Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №14

Препроцессор языка Си

Препроцессор языка Си – это составная часть компилятора, реализующая первую стадию компиляции.

Препроцессор - это мощный инструмент, часто используемый для повышения удобочитаемости, надежности и переносимости программ.

Обработка препроцессором выполняется на первом этапе компиляции исходного текста программы.

Препроцессор читает исходный текст, отыскивая и обрабатывая *директивы препроцессора*, которые представляют собой строки текста, начинающиеся с символа **#**.

Пример 1

```
#include <stdio.h>
#define SIXE 100
```

Для переносимости программы следует указывать символ **#** в позиции 1, а непосредственно после него – управляющее слово (**include**, **define** , и т.п.), поскольку некоторые препроцессоры языка K&R Си не допускают, чтобы символы промежутков предшествовали символу **#** или следовали непосредственно после него.

Препроцессор позволяет определять константы и макросы, включать файлы, предоставляет возможности условной компиляции.

Поименованные константы

Поименованные константы определяются с помощью *директивы препроцессора define*:

```
#define идентификатор строка-шаблон
```

Поименованные константы используются для того, чтобы облегчить чтение и изменение программ.

До и после идентификатора в директиве **define** должен быть один или более пробелов. Строка-шаблон начинается с первого отличного от пробела символа и заканчивается символом перевода строки. Строку-шаблон можно продолжить более чем на одну строку, начиная новые строки символом ****.

После определения поименованной константы препроцессор заменяет в исходном тексте программы идентификатор значением строки-шаблона. Внутри комментариев и строковых констант такая замена не производится.

В отличии от операторов языка Си, директивы препроцессора Си не заканчиваются символом `;`.

Пример 1

```
#define TRUE    1
#define FALSE   0
#define MAXITEMS 500
int main()
{
  int i, found, val[MAXITEMS];
  found=FALSE;
  while (found==FALSE)
    {
     ...
     if (выражение)
         found=TRUE;
    }
  ...
  for (i=0; i<MAXITEMS; i++)
  /*  прочитать значение val [i] */
  ...
}
```

Использование в прим. 1 поименованных констант **`TRUE`** и **`FALSE`** делает текст программы более понятным. Константа **`MAXITEMS`** упрощает сопровождение программы. К примеру, если потребуется изменить количество предметов, придется исправить только одну директиву препроцессора и перекомпилировать программу.

При обработке препроцессором поименованные константы **`TRUE`**, **`FALSE`** и **`MAXITEMS`** заменяются, соответственно, на значения 0, 1 и 500. При этом директивы препроцессора удаляются из исходного текста программы и не попадают на следующий этап компиляции.

Примечание 1

Использование заглавных букв в идентификаторах поименованных констант является программистским соглашением.

Макросы

Макрос — это короткая процедура, у которой могут быть аргументы.

Макрос определяется с помощью директивы препроцессора #define.

Определение макроса похоже на определение поименованной константы.

`#define идентификатор(аргумент[,аргумент]...) строка-шаблон`

В скобках непосредственно после идентификатора могут быть указаны имена аргументов. Между идентификатором и символом **(** не должно быть символов промежутка. При расширении макроса препроцессор осуществляет вставку строк. Препроцессор ничего не знает о синтаксисе языка Си или логике программы. Поэтому такие подстановки могут быть источниками синтаксических ошибок.

Пример 1

```
#define SQUARE(X) ((X)*(X))
#define MAX(A,B) ((A)>(B)?(A):(B))
int main()
{
  int int1, int2;
  int1=SQUARE(3);
  int2=SQUARE(int1+1);
  printf("max = %d\n", MAX(int1,int2));
}
```

Результат препроцессирования:

```
int main()
{
  int int1, int2;
  int1 = ((3)*(3));
  int2 = ((int1+1)*(int1+1));
  printf ("max = %d\n", ((int1)>(int2)?(int1):(int2)));
}
```

Вывод программы:

```
max = 100
```

Если в прим. 1 в определении **SQUARE** не будут использованы скобки, то переменной **int2** будет присвоено другое значение:

```
int2 = int1 + 1 * int1 + 1;
```

Для просмотра результата препроцессирования надо использовать команду компиляции **cc** с опцией **-E**.

