

## Структура языка программирования. Препроцессор

**План лекции «Структура языка программирования. Препроцессор»:**

- препроцессор, общие сведения;
- препроцессор C++;
- включение файлов;
- управление процессом компиляции;
- препроцессор, как отдельная команда;
- макроопределения;
- условная компиляция;
- предопределенные макросы;
- примеры.

### 1. Препроцессор:

- часть транслятора, которая выполняется до процесса трансляции;
- выполняет директивы препроцессора;
- результатом препроцессирования является текст, сформированный из исходного кода под управлением директив препроцессора.

***Препроцессор*** – программа для обработки текста.

Препроцессор может быть отдельной программой, или же интегрирован в компилятор.

Входные и выходные данные для препроцессора имеют ***текстовый формат***.

Препроцессор преобразует текст в соответствии с ***директивами препроцессора***.

В случае если текст не содержит директив препроцессора, то он остаётся без изменений.

### C/C++

В языке программирования C/C++ поддерживается встроенная поддержка препроцессора.

## В других языках программирования

### Ада:

директивы компилятора называются прагмами (от «pragmatic information» — «полезная информация»). Стандарт языка Ада определяет 39 директив.

Общий вид директивы:

```
pragma <имя_директивы> ( <параметры_директивы> );
```

### Паскаль

директивой называют указательные комментарии, которые имеют специальный синтаксис и могут использоваться везде, где разрешены комментарии.

Директивы компилятора:

- начинаются со знаков: <{\$> или <(\*\$>;
- далее следует *имя\_директивы*;
- заканчиваются знаками: <}> или <\*)>.

### *Пример.*

Указательный комментарий в Паскале {\$I "file"} – аналог директивы #include "file" языка C/C++.

### Perl

ключевое слово «use» используется как «прагма» (pragma) и выполняет импорт/экспорт пакета включаемого модуля.

### *Пример.*

```
use locale;  
use utf8;
```

### Ассемблер

Макрос – это символьное имя, заменяющее несколько команд языка ассемблера. Макросы «разворачиваются» препроцессором в последовательность кода и данных.

|    |           |   |          |   |
|----|-----------|---|----------|---|
| 12 | .DATA     |   | ; сегме  |   |
| 13 | MB_OK     | EQU 0   | ; EQU о  |   |
| 14 | STR1      | DB "Моя первая программа", 0                            | ; строка |   |
| 15 | STR2      | DB "Привет всем!", 0                                    | ; строка |   |
| 16 | HW        | DD ?  | ; двойн  |   |
| 17 |           |   |          |   |
| 18 | .CODE     |   | ; сегме  | → |
| 19 |           |   |          |   |
| 20 | main PROC |   | ; точка  |   |
| 21 | START :   |   | ; метка  |   |
| 22 |           |   |          |   |
| 23 |           | INVOKE MessageBoxA, HW, OFFSET STR2, OFFSET STR1, MB_OK |          |   |

23:

INVOKE MessageBoxA, HW, OFFSET STR2, OFFSET STR1, MB\_OK

00C31010 6A 00 push 0

00C31012 68 00 40 C3 00 push offset STR1 (0C34000h)

00C31017 68 15 40 C3 00 push offset STR2 (0C34015h)

00C3101C FF 35 22 40 C3 00 push dword ptr [HW (0C34022h)]

00C31022 E8 14 00 00 00 call \_MessageBoxA@16 (0C3103Bh)

## 2. Препроцессор C++:

директивы: `#include`, `#define`, `#if`, `#else`, `#elif`, `#endif`, `#ifdef`, `#ifndef`, `#error`, `#line`, `#pragma`, `#undef`; операторы: `defined`, `#`, `##`.

### *Препроцессор C/C++:*

текстовый процессор, который обрабатывает текст исходного файла на первой фазе компиляции.

Инструкции, регламентирующие работу компилятора, называются директивами препроцессора.

### *Назначение.*

директивы препроцессора могут:

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

Директивы препроцессора могут появляться в произвольном месте исходного текста, при этом они будут воздействовать только на оставшуюся часть исходного файла.

Препроцессор можно вызвать отдельно для обработки текста программы без ее компиляции.

