Препроцессор

Лекция 9

Препроцессор

- Специальная программа, являющаяся частью компилятора языка Си. Она предназначена для предварительной обработки текста программы
- Работа препроцессора осуществляется с помощью специальных директив (указаний). Они отмечаются знаком решетка # (диез).
- Результатом работы препроцессора является обработанный текстовый файл

Основные директивы препроцессора

#include	вставляет текст из указанного файла
#define	задаёт макроопределение (макрос) или символическую константу
#undef	отменяет предыдущее определение
#if	осуществляет условную компиляцию при истинности константного выражения
#ifdef	осуществляет условную компиляцию при определённости символической константы
#ifndef	осуществляет условную компиляцию при неопределённости символической константы
#else	ветка условной компиляции при ложности выражения
#elif	ветка условной компиляции, образуемая слиянием else и if
#endif	конец ветки условной компиляции
#error	выдача диагностического сообщения
#pragma	действие, зависящее от конкретной реализации компилятора.

Включение заголовочных файлов

- Директива **#include** позволяет включать в текст программы указанный файл.
- Если файл является стандартной библиотекой и находится в папке компилятора, он заключается в угловые скобки **<>**.

```
#include <string.h>
```

- Если файл находится в текущем каталоге проекта, он указывается в кавычках "".

```
#include "myfile.h"
```

- Для файла, находящегося в другом каталоге необходимо в кавычках указать полный путь.

```
#include "C:/SFML-2.2/include/SFML/Graphics.hpp"
```

Определение макросов

- Директива **`#define`** позволяет вводить в текст программы константы и макроопределения.

```
#define Идентификатор Замена
```

- Нужно подставить строку «**Замена**» вместо каждого аргумента **Идентификатор** в исходном файле.
- Идентификатор не заменяется, если он находится
  - в комментарии
  - в строке
  - как часть более длинного идентификатора

Развертывание макросов

- Если препроцессор находит имя макроса, он заменяет его на соответствующий замещающий текст — это называется макроподстановка.
- Если в замещающем тексте макроса встречаются имена других макросов, препроцессор выполнит макроподстановку для каждого из них, и так далее, пока не будут развёрнуты все известные на данный момент макросы
- Если макрос содержит какое-то выражение, то оно обязательно должно быть заключено в скобки, иначе могут происходить всякие неочевидные вещи

Развертывание макросов

```
#define BUFFER_SIZE 32
#define EXTRA_BUFFER    (BUFFER_SIZE + 10)
#define DOUBLE_BUFFER EXTRA_BUFFER * 2
```

Здесь пробел обязателен

В исходном файле

```
char buffer[DOUBLE_BUFFER];
```

```
char buffer[EXTRA_BUFFER * 2];
```

```
char buffer[(BUFFER_SIZE + 10) * 2];
```

```
char buffer[(32 + 10) * 2];
```

После работы препроцессора

Развертывание макросов

- Макросы не допускают рекурсии

```
int flags;
#define flags flags
```

Будет определена и переменная `flags`, и символ препроцессора `flags`

Такую особенность часто используют для того, чтобы иметь возможность проверить наличие переменной (или любого другого идентификатора) с помощью директив условной компиляции `#ifdef/#else/#endif`

Ошибки в описании макросов

Правильно:

```
#define NAME "database"
Connect(NAME);
```

```
Connect("database");
```

Неправильно:

```
#define NAME "database";
Connect(NAME);
```

```
Connect("database";);
```

Макросы с параметрами

- Фактически являются макро-функциями
- Определяются макросы с параметрами с помощью той же директивы **#define**, после которой (сразу без пробелов) в круглых скобках идёт список разделённых запятыми аргументов

```
#define Идентификатор(аргумент1, …, аргументn) Замена
```

- Все аргументы макросов, используемые в математических и не только выражениях, надо обязательно заключать в скобки

Макросы с параметрами

```
#define MULT(x,y) x*y
int main()
{
    std::cout << MULT(6+3, 8-2) <<std::endl;
    return 0;
}
```

Плохо!!!

6+3*8-2

```
#define MULT(x,y) (x)*(y)
int main()
{
    std::cout << MULT(6+3, 8-2) <<std::endl;
    return 0;
}
```

Хорошо!!!

(6+3)*(8-2)

«Стражи включения»

- Позволяют избежать повторного включения заголовочных файлов
- Прописываются в заголовочных файлах
- Используются при организации многофайловых программ

Файл "function.h"

```
#ifndef FUNCTION_H
#define FUNCTION_H
int f1(int);
void f2(char*, char*);
#endif
```

Второй вариант:

Файл "function.h"

```
#pragma once
int f1(int);
void f2(char*, char*);
```

Условная компиляция

- Применяется, когда в зависимости от значения различных макросов, нужно компилировать, или нет, тот или иной кусок кода, или установить другие макросы
- Условие — это выражение препроцессора. Это может быть любая комбинация макросов, условий и целочисленных литералов, которая в результате макроподстановки превратится в выражение состоящее только из целочисленных литералов, арифметических операций и логических операторов.
- Так же здесь ещё можно использовать единственный «макрооператор» — **`defined`** — он превращается в 1, если его операнд определён, и 0 — если нет

Условная компиляция

```
#if LEVEL > 3
//текст1
#elif LEVEL > 1
//текст2
#else
//текст3
#endif
```

Условная компиляция

```
int foo;
// попытка проверить определена, ли переменная foo
#if defined(foo)
…
#endif
```

Условие if будет ложным

```
int foo;
#define foo foo
#if defined(foo)
…
#endif
```

Теперь условие if сработает

Конец