Макрос может использовать другие ранее определенные макросы. Многие препроцессоры Си, включая препроцессор системы ОС UNIX, не допускают использования рекурсивных макросов.

Иногда удобно определять макрос так, чтобы он был блоком операторов.

Одно из преимуществ блока — возможность определить локальные переменные. Если только макрос содержит более одного оператора, эти операторы должны быть включены в блок, иначе возможны неприятности.

Пример 2

```
/* Эти операторы следует заключить в блок */
#define DOIT(A, B, C) A=D*2; \
                      B=C*2;
void f(void)
{
  int i, j, k;
  if (i<j)
     DOIT(i, j, k) /* Здесь после обработки препроцессором будет синтаксическая ошибка */
  else
    printf("Something \n");
}
```

Хотя этот фрагмент программы и выглядит корректно, однако после расширения препроцессором компилятор обнаружит ошибку. Синтаксис оператора if-else требует единственного оператора между **`if`** и **`else`**. В такой ситуации использование блока в макросе устранит проблему.

Стандарт ANSI языка Си запрещает макроподстановки в строковых литералах.

Операция

Операция # - операция для создания строки из аргумента макроса.

Пример 3

Определение макроса:
#define string(x) #x
Использование макроса:
string(hello)
Результат подстановки:
"hello"

Пример 4

```
#define SWAP(A,B) {int temp; \
                   temp = A;\
                  A = B;\
                  B =temp; \
                  }
#define PRINT(A,B)   printf(#A": %d, "#B": %d\n", A, B)
int main( )
{
  int num1 = 30, num2 = 90;
  PRINT(num1, num2);
  if (num2 > num1)
```

```
    SWAP(num1, num2);
  PRINT(num1, num2);
}
```

Вывод программы:

```
num1: 30, num2: 90
num1: 30, num2: 90
```

Операция

Операция ## — операция для объединения лексем в определении макросов в одну лексему.

Пример 5

```
#define WheatBread 0
#define RyeBread 1
#define WhiteBread 2
#define PumpernickelBread 3
#define Bread(x) x ## Bread
int main ()
{
   printf("Value of WheatBread is %d\n", Bread(Wheat));
   printf("Value of RyeBread is %d\n", Bread(Rye));
   printf("Value of WhiteBread is %d\n", Bread(White));
}
```

Вывод:

```
Value of WheatBread is 0
Value of RyeBread is 1
Value of WhiteBread is 2
```

Вызов макроса **`Bread(Rye)`** расширяется сначала в **`Rye##Bread`**, потом – в одно слово **`RyeBread`**, заменяемое затем константой **`1`**.

Сравнение функций и макросов

Большим преимуществом макросов является то, что они быстрее функций. Потери времени при вызове функции включают в себя время на сохранение аргументов и регистров на стеке и восстановление их при возврате. При вызове макроса этого не происходит.

Однако, если макрос используется много раз, исполняемая программа будет иметь больший размер, чем при использовании функций.

Преимуществом функций является возможность возвратить результат при помощи оператора return. В макросе оператор **`return`** вызовет немедленный выход из функции, в которой вызван макрос.

Кроме того, функции могут быть рекурсивными. Большая же часть препроцессоров Си (в том числе и в операционной системе UNIX) запрещают рекурсивные макросы.

К тому же макросы сложнее для отладки, чем функции. Отлаживая макрос, надо смотреть на результат работы препроцессора. Для этого в операционной системе UNIX можно воспользоваться следующими командами:

```
$cc -E -o proc.i proc.c
$cat proc.i
```

Организация файлов программы

Файлы заголовков

Директива препроцессора include влечет вставку в текущую позицию текста программы на языке Си копии указанного файла.

```
#include  <file.h>
```

```
#include "file.h"
```

Подобные файлы, называемые *файлами заголовков* или файлами вставок, обычно содержат директивы define для поименованных констант и макросов, объявления функций, внешних переменных и т.п., необходимые для нескольких исходных файлов. Преимущество подобного выделения в отдельный файл в том, что такие определения локализуются в общем месте. Несколько программ могут включать этот файл и быть уверенными в использовании одинаковой информации. Если необходимо изменение, его можно сделать в одном месте.