### **Основные директивы препроцессора:**

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

### 3. Препроцессор C++:

директива `#include` (включение файла).

Директива `#include` вставляет содержимое заданного файла в место расположения этой директивы в исходном тексте программы.

Синтаксис:

```
#include "спецификация_пути"
```

```
#include <спецификация_пути>
```

где "спецификация\_пути" — это имя файла, с необязательным указанием его местоположения.

Кавычки и угловые скобки определяют *способ поиска* включаемых файлов:

- если спецификация файла заключена в угловые скобки, то он должен находиться в каталоге, указанном компилятором (обычно это каталог **INCLUDE**, в котором хранятся заголовочные файлы).
- если имя файла заключено в кавычки, то поиск выполняется в *рабочем каталоге*.
- если указан полный путь, то препроцессор использует его для поиска файла.

### 4. Управление процессом компиляции:

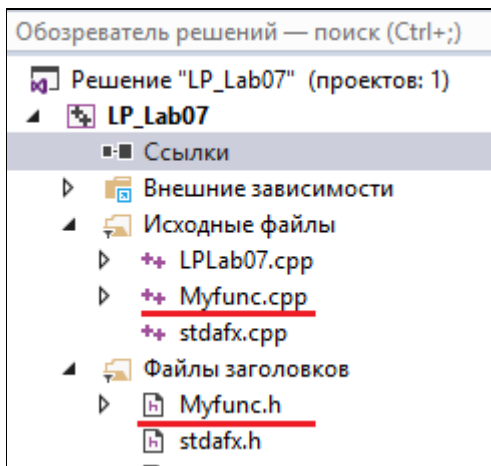
«Прагма» – это инструкция компилятору C/C++, используется для указания опций компилятору для управления его работой.

Синтаксис:

```
#pragma параметры
```

Так `#pragma once` контролирует, чтобы конкретный включаемый файл при компиляции подключался строго один раз.

## 5. Пример.



```
// Myfunc.h
#pragma once
namespace Myfunc
{
    struct DATE
    {
        short yyyy;
        short mm;
        short dd;
    };
    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2);
    unsigned long Distance(DATE d1, DATE d2);
};
```

В заголовочном файле Myfunc.h определено пространство имен Myfunc, объявлены структура DATE и прототипы 2-х функций для определения количества дней между заданными датами (тип возвращаемого параметра unsigned long).

Реализация функций содержится в Myfunc.cpp.

Функция distance вычисляет разницу в днях между двумя датами, формат даты задается в виде уууу, mm, dd.

Функция Distance вычисляет разницу в днях между двумя датами, определенными как объекты структуры DATE.

При расчетах используется функция datetoday, которая определяет количество дней до заданной даты с учетом поправки, возникшей с введением григорианского календаря. Эта функция доступна только в функции Myfunc.cpp.

## ***Историческая справка.***

***15.10.1582г.***

**Inter gravissimas** (в переводе с латинского — «Среди важнейших») — **папская булла**, выданная папой Григорием XIII о введении нового календаря, которая начинается так: «Inter gravissimas Pastoralis officii Nostri curas ...» («Среди наиболее серьёзных обязанностей нашей пастырской службы ...»). Название **Inter gravissimas** состоит из первых двух слов буллы.

Документ реформировал юлианский календарь и создал новый календарь, который получил название григорианского и сейчас используется в большинстве стран мира.

В 1582 г. изменили:

- 1) начало отсчета времени на 10 дней:
  - дату 4 октября 1582 года перенесли на 15 октября 1582 года;
- 2) сократили количество високосных годов:
  - вековые годы XX00, которые можно разделить на 400 остались високосными, например, 1600 и 2000 годы;
  - вековые годы XX00, которые нельзя разделить на 400 перестали быть високосными, например, 1800 и 1900 годы;
- 3) ввели новые таблицы для определения дня Пасхи.

**Джон фон Нейман** заложил основы учения об архитектуре вычислительных машин в 1944 году при создании первого в мире лампового компьютера ЭНИАК.

