" (вызовов функций exec,execl,execv) и оверлейных структур,чтобы быть уверенным,что данные находятся на том же месте,что и вызывающей программе. Чтобы указать адрес,воспользуйтесь следующей процедурой: сначала запустите команду CLINK с данными файлами,но без опции -e,чтобы узнать максимальный адрес начала данных среди всех файлов (печатается после фразы "Data starts at:"), а затем укажите его в опции -e. Если файл,содержащий функцию main,компилировался с опцией -e,то CLINK автоматически знает адрес начала области данных,но если один из файлов,указанных в командной строке,компилировался без опции -e или адрес при компиляции отличался от адреса при редактировании связей,то полученный .COM-файл будет работать неверно. Файл,откомпилированный без опции -e,можно использовать в такой ситуации,только указав адрес в опции -e,совпадающий с адресом,указанным при компиляции главного файла.

-z Запрещает инициализацию области внешних данных нулями перед выполнением. Если указана опция -z,то все внешние переменные принимают произвольные значения ("мусор"). В противном случае все переменные обнуляются.

-t xxxx Устанавливает начало резервируемой памяти на адрес xxxx (шестнадцатиричное). На этом месте будет размещаться стек при выполнении программы. Отводимое пространство должно быть достаточно большим для размещения программы и внешних данных до данного адреса.

-o name Переименовывает выходной файл в name.COM. Если перед именем файла стоит идентификатор диска,то вывод производится на указанное устройство. По умолчанию выод производится на текущее устройство в файл с именем первого из указанных в командной строке файлов и расширением .COM.

-n Получающийся .COM-файл не портит системную область в памяти.

Это сокращает объем доступной памяти на 2Кбайт,но позволяет завершать работу программы без подгрузки системы с диска. Эту возможность рекомендуется использовать для небольших программ,которые не требуют всей памяти машины и достаточно часто используются. Опция -n игнорируется,если вместе с ней используется опция -t.

-w Записывает в файл на диске таблицу имен и абсолютные адреса всех функций,включенных в результирующий файл. Эта возможность используется для организации оверлейных сегментов. Файл получает имя результирующего .COM-файла и расширение .SYM.

-y name Считывает с диска файл name.SYM и использует адреса,указанные в нем, в текущем редактировании. Опции -y и -w используются совместно для создания оверлеев следующим образом: когда редактируется корневой сегмент (та часть программы,которая загружается сначала),необходимо указать опцию -w для создания таблицы функций корневого сегмента. Затем,при редактировании оверлейных сегмен тов,указывается опция -y для считывания имен глобальных функций и предотвращения присоединения библиотеки времени выполнения. Эта процедура может быть повторена: при редактировании оверлейного сегмента можно указать опцию -w,чтобы перечислить новые функции для оверлейного модуля второго уровня. Это вложение может продолжаться достаточно много раз. Помните,что важно положение опции -y в командной строке: символьный файл,определенный данной опцией,будет просматриваться только после присоединения всех .CRL-файлов,указанных в командной строке до нее. Поэтому во избежание неприятностей указывайте данную опцию сразу после имени главного файла. Если при считывании .SYM-файла обнаруживается имя функции,уже встретившейся в процессе редактирования,то об этом будет выдано соответствующее сообщение и старое значение функции сохранится. Для более подробной информации об оверлеях см. соответствующую главу.

-l xxxx Изменяет адрес загрузки .COM-файла на xxxx(шестнадцатиричное). Эта опция используется при создании оверлейных модулей (совместно с опцией -v) или для программ,которые будут работать на нестандартном оборудовании. В последнем случае необходим специально подготовленный файл C.CCC. Кроме этого,надо указать опции -t и -e для определения соответствующих областей памяти.

-v Обозначает,что создается оверлейный сегмент. При этом не присоединяется библиотека времени выполнения. Считается,что она уже есть в корневом сегменте.

-c d Требует продолжить поиск ненайденных .CRL-файлов и библиотек (DEFF.CRL,DEFF2.CRL,DEFF3.CRL,C.CCC) на диске d.

-d "arg" Для быстрой проверки запускает сгенерированный файл после записи на диск. Если указаны аргументы в кавычках,то это аналогично вызову данного файла с перечисленными аргументами. Опция -d не должна указываться при редактировании оверлейных сегментов. Опция -d игнорируется,если она используется совместно с опцией -n.

-r xxxx Отводит xxxx (шестнадцатиричное) байт под таблицу ссылок вперед. По умолчанию отводится 600h байт. Может использоваться при сообщении редактора "Ref table overflow".

Примеры:

```
A>clink ted -s -t 6000 -o jouce
```

```
     CLINK связывает файл TED.CRL,содержащий функцию main,с необходимыми
функциями из библиотек  DEFF,DEFF2,DEFF3.  После завершения печатается
статистика,стек устанавливается на адрес 6000h и растет вниз. Получив-
шийся .COM-файл называется JOUCE.COM.

A>clink b:lois c:vicky -f janet -s

     CLINK связывает файл LOIS.CRL,находящийся на диске b: и содержащий
функцию main,а также файл VICKY.CRL с диска c: с необходимыми функциями
из библиотек DEFF,DEFF2,DEFF3,также находящимися на диске b:.  Из файла
JANET считываются только необходимые функции,не найденные в предыдущих
файлах. После завершения печатается статистика.

   Помните о необходимости перечислять все используемые в программе
внешние переменные в файле,содержащем функцию main !
```

Ссылки

http://sysadminmosaic.ru/bds_c/bds_c/differences_from_unix-c

2022-11-21 20:55