По соглашению, имена файлов-вставок имеют расширение **.h** , что означает файл заголовков (header), названных так потому, что директивы **include** обычно указываются в начале исходных файлов языка Си. Если имя файла заключено в угловые скобки **<** и **>**, препроцессор ищет файл в "стандартном месте". В операционных системах UNIX таким стандартным местом является каталог **/usr/include**. Если имя файла заключено в двойные кавычки **"**, используется стандартное соглашение операционной системы для доступа к файлу. В операционных системах UNIX может использоваться полное или относительное путевое имя.

Пример 1

```
#include "proj.h"
#include "/local/include/x33.h"
#include "../include/defs.h"
```

Файлы вставки могут быть вложены; файл заголовков может содержать директивы **include**.

Образец файла заголовков

```
#define TABLTSIZE 1000
#define SYSNAMELEN 20
#define MASK 010
#define CLEARLINE() while (getchar () != '\n');
#define MAX(A,B) (A>B ? A : B)
extern int status;
extern char sysname[];
extern double table[];
extern void print_err(int);
extern double std_dev(double);
typedef char BYTE;
```

В файле заголовков не должно быть ни определений функций (программ этих функций), ни определений внешних переменных, по которым выделяется память (например, **`int x`**). Иначе, если в нескольких файлах используется один и тот же файл заголовков, возникнут неприятности. При связывании этих файлов редактор связей может сообщить о "многократном определении идентификаторов".

В файле заголовков могут содержаться и другие директивы препроцессора и языка Си, например, операторы typedef, шаблоны структур и объединений (**`struct`**, **`union`**) и операторы **`enum`**.

Организация файлов программы

На рис. 1 приводится типичный способ организации файлов, составляющих программу. Файл заголовков **`projX.h`** содержит директивы препроцессора **`define`**, объявления внешних переменных и оператор переопределения типа **`typedef`**, используемые в файлах с расширением **`.c`**.

Файл **`defs.c`** определяет только внешние переменные, используемые несколькими файлами. Функции не должны включаться в этот файл.

Файл, содержащий функцию **`main()`**, часто называют **`main.c`** для простоты идентификации.

Функции, определенные в файлах **`calc.c`** и **`bufct1.c`** взаимосвязаны и являются составными частями программы.