**Графиня Ада Лавлейс** — математик, дочь английского поэта Байрона, считается первым в истории программистом. Она составила первую в мире программу для вычислительной машины, разработанной Чарльзом Бэббиджем, и ввела в употребление термины «цикл» и «рабочая ячейка».

```

//Myfunc.cpp
#include "stdafx.h"
#include <iostream>
#include "Myfunc.h"
namespace Myfunc
{
    unsigned long datetoday(short yyyy, short mm, short dd)
    {
        bool G = (yyyy < 1582) || (yyyy == 1582 && mm < 10) || (yyyy == 1582 && mm == 10 && dd < 15);
        //int A = (G?0:2-(yyyy/100) + (yyyy/400)); // это правильно
        int A = 2-(yyyy/100) + (yyyy/400); // так у Microsoft
        mm = (mm <= 2? (yyyy--, mm+12): mm);
        unsigned long rc = (1461L * long(yyyy))/4L;
        unsigned long k = (306001L * long(mm+1))/10000L;
        rc += k + dd + 1720995L + A;
        return rc;
    };
    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2)
    {
        if (yyyy1 < 1 || yyyy2 < 1) throw "Date: год должен быть 1 или больше";
        if (mm1 < 1 || mm1 > 12 || mm2 < 1 || mm2 > 12) throw "Date: месяц должен быть в интервале от 1 до 12";
        if (dd1 < 1 || mm1 > 31 || dd2 < 1 || dd2 > 31) throw "Date: день должен быть в интервале от 1 до 31";
        if (dd1 > 28 && yyyy1%4 > 0) throw "Date: день должен быть в интервале от 1 до 28";
        if (dd1 > 29 && yyyy1%4 == 0) throw "Date: день должен быть в интервале от 1 до 29";
        return datetoday(yyyy1, mm1, dd1) - datetoday(yyyy2, mm2, dd2);
    };
    unsigned long Distance(DATE d1, DATE d2)
    {
        return distance(d1.yyyy, d1.mm, d1.dd, d2.yyyy, d2.mm, d2.dd);
    };
};

```

```

#include "stdafx.h"
#include <iostream>
#include <Windows.h>
#include "Myfunc.h"

int main()
{
    setlocale(LC_ALL, "rus");
    try
    {
        Myfunc::DATE fon1 = { 1903, 12, 28 }, fon2 = { 1957, 2, 9 };
        long d1 = Myfunc::Distance(fon2, fon1);
        std::cout << "Джон фон Нейман прожил " << d1 << " дней " << std::endl;
        Myfunc::DATE x1 = { 1, 1, 7 }, x2 = { 2019, 3, 20 };
        long d2 = Myfunc::Distance(x2, x1);
        std::cout << "От рождества Христова прошло " << d2 << " дней " << std::endl;
        long d3 = Myfunc::distance(2019, 3, 20, 1852, 12, 27);
        std::cout << "Со дня смерти Ады Лавлейс прошло " << d3 << " дней " << std::endl;
    }
    catch (char* e)
    {
        std::cout << "Ошибка: " << e << std::endl;
    };
    system("pause");
    return 0;
}

```

Выбрать F:\Наркевич\К лекциям\LP\_Lab07\Debug\LP\_Lab07.exe

Джон фон Нейман прожил 19402 дней  
 От рождества Христова прошло 737131 дней  
 Со дня смерти Ады Лавлейс прошло 60713 дней  
 Для продолжения нажмите любую клавишу . . .

