Препроцессор и библиотека языка Си. Директивы препроцессора. Функциональные макросы.

Препроцессор – анализирует программу до ее компиляции. Следуя указанным директивам, препроцессор ***заменяет*** символические сокращения в программе сущностями, которые они представляют. Препроцессор может включать другие файлы, и вы можете выбирать, какой код будет видеть компилятор. Препроцессору **ничего не известно о языке Си**, он преобразует один текст в другой. Этот процесс и называется трансляцией программы.
Этапы трансляции программы:

1. Устанавливает соответствие символов исходного кода с исходным набором символов.
   2. Обнаружение всех вхождений обратной косой черты (). Заменить все комментарии на символ ‘ ’.
   3. Поиск директив начинающихся с #
   Каждая строка `#define` состоит из:

   Как правило, для имен константных и функциональных макросов используются прописные (заглавные) буквы.
   Лексемы препроцессора Си — это отдельные “слова” в *теле* определения *макроса*. Они отделяются друг от друга пробельными символами.

```
#define FOUR 2*2
#define SIX 2 * 3
#define EIGHT 4   *   8
```

В стандарте ANSI разрешено только *переопределение*, которое дублирует предыдущее:

```
#define SIX 2 * 3
#define SIX 2   *   3
```

```
#define 2*3
```

В `#define` разрешено использовать аргументы, тем самым декларируя *функциональные макросы*.

```
#define SQUARE(X) X*X
int z = SQUARE(2);
//Значение z = 4
```

Особенности работы:

```
#define SQUARE(X) X*X
SQUARE(x+2)
//при x=5, 5+2*5+2 = 17 100/SQUARE(2)
// 100/2*2 = 100
SQUARE(++x)
//при x=5, ++5*++6 = 42
```

Чтобы поместить *аргумент макроса* в строку ставят символ #, тогда `#аргумент` – имя *аргумента макроса*, а процесс назван *преобразование в строку.*

```
#define PSQR(x) printf("Квадрат " #x "равен %d.\n", ((x) * (x)))
```

Операция ## может применятся в заменяющей части функционального макроса, для объединения лексем.

```
#define XNAME(n) x ## n
int XNAME(1) = 14; // int x1 = 14;
```

Директива `#undef`:
Отменяет заданное определение `#define`

```
#define LIMIT 400
// LIMIT существует
#undef LIMIT
// LIMIT не существует
```

Директива `#include`:
Когда процессор встречает `#include` он ищет файл с указанным в директиве именем и включает его содержимое в текущий файл.

```
#include <stdio.h> //Поиск в системных каталогах
#include "hot.h" //Поиск в текущем рабочем каталоге
#include "/usr/biff/p.h" //Поиск в каталоге /usr/biff/
```

Директивы `#ifdef`, `#else`, `#endif` и `#ifndef`:
Позволяют компилировать код в зависимости от условий.

```
#ifdef MAVIS
    #include "horse.h"
    #define STABLES 5
#else
    #include "cow.h"
    #define STABLES 15
#endif
```

Директива `#ifndef` представляет собой инверсию директивы `#ifdef`.

```
#ifndef SIZE
    #define SIZE 100
#endif
```

`#ifndef` применяется для защиты от многократного включения заголовочного файла (имя записывается в верхнем регистре, точки заменяются на нижнее подчеркивание, а также добавляется подчеркивание в качестве суффикса или префикса):

```
#ifndef _STDIO_H
    #define _STDIO_H
    //содержимое файла
#endif
```

Директивы `#if` и `#elif`:
Во многом похожи на обычные операторы **if** и **else if** языка Си.

```
#if SYS == 1
    #include "ibmpc.h"
#elif SYS == 2
    #include "vax.h"
#else
    #include "general.h"
#endif
```

Предопределенные макросы

В стандарте Си описано несколько предопределенных макросов, которые перечислены.

Макрос	Описание
`__DATE__`	Строка символов в форме "Ммм дд гггг", представляющая дату обработки препроцессором, например, Aug 24 2014
`__FILE__`	Строка символов, представляющая имя текущего файла исходного кода
`__LINE__`	Целочисленная константа, представляющая номер строки в текущем файле исходного кода
`__STDC__`	Установлен в 1 для указания, что реализация соответствует стандарту С
`__STDC_HOSTED__`	Установлен в 1 для размещаемой среды; в противном случае — 0
`__STDC_VERSION__`	Для С99 установлен в 199901L; для С11 установлен в 201112L
`__TIME__`	Время трансляции в форме "чч:мм:сс"

и другие...

Директивы `#line` и `#error`:
Директива `#line` позволяет переустанавливать нумерацию строк и имя файла, выводимые с помощью макросов __LINE_ и __FILE_

```
#line 1000
//переустанавливает текущий номер строки в 1000
#line 10 "cool.c"
//переустанавливает номер строки в 10, а имя файла - в cool.c
```

Директива `#error` заставляет препроцессор выдать сообщение об ошибке, которое включает любой текст, указанный в директиве.

```
#if __STDC_VERSION__ != 201112L
    #error NOT C11
#endif
```

Директива `#pragma`
Директива `#pragma` позволяет помещать инструкции для компилятора в исходный код.

```
#pragma once // предотвращение повторного включения заголовочных файлов
```

Библиотека си

Библиотека математических функций **math.h**

`double log(double x)`	Возвращает натуральный логарифм x
`double log10(double x)`	Возвращает логарифм x по основанию 10
`double pow(double x, double y)`	Возвращает x в степени y
`double sqrt(double x)`	Возвращает квадратный корень x
`double cbrt(double x)`	Возвращает кубический корень x
`double ceil(double x)`	Возвращает наименьшее целое, которое не меньше x
`double fabs(double x)`	Возвращает абсолютное значение x
`double floor(double x)`	Возвращает наибольшее целое, которое не больше x
`double acos(double x)`	Возвращает угол (от 0 до π радиан), косинус которого равен x
`double asin(double x)`	Возвращает угол (от $-\pi/2$ до $\pi/2$ радиан), синус которого равен x
`double atan(double x)`	Возвращает угол (от $-\pi/2$ до $\pi/2$ радиан), тангенс которого равен x
`double atan2(double y, double x)`	Возвращает угол (от $-\pi$ до π радиан), тангенс которого равен y/x
`double cos(double x)`	Возвращает косинус x (x в радианах)
`double sin(double x)`	Возвращает синус x (x в радианах)
`double tan(double x)`	Возвращает тангенс x (x в радианах)
`double exp(double x)`	Возвращает экспоненциальную функцию x (e^x)

Библиотека утилит общего назначения **stdlib.h**
rand(), srand(), malloc(), free(), exit(),
atexit() - даёт возможность указать в качестве аргумента, функцию, после которой программа выполняет exit()
qsort() - метод быстрой сортировки

Библиотека утверждений **assert.h**
Идея состоит в том, чтобы идентифицировать критические места в программе, где должны быть истинными определенные условия, и с помощью оператора **assert()** завершать программу, если одно из указанных условий нарушается. Обычно аргументом служит выражение отношения или логическое выражение.

Библиотека для строк и массивов **string.h**
strcpy(), strncpy()
memmove(), memcpy() - функции копируют n байтов из области, на которую указывает аргумент s, в область, указанную аргументом d, и обе о ни возвращают значение d.

```
void * memcpy( void *restrict d, const void *restrict s, size_t n);
void * memmove( void * d, const void * s, size_t n);
```

memcpy() - две области памяти нигде не перекрываются друг с другом.
memmove() - копирование происходит так, как будто все байты сначала помещаются во временный буфер и только затем копируются в область назначения.