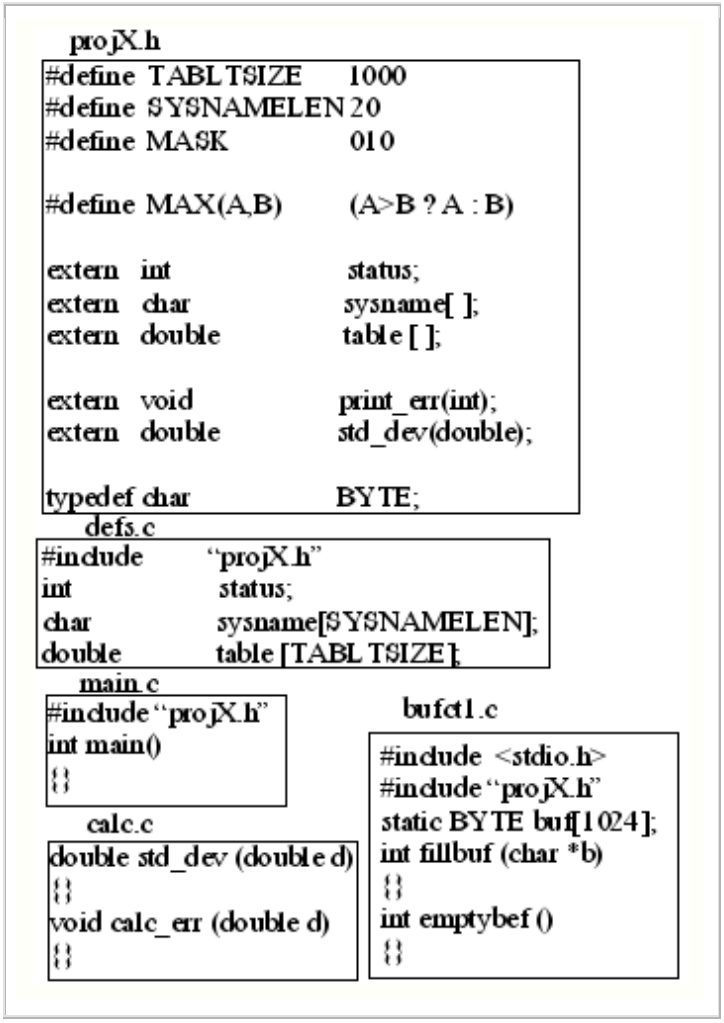

Рис. 1. Пример организации файлов программы

Условная компиляция

Условная компиляция — это средство препроцессора, которое в зависимости от условия включает в программу операторы языка Си или препроцессора.

Условная компиляция позволяет строить гибкие программы на языке Си. Это средство особенно часто используется при разработке программ, которые должны переноситься на различные процессоры, в различные операционные системы и в прочие операционные среды.

Условная компиляция реализуется с помощью следующих директив препроцессора:

- **#if**
- **#else**
- **#elif** (аналогично **else if**)
- **#endif**
- **#ifdef**
- **#ifndef**

В основном, условная компиляция выполняется следующим образом:

если это-и-это истинно

вставить указанные здесь операторы в исполнимую программу

Можно задать и ветвь **иначе**.

Директивы **#if** используются для вставки строк в файл, в случае, если истинно некоторое константное выражение.

Директивы **#ifdef** используются для вставки строк, если определена указанная поименованная константа.

Для вставки строк при условии, что константа не имеет определения, вместо директивы **#ifdef** можно использовать директиву **#ifndef**.

Большинство препроцессоров Си включает предопределенную константу, имя которой указывает на используемый компьютер, со значением равным 1 (истина). Это надо проверять в документации используемого препроцессора.

Пример 1

```
#include "local.h"
#if vax||u3b
#define MAGIC 330
#elif u3b5
#define MAGIC 430
#elif u3b2
#define MAGIC 530
#else
#define MAGIC 500
#endif
```

Чтобы изменить значение используемой поименованной константы, следует применять директиву **#undef** перед изменяющей значение директивой #define (этого требуют многие препроцессоры). Не является ошибкой применение **#undef** к еще неопределенному идентификатору, либо переопределение макроса или поименованной константы той же самой строкой-шаблоном.

Пример 2

```
#ifdef LIMIT      /* #if определена константа LIMIT*/
#undef LIMIT
#endif
#define LIMIT 1000
```

Пример 3

```
f()
{
...
#ifdef DEBUG
  printf ("x=%d\n", x);
  printf ("y=%d\n", y);
#endif
...
}
```

Для запуска программы в режиме отладки надо включить в файл определение поименованной константы:

```
#define DEBUG
```

Тогда любые операторы, находящиеся между **`#ifdef DEBUG`** и **`#endif`** будут включены в исполнимый код. Возможно любое количество пар **`#ifdef`**и **`#endif`**. В системах UNIX константа **`DEBUG`** может быть определена не в программе, а в вызывающей ее командной строке:

```
$cc -DDEBUG prog.c
```

Библиотеки в ОС UNIX

Библиотека - это набор функций, которые могут быть использованы во многих программах. Компиляторы языка Си оснащены одной или более библиотеками. Кроме того, программист сам может создавать собственные библиотеки. Обычно библиотека - это файл, содержащий объектный код различных функций, представленный в специальном формате.

Ниже приводится список библиотечных функций, которые обычно поставляются вместе с компилятором Си:

- Ввод-вывод
- Обработка строк
- Обработка символов
- Распределение памяти
- Функции общего назначения
- Математические функции

Все эти функции поставляются с компилятором Си ОС UNIX. Стандарт ANSI языка Си языка Си специфицирует, что эти и другие функции должны

быть в составе компилятора Си. Рекомендуемый список основан на составе библиотек, поставляемых вместе с компилятором Си ОС UNIX.