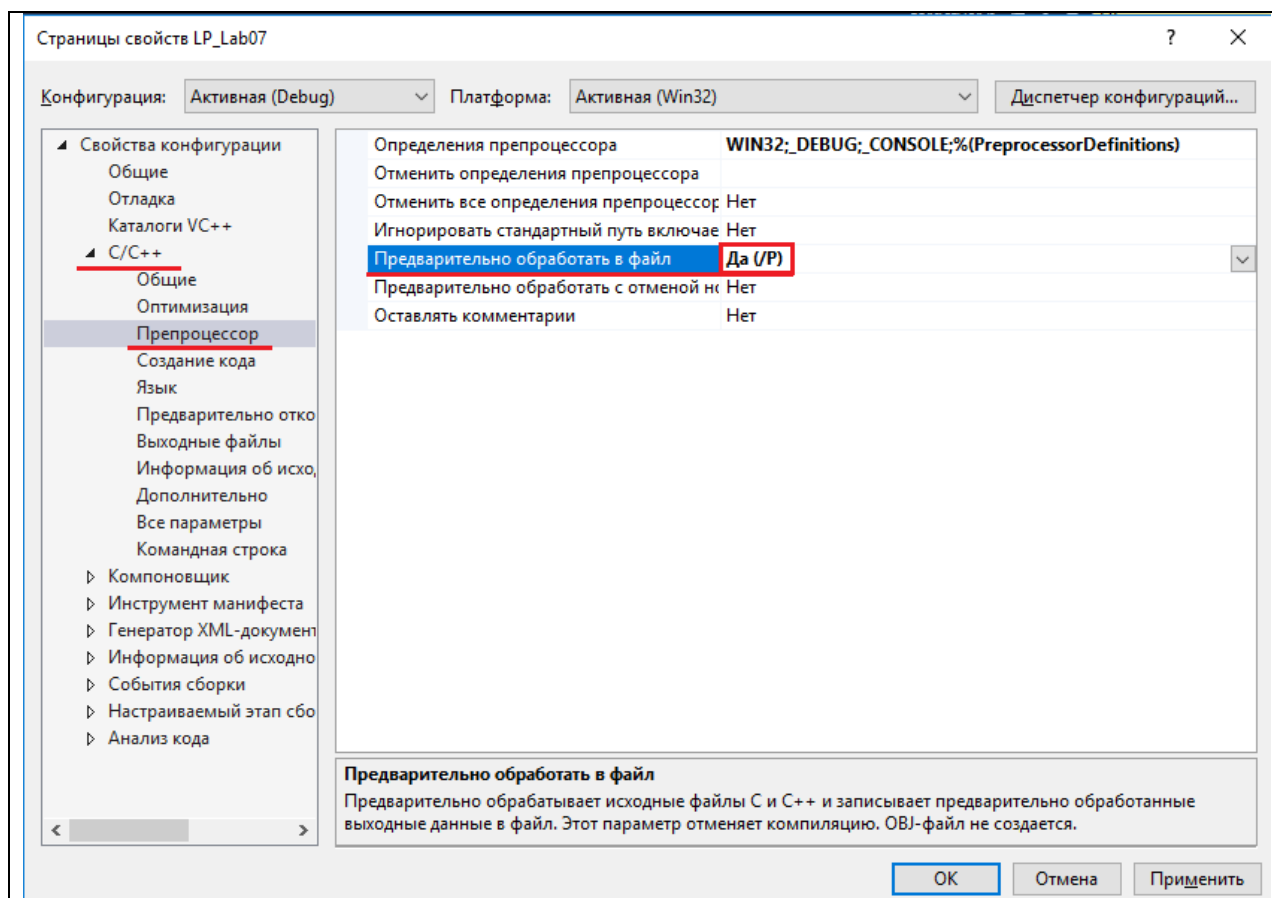
## 6. Препроцессор C++:

препроцессор, как отдельная команда (промежуточный файл с результатом работы препроцессора)

**Препроцессор** можно вызвать для выполнения обработки текста программы отдельно без ее компиляции

В этом случае объектный модуль **не создается**.

Свойства\_проекта → C/C++ → Препроцессор → установить для параметра «Предварительно обработать в файл» значение «Да (/P)»



Результат:

| ж (F:) > Наркевич > Клекциям > LP_Lab07 > LP_Lab07 > Debug |                  |                   |          |
|--|------------------|-------------------|----------|
| Имя  | Дата изменения   | Тип               | Размер   |
| LP_Lab07.tlog  | 20.03.2019 15:37 | Папка с файлами   |          |
| LP_Lab07.Build.CppClean.log                                | 20.03.2019 15:00 | Текстовый докум   | 2 КБ     |
| LP_Lab07.log   | 20.03.2019 15:37 | Текстовый докум   | 1 КБ     |
| LP_Lab07.i   | 20.03.2019 15:37 | Подготовленный... | 4 186 КБ |
| Myfunc.i   | 20.03.2019 15:37 | Подготовленный... | 1 170 КБ |



