Препроцессор. Пространство имен.

Определение и особенности использования

- **Препроцессор** — это специальная программа, являющаяся частью компилятора языка **C**. Она предназначена для предварительной обработки текста программы. Препроцессор позволяет включать в текст программы файлы и вводить макроопределения.

Работа препроцессора осуществляется с помощью специальных директив (указаний).

Специфика использования директив:

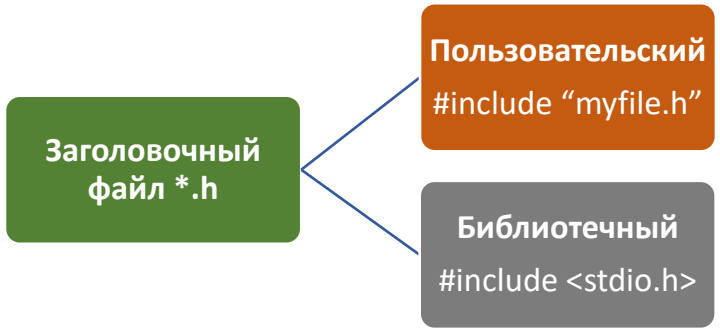
- Начинаются с символа «**#**»
- Директивы пишутся каждая на отдельной строчке;
- **Раздеритель** «;» в конце строчке **не нужны**;
- **При переносе** длинной директивы на другую строку используется «****»;
- Большинство директив можно использовать **в любом месте программы**;
- **Однострочные комментарии** лучше **не использовать** для директив;

Директивы препроцессора

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

Директива #include

Способ включения в программу стандартного или определяемого пользователем файла, который в основном пишется в начале любой программы на C/C++.



Встретив эту директиву препроцессор заменяет её на содержимое указанного файла.

Заголовочные файлы (*.h) позволяют программисту:

- Воспользоваться средствами стандартных библиотек;
- Отделение интерфейса от реализации;

server.h

```
extern int x,y,z
```

server.cpp

```
int x,y,z
```

client1.cpp

```
#include “server.h”
void f1 (){
z = x + y;
}
```

client2.cpp

```
#include “server.h”
void f2 (){
z = x - y;
}
```

client3.cpp

```
#include “server.h”
void f3 (){
z = x * y;
}
```

Директива #define

Определяет идентификатор и последовательность символов, которые будут подставляться вместо идентификатора каждый раз, когда он встретится в исходном файле. Позволяет задавать «макроопределения» (макросы).

```
#include <stdio.h>
#define N 23      // N именованная константа

int main(void)
{
   int x = N;       // int x = 23;
   printf("Number: %d", x);   // Number: 23
   return 0;
}
```

Идентификатор **должен** представлять одно слово, а заменяющая его последовательность символов **может** состоять из нескольких слов или символов, разделенных пробелами.

Макросы бывают различные:

```
#include <stdio.h>

#define BEGIN {
#define END }
#define N 23

int main(void)
BEGIN
   int x = N;
   printf("Number: %d", x);   // Number: 23
   return 0;
END
```

```
int main(void)
{
   int x = 23;
   printf("Number: %d", x);
   return 0;
}
```

```
#include <stdio.h>

#define ADD(a,b) (a+b)

int main(void)
{
   int n1 = 10;
   int n2 = 5;
   printf("%d + %d = %d", n1, n2, ADD(n1, n2));  // 10 + 5 = 15
return 0;
}
```

```
int main(void)
{
   int n1 = 10;
   int n2 = 5;
   printf("%d + %d = %d", n1, n2, (n1 + n2));
   return 0;
}
```

Следует учитывать, что директива препроцессор **не заменяет** последовательности символов в «**“**» и «**‘**» и **в комментариях**:

Директива #undef

Используется для снятия определения макроса

```
#define LEN 100
#define WIDTH 100

char array[LEN][WIDTH];

#undef LEN
#undef WIDTH

/* в данный момент как LEN, так и WIDTH не определены */
```

```
#include <stdio.h>
#define far (a-32)*5/9

int main()
{
  float a;
  a=79;       /*Температура в Фаренгейтх*/
  printf("Температура в Цельсиях:%.2f\n", far);
        /*Выводим на экран содержимое переменной b*/

  #undef far /*Освобождаем идентификатор*/

   int far = 5;
   printf ("%d\n", far);
   return 0;
}
```

Директивы условной компиляции #if, #elif, #else, #endif, #ifndef, #ifdef

Дают возможность выборочно компилировать части исходного кода вашей программы.