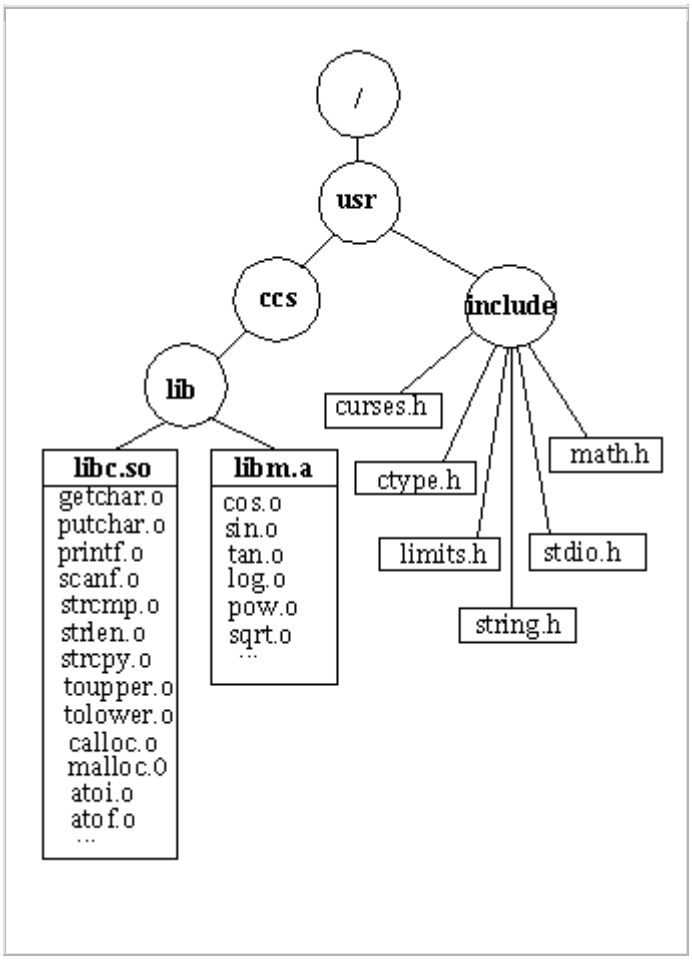

Рис. 1. Библиотеки в ОС UNIX

На рис.1 представлен сокращенный список библиотек и сопровождающих файлов, поставляемых вместе с компилятором Си ОС UNIX. Хотя места хранения этих файлов и их форматы зависят от версии системы UNIX, общая организация библиотеки такова же, как и в среде других операционных систем.

В ОС UNIX стандартные библиотеки находятся в каталоге **in/usr/ccs/lib**. Особый интерес представляет файл **/usr/ccs/lib/libc.so**, называемый "Стандартная библиотека Си". Он содержит объектный код стандартных функций ввода-вывода, функций обработки строк и символов, функций распределения памяти и функций общего назначения. Указанный файл используется редактором связей для автоматического поиска идентификатора, неопределенного в используемой программе. Например, если программа использует функцию **printf()**, редактор связей выполняет поиск определения **printf()** в каталоге **/usr/ccs/lib/libc.so** (разделяемой объектной библиотеке). Определение **printf()** отображается в виртуальное адресное пространство программы во время выполнения. Это называется динамическим

связыванием. Оно позволяет связывать разделяемые объекты с исполнимым кодом во время выполнения программ. Это позволяет обеспечить совместное использование несколькими программами размещенных в памяти библиотечных функций.

В руководстве по некоторым функциям указано, что в использующие их программы должен быть вставлен файл заголовков. Например, файл**stdio.h** - для функций ввода-вывода, **string.h** - для функций обработки строк. В ОС UNIX эти файлы находятся в каталоге **/usr/include**.

В число других библиотек, поставляемых вместе с компилятором Си ОС UNIX, входят математическая библиотека **/usr/ccs/lib/libm.a** и библиотека функций работы с экраном **/usr/ccs/lib/libcurses.a**. Библиотека функций работы с экраном - набор не зависящих от типа монитора функций для управления экраном, например, очистки экрана, позиционирования курсора, подчеркивания, выделения цветом и т.д. Если используются функции из библиотеки, отличной от стандартная библиотека Си, необходимо сообщить об этом редактору связей путем указания в командной строке флага **-l**, который должен следовать после имени исходного файла.

Пример 1

$cc -Xa prog.c -lm используются математические функции

$cc -Xa prog.c -lcurses используются функции работы с экраном