МИНИСТЕРСТВО РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

«КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ»

РЕФЕРАТ

по дисциплине: Основы алгоритмизации и программирования

на тему: Директивы препроцессора

Выполнил:

студент 1 курса 7 группы

факультет ИТ

Кишко Иван Петрович

Проверил:

доцент, к.т.н.

Белодед Николай Иванович

Минск 2020

Оглавление

[**Цели** 3](#_Toc37204291)

[**Введение** 3](#_Toc37204292)

[Назначение: 3](#_Toc37204293)

[Основные директивы препроцессора 3](#_Toc37204294)

[**Директива #include** 4](#_Toc37204295)

[**Директива #define** 4](#_Toc37204296)

[Определение констант 4](#_Toc37204297)

[Определение макросов 6](#_Toc37204298)

[**Директива #undef** 7](#_Toc37204299)

[**Директивы #ifdef и #ifndef** 7](#_Toc37204300)

[**Директива #if , #else и # elif** 9](#_Toc37204301)

[**Директива #error** 10](#_Toc37204302)

[**Операторы препроцессора # и ##** 10](#_Toc37204303)

[**Директива #pragma** 11](#_Toc37204304)

[**Стандартные макросы** 12](#_Toc37204305)

[**Директива #line** 13](#_Toc37204306)

[**Проблемы при использовании макросов** 14](#_Toc37204307)

[**Вывод** 15](#_Toc37204308)

# **Цели**

Узнать, что такое директивы препроцессора, для чего они нужны, их синтаксис, а также плюсы и минусы их использования.

# **Введение**

***Препроцессор*** — это специальная программа, являющаяся частью компилятора языка Си. 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). Они отмечаются знаком решетка #.

***Директива*** – это указание компилятору языка С++ выполнить то или иное действие в момент компиляции программы.

## Назначение:

Директивы препроцессора могут:  
- заменить какие-то лексемы в исходном тексте;  
- вставить содержимое других файлов в указанном месте;  
- подавить компиляцию части файла.

## 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  
**#line** — препроцессор изменяет номер текущей строки и имя компилируемого файла  
**#error** — выдача диагностического сообщения  
**#pragma** — действие, зависящее от конкретной реализации компилятора.

# **Директива #include**

Директива **#include** позволяет включать в текст программы указанный файл. Если файл является стандартной библиотекой и находится в папке компилятора, он заключается в угловые скобки < >.  
Если файл находится в текущем каталоге проекта, он указывается в кавычках " ". Для файла, находящегося в другом каталоге необходимо в кавычках указать полный путь.



Где Hash.h – заголовочный файл ,находящийся в текущем каталоге проекта,а iostream – стандартная библиотека С++

# **Директива #define**

Директива **#define** принимает две формы:

* определение констант;
* определение макросов.

## Определение констант

Общий синтаксис:

#define nameToken value

При использовании имени константы — nameToken, оно будет заменено значением value, то есть, грубо говоря — это та же самая переменная, значение которой изменить нельзя(без использования других директив). Смотрим пример использования константы:

#include <iostream>

#define MAXSIZE 15

using namespace std;

int main()

{

int iArr[MAXSIZE];

for (int i = 0; i < MAXSIZE; i++)

{

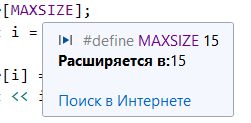
iArr[i] = i;

cout << iArr[i] << " ";

}

}

При наведении на используемую константу можно увидеть значение, на которое она будет заменена:



Результат:



В зависимости от значения константы компилятор присваивает ей тот или иной тип. С помощью суффиксов можно явно указать тип константы:

* U или u представляет целую константу в беззнаковой форме (unsigned);
* F (или f) позволяет описать вещественную константу типа float;
* L (или l) позволяет выделить целой константе 8 байт (long int);
* L (или l) позволяет описать вещественную константу типа long double

#define A 280U // unsigned int

#define B 280LU // unsigned long int

#define C 280 // int (long int)

#define D 280L // long int

#define K 28.0 // double

#define L 28.0F // float

#define M 28.0L // long double

## Определение макросов

Общий синтаксис:

#define nameMacros(arg1, arg2, ... ) expression

Пример использования:

#include <iostream>

#define print1 cout<<"k == 0"//определение константы

#define print2 cout<<"k != 0"//определение константы

#define isNull(x)x==0?true:false//функция с тернарным оператором в теле ,возвращает true ,если x==0 ,и false ,если x!=0

using namespace std;

int main()

{

int k = 5;

if (isNull(k))

print1;

else print2;

}

Результат:



Макросы можно определять в несколько строк, для этого в конце текущей строчки необходимо ставить символ '\'

Пример выше с определением в несколько строк:

#include <iostream>

#define print1 cout<<"k == 0"//определение константы

#define print2 cout<<"k != 0"//определение константы

#define isNull(x)\

x==0?\

true:false//функция с тернарным оператором в теле,возвращает true ,если x==0 ,и false , если x!=0

using namespace std;

int main()

{

int k = 5;

if (isNull(k))

print1;

else print2;

}

Результат тот же:



# **Директива #undef**

С помощью директивы #undef можно отменить макроопределение.

Общий синтаксис:

#undef nameToken

Например:

#include <iostream>

using namespace std;

int main()

{

#define pi 3.14 //определение константы

cout << pi<<endl;

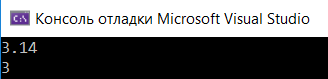
#undef pi //теперь pi не определено

#define pi 3 //определение константы

cout << pi;

}

Результат:



# **Директивы #ifdef и #ifndef**

Общий синтаксис #ifdef:

#ifdef nameToken

// код, который выполнится, если nameToken определен

#else

// код, который выполнится, если nameToken не определен

#endif

Директива **#ifdef** проверяет, был ли ранее определен макрос или символическая константа как **#define**. Если — да, компилятор включает в программу код, который находится между директивами **#ifdef** и **#else**, если nameToken ранее определен не был, то выполняется код между **#else** и **#endif**, или, если нет директивы **#else**, компилятор сразу переходит к **#endif**.

Каждая директива **#ifdef** и **#ifndef** в исходном коде должна иметь соответствующую закрывающую директиву **#endif**.

Пример:

#include<iostream>

using namespace std;

#define TEST

int main()

{

#ifdef TEST

cout << "TEST defined";

#endif

}



Общий синтаксис #ifndef:

#ifndef nameToken

// код, который выполнится, если nameToken НЕ определен

#else

// код, который выполнится, если nameToken определен

#endif

Директива **#ifndef** проверяет, был ли ранее определен макрос или символическая константа как **#define**. Если — да, компилятор включает в программу код, который находится между директивами  **#else** и **#endif**, если nameToken ранее определен не был, то выполняется код между **#ifndef** и **#else**, или, если нет директивы **#else**, компилятор сразу переходит к **#endif**.

Пример:

#ifndef RAND\_MAX //если константа не определена, подключаем библиотеки

#include <iostream>

#include <cstdlib>

#endif

using namespace std;

int main()

{

cout << RAND\_MAX;

}



Примечание: RAND\_MAX — это константа, определённая в библиотеке <cstdlib>. Для MVS RAND\_MAX = 32767, но оно может быть и больше, в зависимости от компилятора.

# **Директива #if , #else и # elif**

Директива **#if** проверяет, является ли значение value истиной и, если это так, то выполняется код, который стоит до закрывающей директивы **#endif**. В противном случае, код внутри **#if** не будет компилироваться, он будет удален компилятором, но это не влияет на исходный код в исходнике.В **#if** могут быть вложенные директивы **#elsif** и **#else**.

Общий синтаксис:

#if value

// код, который выполнится, в случае, если value - истина

#elif value1

// этот код выполнится, в случае, если value1 – истина, а value - ложь

#else

// код, который выполнится в противном случае

#endif

Пример:

#include <iostream>

#define MARK\_STUDENT 10

#define MAX\_MARK 10

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

#if MARK\_STUDENT==MAX\_MARK

cout << "Да ты отличник!";

#elif MARK\_STRUDENT>MAX\_MARK||MARK\_STRUDENT<1

cout << "Что ты такое?";

#else

cout << "Не отличник, но тоже ничего";

#endif

Результат:



# **Директива #error**

С помощью директивы **#error** можно определить строку (обратите внимание, что строка без кавычек), которая будет выведена, как сообщение об ошибке при компиляции.