Этот процесс называется “условной компиляцией”

#if константное выражение
    последовательность операторов
#endif

```
/* Простой пример #if/#else. */
#include <stdio.h>

#define MAX 10

int main(void)
{
  #if MAX > 99
     printf("Компилирует для массива, размер которого больше 99.\n");
  #else
    printf("Компилирует для небольшого массива.\n");
  #endif

  return 0;
}
```

Директива #pragma

Позволяет передавать компилятору различные инструкции

#pragma link	**#pragma pack**
#pragma exit	**#pragma startup**
#pragma message	**#pragma once**

Директива **#pragma pack** задает выравнивание данных в памяти.
Синтаксис:

#pragma pack(n)
#pragma pack (push, n)
#pragma pack(pop)

Директивы **#pragma exit** и **#pagma startup** позволяют программе задать функцию (функции), которая должна вызываться либо при загрузке программы (перед вызовом **main**), либо при выходе из программы (непосредственно перед выходом из программы через **_exit**).
Синтаксис :

#pragma exit имя-функции <приоритет>
#pragma startup имя-функции <приоритет>

#pragma once — нестандартная, но широко распространённая препроцессорная директива, разработанная для контроля за тем, чтобы конкретный исходный файл при компиляции подключался строго один раз.

Директива **#pragma message** выдает сообщение при компиляции.
Синтаксис:

#pragma message ("текст"…)

Директива **#pragma link** заставляет компоновщик подключить к исполняемому модулю указанный объектный файл.
Синтаксис:

#pragma link "имя_файла"

Пространство имен

- Переменная вне **{}** и без **static**, по умолчанию считается **глобальной**.
- В глобальной области видимости совместно могут находится:
  - Имена ваших собственных глобальных переменных;
  - Имена стандартной библиотеке;
  - Глобальные имена вашего коллеги работающего над тем же проектом.
- При сборке проекта линковщик выдаст сообщение, о множественном определении имени.
- **Выход**: Использование понятие «**Пространства имЁн**». Которое является средством ограничения области видимости.
- Каждое имя становится уникальным в своем пространстве имен.

```
#include <iosteram>
namespace One {
        int my_val  = 1;
}
namespace Two {
        int my_val  = 21;
}

int main (){
   int n1 = One::my_val;
   int n2 = Two::my_val;
}
```

Особенности использования

Пространства имен позволяют сгруппировать имена по к.л. критерию:

```
namespace Version
{
  int current_version;
  int previous_version;
  void SetVersion(int version);
}
```

Для использование имен в других модулях необходимо использовать **extern:**

```
namespace One  {int version=1;}        // ->1.cpp
…..

namespace One {extern int version;}  // ->2.cpp
….
```

Имена из **namespace** являются глобальными, но ограничена область их видимости

Можно использовать **псевдонимы** пространства имен:

```
{
namespace ver = Version_alfa_122;
ver::currentVersion++;
}
```

Любое пространство имен может быть дополнено программистом по мере необходимости:

```
namespace One  {int version=1;}        // ->1.cpp
…..

namespace One {int prev_version;}     // ->2.cpp
…..
namespace One {
                        extern int version;
                        extern int prev_version;
                        int my_version=3;
}                                                  // ->3.cpp
```

Директива using

Использование директивы делает доступными имена из упомянутого пространства имен, как если бы они были объявлены глобально.

using namespace имя_пространства;

using namespace имя_пространства::имя_из_пространства;

```
namespace A{
      int aa=1;
}
Int main()
{
    using A::aa;       // глоб. видимость только aa
     aa++;             // можно обращаться к aa без префикса
     int aa = 3;       // ошибка, повторное определение
}
```

На сегодня всё !