Содержание

препроцессор

Препроцессор имеет следующие директивы:

- `#define` - определяет макрос или препроцессорный идентификатор
- `#undef` - отменяет определение макроса или идентификатора
- `#ifdef` - проверяет, определен ли идентификатор
- `#ifndef` - проверяет неопределенность идентификатора
- `#include` - включает текст из файла
- `#if` - проверяет условие выражения (как условная конструкция if)
- `#else` - задает альтернативное условие для #if
- `#endif` - окончание условной директивы #if
- `#elif` - задает альтернативное условие для #if
- `#line` - меняет номер следующей ниже строки
- `#error` - формирует текст сообщения об ошибке трансляции
- `#pragma` - определяет действия, которые зависят от конкретной реализацией компилятора
- `#` - пустая директива, по сути ничего не делает

макросы

Компилятор	**Макрос**
Borland	__BORLANDC__
Clang	__clang__
Codeplay VectorC	__VECTORC__
Digital Mars	__DMC__
Gnu	__GNUC__
Intel legacy “Classic”	__INTEL_COMPILER__
Intel LLVM based	__INTEL_LLVM_COMPILER__
Microsoft	_MSC_VER
Pathscale	__PATHSCALE__
Symantec	__SYMANTECC__
Watcom	__WATCOMC__

Архитектура	**Макрос**
x86	_M_IX86, __INTEL__, __i386__
x86-64	_M_X64, __x86_64__, __amd64__
IA64	__IA64__
DEC Alpha	__ALPHA__
Motorola Power PC	__POWERPC__

Архитектура	**Макрос**
Any little endian	__LITTLE_ENDIAN__
Any big endian	__BIG_ENDIAN__

Операционная система	**Макрос**
DOS 16 bit	__MSDOS__, _MSDOS
Windows 16 bit	_WIN16
Windows 32 bit	_WIN32, __WINDOWS__

Операционная система	**Макрос**
Windows 64 bit	_WIN64, _WIN32
Cygwin	__CYGWIN__
Mingw	__MINGW32__, __MINGW64__
Linux 32 bit	__unix__, __linux__
Linux 64 bit	__unix__, __linux__, __LP64__, __amd64__
BSD	__unix__, __BSD__, __FREEBSD__
Mac OS	__APPLE__, (__DARWIN__, __MACH__)
OS/2	__OS2__

#define

определяет идентификатор и последовательность символов, которые будут подставляться вместо идентификатора каждый раз, когда он встретится в исходном файле

```
#include <stdio.h>
#define BEGIN {
#define END }
#define N 23
#define ADD(a,b) (a+b)

int main(void)
BEGIN
  int x = N;
  printf("Number: %d", x); // Number: 23
  printf("%d + %d = %d", 10, 5, ADD(10, 5)); // 10 + 5 = 15
  return 0;
END
```

Следует учитывать, что директива препроцессор **не** заменяет последовательности символов в двойных и одинарных кавычках и в комментариях

NTB

Макрос можно определять в компиляторе, например `gcc -D ИЛЕНТИФКАТОР=ЗНАЧЕНИЕ`

#undef

undef отменяет действие макроса define

```
#include <stdio.h>

#define STRING "Good morning \n"

int main(void) {
  printf(STRING);
```

```
  #undef STRING
  #define STRING "Good afternoon \n"
  printf(STRING);
  #undef STRING
  #define STRING "Good evening \n"
  printf(STRING);
  return 0;
}
```

#ifdef

Директива #ifdef проверяет, определен ли идентификатор. Если он определен, выполняется соответствующий код.

```
#include <stdio.h>

#define FEATURE_ENABLED

int main(void) {
#ifdef FEATURE_ENABLED
  printf("Feature is enabled.\n"); // Этот код будет выполнен
#else
  printf("Feature is disabled.\n");
#endif
  return 0;
}
```

Если `FEATURE_ENABLED` будет отменен, код в первом блоке не выполнится.