#error текст без кавычек

Пример:

#include <iostream>

#define MAXSIZE -15

using namespace std;

int main()

{

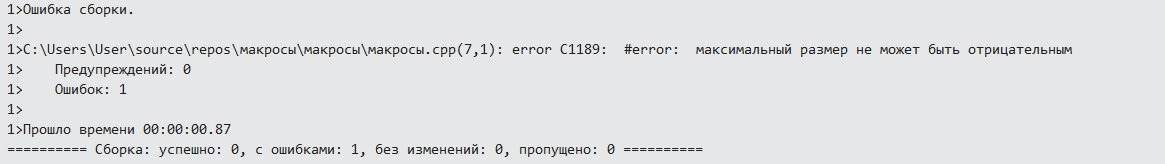
#if MAXSIZE<0

#error максимальный размер не может быть отрицательным

#endif

}

Результат:



# **Операторы препроцессора # и ##**

Стрингификация (**#**) – это преобразование фрагмента кода в строковую константу, т.е. преобразование аргумента в строку.

Пример:

#include <iostream>

#define message(x) cout<<"Привет, "#x//стрингификация

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

message(мир);

}

Результат:



Конкатенация (**##**) – это слияние двух аргументов, передаваемых в макрос. (НЕ значений переменных)

Пример:

#include <iostream>

#define concat(a, b) a ## b

int main()

{

int x = 10;

int y = 11;

int xy = 7777;

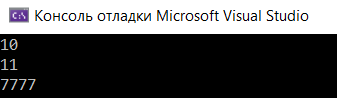
printf("%d\n%d\n", x,y);

printf("%d", concat(x, y));//произошло слияние x##y в xy, в результат выводится переменная xy

return 0;

}

Результат:



# **Директива #pragma**

Общий синтаксис:

#pragma compiler specific extension

Директива #pragma используется для доступа к специфическим расширениям компилятора. Совместное использование директивы #pragma c лексемой once просит компилятор включить файл заголовка только один раз, независимо от того, сколько раз он был импортирован:

// header.h

#pragma once

//код, размещенный здесь, включается только 1 раз

Компиляторы Microsoft C и C++ распознают следующие директивы pragma:

|  |  |  |
| --- | --- | --- |
| [alloc\_text](https://docs.microsoft.com/ru-ru/cpp/preprocessor/alloc-text?view=vs-2019) | [auto\_inline](https://docs.microsoft.com/ru-ru/cpp/preprocessor/auto-inline?view=vs-2019) | [bss\_seg](https://docs.microsoft.com/ru-ru/cpp/preprocessor/bss-seg?view=vs-2019) |
| [check\_stack](https://docs.microsoft.com/ru-ru/cpp/preprocessor/check-stack?view=vs-2019) | [code\_seg](https://docs.microsoft.com/ru-ru/cpp/preprocessor/code-seg?view=vs-2019) | [comment](https://docs.microsoft.com/ru-ru/cpp/preprocessor/comment-c-cpp?view=vs-2019) |
| [component](https://docs.microsoft.com/ru-ru/cpp/preprocessor/component?view=vs-2019) | [соответствие](https://docs.microsoft.com/ru-ru/cpp/preprocessor/conform?view=vs-2019) 1 | [const\_seg](https://docs.microsoft.com/ru-ru/cpp/preprocessor/const-seg?view=vs-2019) |
| [data\_seg](https://docs.microsoft.com/ru-ru/cpp/preprocessor/data-seg?view=vs-2019) | [deprecated](https://docs.microsoft.com/ru-ru/cpp/preprocessor/deprecated-c-cpp?view=vs-2019) | [detect\_mismatch](https://docs.microsoft.com/ru-ru/cpp/preprocessor/detect-mismatch?view=vs-2019) |
| [fenv\_access](https://docs.microsoft.com/ru-ru/cpp/preprocessor/fenv-access?view=vs-2019) | [float\_control](https://docs.microsoft.com/ru-ru/cpp/preprocessor/float-control?view=vs-2019) | [fp\_contract](https://docs.microsoft.com/ru-ru/cpp/preprocessor/fp-contract?view=vs-2019) |
| [function](https://docs.microsoft.com/ru-ru/cpp/preprocessor/function-c-cpp?view=vs-2019) | [hdrstop](https://docs.microsoft.com/ru-ru/cpp/preprocessor/hdrstop?view=vs-2019) | [include\_alias](https://docs.microsoft.com/ru-ru/cpp/preprocessor/include-alias?view=vs-2019) |
| [init\_seg](https://docs.microsoft.com/ru-ru/cpp/preprocessor/init-seg?view=vs-2019) 1 | [inline\_depth](https://docs.microsoft.com/ru-ru/cpp/preprocessor/inline-depth?view=vs-2019) | [inline\_recursion](https://docs.microsoft.com/ru-ru/cpp/preprocessor/inline-recursion?view=vs-2019) |
| [intrinsic](https://docs.microsoft.com/ru-ru/cpp/preprocessor/intrinsic?view=vs-2019) | [цикл](https://docs.microsoft.com/ru-ru/cpp/preprocessor/loop?view=vs-2019) 1 | [make\_public](https://docs.microsoft.com/ru-ru/cpp/preprocessor/make-public?view=vs-2019) |
| [управляемых](https://docs.microsoft.com/ru-ru/cpp/preprocessor/managed-unmanaged?view=vs-2019) | [message](https://docs.microsoft.com/ru-ru/cpp/preprocessor/message?view=vs-2019) | [omp](https://docs.microsoft.com/ru-ru/cpp/preprocessor/omp?view=vs-2019) |
| [once](https://docs.microsoft.com/ru-ru/cpp/preprocessor/once?view=vs-2019) | [optimize](https://docs.microsoft.com/ru-ru/cpp/preprocessor/optimize?view=vs-2019) | [pack](https://docs.microsoft.com/ru-ru/cpp/preprocessor/pack?view=vs-2019) |
| [pointers\_to\_members](https://docs.microsoft.com/ru-ru/cpp/preprocessor/pointers-to-members?view=vs-2019) 1 | [pop\_macro](https://docs.microsoft.com/ru-ru/cpp/preprocessor/pop-macro?view=vs-2019) | [push\_macro](https://docs.microsoft.com/ru-ru/cpp/preprocessor/push-macro?view=vs-2019) |
| [region, endregion](https://docs.microsoft.com/ru-ru/cpp/preprocessor/region-endregion?view=vs-2019) | [runtime\_checks](https://docs.microsoft.com/ru-ru/cpp/preprocessor/runtime-checks?view=vs-2019) | [section](https://docs.microsoft.com/ru-ru/cpp/preprocessor/section?view=vs-2019) |
| [setlocale](https://docs.microsoft.com/ru-ru/cpp/preprocessor/setlocale?view=vs-2019) | [strict\_gs\_check](https://docs.microsoft.com/ru-ru/cpp/preprocessor/strict-gs-check?view=vs-2019) | [неуправляемых](https://docs.microsoft.com/ru-ru/cpp/preprocessor/managed-unmanaged?view=vs-2019) |
| [vtordisp](https://docs.microsoft.com/ru-ru/cpp/preprocessor/vtordisp?view=vs-2019) 1 | [warning](https://docs.microsoft.com/ru-ru/cpp/preprocessor/warning?view=vs-2019) |  |

1 поддерживается только C++ компилятором.

# **Стандартные макросы**

Компилятор поддерживает эти предопределенные макросы, заданные стандартом ISO C99 и ISO C++ 17.

\_\_cplusplus, определенное как целочисленное литеральное значение, если запись преобразования компилируется как C++. В противном случае — не определяется.

\_\_DATE\_\_ дату компиляции текущего исходного файла. Дата — строковой литерал постоянной длины в формате ммм дд гггг. Название месяца MMM совпадает с сокращенным названием месяца, созданным функцией [asctime](https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/asctime-wasctime?view=vs-2019) в библиотеке времени выполнения C (CRT). Первый символ даты дд — это пробел, если значение меньше 10. Этот макрос определяется всегда.

\_\_FILE\_\_ имя текущего исходного файла. \_\_FILE\_\_ расширяется до литерала символьной строки. Чтобы убедиться, что полный путь к файлу отображается, используйте [/FC (полный путь к файлу исходного кода в Diagnostics)](https://docs.microsoft.com/ru-ru/cpp/build/reference/fc-full-path-of-source-code-file-in-diagnostics?view=vs-2019). Этот макрос определяется всегда.

\_\_LINE\_\_, определенный как целочисленный номер строки в текущем исходном файле. Значение макроса \_\_LINE\_\_ можно изменить с помощью директивы #line. Этот макрос определяется всегда.

\_\_STDC\_\_ определяется как 1 только при компиляции в формате C и при указании параметра компилятора [/Za](https://docs.microsoft.com/ru-ru/cpp/build/reference/za-ze-disable-language-extensions?view=vs-2019) . В противном случае — не определяется.

\_\_STDC\_HOSTED\_\_, определенный как 1, если реализация является размещенной реализацией, одна из которых поддерживает всю необходимую стандартную библиотеку. В противном случае определяется как 0.

\_\_STDCPP\_THREADS\_\_, определенный как 1, только в том случае, если программа может иметь более одного потока выполнения и скомпилирована C++как. В противном случае — не определяется.

\_\_TIME\_\_ время перевода предварительно обработанной записи преобразования. Время представляет собой символьный строковый литерал в формате чч: мм: СС, то же, что и время, возвращенное функцией CRT [asctime](https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/asctime-wasctime?view=vs-2019) . Этот макрос определяется всегда

# **Директива #line**

Директива #line используется для изменения содержимого \_\_LINE\_\_  и \_\_FILE\_\_, являющихся пред­определенными макросами. \_\_LINE\_\_ содержит номер компилируемой строки, а \_\_FILE\_\_  имя компилируемого файла. Стандартный вид команды #line следующий:

#line число "имя\_файла"

где число - это любое положительное число, а необязательный параметр имя\_файла является любым допустимым файловым идентификатором. Номер строки заносится в \_\_LINE\_\_, а имя файла — в \_\_FILE\_\_. #line предназначена для отладочных целей и специальных приложений.

Например, следующая директива определяет, что отсчет строк будет начинаться со 100. Оператор printf() выводит число 102, поскольку он стоит в третьей строке программы после оператора #line.  
  
#include <stdio.h>

#line 100 // переустановка счетчика строк текущего файла

int main() // строка 100

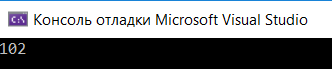
{ // строка 101

printf("%d\n", \_\_LINE\_\_); //строка 102

return 0;

}

Результат:



# **Проблемы при использовании макросов**

1. Макросы нельзя отлаживать

Решения проблемы:

* полностью избавиться от макросов, заменив их на функции (можно inline, если это важно),
* логику макросов перенести в функции, а сами макросы сделать ответственными только за передачу данных в эти функции,
* использовать только макросы, которые не требуют отладки.

1. При разворачивании макроса могут появиться странные побочные эффекты

Решения проблемы:

* отказаться от макросов в пользу функций,
* использовать макросы с понятным именем, простой реализацией и грамотно расставленными скобками, чтобы программист, использующий такой макрос легко понял, как правильно его использовать.

1. Макросы не имеют пространства имен

Решение проблемы — выбирать имена для макросов, которые с низкой вероятностью пересекутся с чем либо, например:

* имена в UPPERCASE, обычно они могут пересечься только с другими именами макросов,
* имена с префиксом (имя проекта, namespace, еще что-то уникальное), пересечение с другими именами будет возможно с очень небольшой вероятностью, но использовать такие макросы за пределами Вашего проекта людям будет немного сложнее.

# **Вывод**

Таким образом, не стоит бояться использования макросов, если используемые макросы являются «хорошими». Хорошие макросы — это макросы, которые

* не требуют отладки (внутри попросту незачем ставить точку останова)
* не имеют побочных эффектов при разворачивании (все обернуто скобочками)
* не конфликтуют с именами где-либо (выбран такой вид имен, которые с небольшой долей вероятности будут использованы кем-либо еще)
* ничего не изменяют неявно (имя точно отражает, что делает макрос, а вся работа с окружающим кодом, по возможности, ведется только через параметры и «возвращаемое значение»)