#ifndef

работает аналогично #ifdef, но проверяет ложность уловия

#include

Эта директива подключает в исходный текст указанного файла.

```
#include <имя_файла> // поиск файла в стандартных системных каталогах
#include "имя_файла" // поиск файла в директории с заголовочниками (определено компилятором)
```

NTB

Вообще есть стандартный набор встроенных заголовочных файлов, который определяется

стандартом языка

- `assert.h` - отвечает за диагностику программ
- `complex.h` - для работы с комплексными числами
- `ctype.h` - отвечает за преобразование и проверку символов
- `errno.h` - отвечает за проверку ошибок
- `fenv.h` - для доступа к окружению, которое управляет операциями с числами с плавающей точкой
- `float.h` - отвечает за работу с числами с плавающей точкой
- `inttypes.h` - для работы с большими целыми числами
- `iso646.h` - содержит ряд определений, которые расширяют ряд логических операций
- `limits.h` - содержит предельные значения целочисленных типов
- `locale.h` - отвечает за работу с локальной культурой
- `math.h` - для работы с математическими выражениями
- `setjmp.h` - определяет возможности нелокальных переходов
- `signal.h` - для обработки исключительных ситуаций
- `stdalign.h` - для выравнивания типов
- `stdarg.h` - обеспечивает поддержку переменного числа параметров
- `stdatomic.h` - для выполнения атомарных операций по разделяемым данным между потоками
- `stdbool.h` - для работы с типом _Bool
- `stddef.h` - содержит ряд вспомогательных определений
- `stdint.h` - для работы с целыми числами
- `stdio.h` - для работы со средствами ввода-вывода
- `stdlib.h` - содержит определения и прототипы функций общего пользования
- `stdnoreturn.h` - содержит макрос noreturn
- `string.h` - для работы со строками
- `tgmath.h` - подключает math.h и complex.h плюс добавляет дополнительные возможности по работе с математическими вычислениями
- `threads.h` - для работы с потоками
- `time.h` - для работы с датами и временем
- `uchar.h` - для работы с символами в кодировке Unicode
- `wchar.h` - для работы с символами
- `wctype.h` - содержит дополнительные возможности для работы с символами

#if

Директива #if проверяет условие выражения, аналогично конструкции `if` в языке C.

```c
#include <stdio.h>
#define VERSION 2

int main(void) {
#if VERSION == 1
  printf("Version 1\n");
#elif VERSION == 2
  printf("Version 2\n"); // Этот код будет выполнен
#else
  printf("Unknown version\n");
```

```
#endif
  return 0;
}
```

#else

Директива #else задает альтернативное условие для директивы #if. Если предыдущее условие не было выполнено, выполняется код под **#else**.

#endif

Директива #endif завершает условную блокировку, начатую с #if, #ifdef или #ifndef.

#elif

Директива #elif позволяет задавать дополнительные условия в цепочке условных проверок, что обеспечивает удобство структурирования кода.

#line

Директива #line позволяет изменять номер строки и имя файла для отчетов об ошибках, которые выводит компилятор.

```
#include <stdio.h>
#line 100 "custom_file.c" // Устанавливает номер строки и имя файла

int main(void) {
  printf("Hello from line %d of %s\n", __LINE__, __FILE__);
  return 0;
}
```

#error

Директива #error заставляет компилятор выдавать сообщение об ошибке и завершать процесс компиляции.

```
#include <stdio.h>

#ifndef REQUIRED_FEATURE
#error "Required feature is not defined!"
#endif
```

```c
int main(void) {
  printf("Program runs successfully.\n");
  return 0;
}
```

#pragma

Директива #pragma используется для передачи специальных инструкций компилятору. Эти инструкции зависят от конкретного компилятора и могут варьироваться.

```c
#include <stdio.h>

#pragma message("Compiling the program...") // Выводит сообщение во время компиляции

int main(void) {
  printf("Hello, World!\n");
  return 0;
}
```

#

Пустая директива # не выполняет никаких действий, но может использоваться для организации или форматирования кода.