## Просмотр файла:


```
#line 5 "f:\\наркевич\\к лекциям\\lp_lab07\\lp_lab07\\lpplab07.cpp"
#line 1 "f:\\наркевич\\к лекциям\\lp_lab07\\lp_lab07\\myfunc.h"

#pragma once
namespace Myfunc
{
    struct DATE
    {
        short yyyy;
        short mm;
        short dd;
    };
    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2);
    unsigned long Distance(DATE d1, DATE d2);
}

#line 6 "f:\\наркевич\\к лекциям\\lp_lab07\\lp_lab07\\lpplab07.cpp"

int main()
{
    setlocale(0, "rus");
    try
    {
        Myfunc::DATE fon1 = { 1903, 12, 28 }, fon2 = { 1957, 2, 9 };
        long d1 = Myfunc::Distance(fon2, fon1);
        std::cout << "Джон фон Нейман прожил " << d1 << " дней " << std::endl;
        Myfunc::DATE x1 = { 1, 1, 7 }, x2 = { 2019, 3, 20 };
        long d2 = Myfunc::Distance(x2, x1);
        std::cout << "От рождества Христова прошло " << d2 << " дней " << std::endl;
        long d3 = Myfunc::distance(2019, 3, 20, 1852, 12, 27);
        std::cout << "Со дня смерти Ады Лавлейс прошло " << d3 << " дней " << std::endl;
    }
    catch (char* e)
    {
        std::cout << "Ошибка: " << e << std::endl;
    };
    system("pause");
    return 0;
}
```

## Ошибка компоновщика. Объектный модуль **не создается**:

|   | Описание   |
|---|--|
|  1 | error LNK1104: не удастся открыть файл "Debug\\LP_Lab07.obj" |

## 7. Препроцессор C++:

директивы `#define` и `#undef` (макроопределения).

Директива препроцессора `#define` определяет идентификатор и последовательность символов, которая заменит этот идентификатор в тексте программы. Идентификатор — это имя макроса. Процесс замены называется макроподстановкой.

Директива `#define` заменяет все вхождения идентификатора макроса в исходном файле на последовательность символов.

**Внимание!** Идентификатор не будет заменен, если он является частью более длинного идентификатора (подстрокой в имени более длинного идентификатора).

*Пример.*

```
#define ICY 1582
```

После определения директивы идентификатор `ICY` считается **определенным**. Препроцессор заменит каждое вхождение идентификатора `ICY` в тексте программы на целочисленный литерал `1582`.

Исходный текст программы:

```
// Myfunc.h
#pragma once
#define ICY 1582           // год  булла Inter Gravissimas
#define ICM 10            // месяц булла Inter Gravissimas
#define ICD 15            // день  булла Inter Gravissimas
#define DAYMON 306001L    // 30.6001 кол. дней в месяц
#define DAYEAR 1461L      // кол. дней в 4х годах 365.24*4

namespace mytunc
{
    struct DATE
    {
        short yyyy;
        short mm;
        short dd;
    };

    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2);
    unsigned long Distance(DATE d1, DATE d2);
};
```

```

#include "stdafx.h"
#include <iostream>
#include "Myfunc.h"

namespace Myfunc
{
    unsigned long datetoday(short yyyy, short mm, short dd)
    {
        bool G = (yyyy < ICY) || (yyyy == ICY && mm < ICM) || (yyyy == ICY && mm == ICM && dd < ICD);
        //int A = (G?0:2-(yyyy/100) + (yyyy/400)); // это правильно
        int A = 2 - (yyyy / 100) + (yyyy / 400); // так у у Microsoft
        mm = (mm <= 2 ? (yyyy--, mm + 12) : mm);
        unsigned long rc = (DAYEAR * long(yyyy)) / 4L;
        unsigned long k = (DAYMON * long(mm + 1)) / 10000L;
        rc += k + dd + 1720997L + A;
        return rc;
    }
};

```

В отладке при наведении курсора на макроопределение:

```

namespace Myfunc
{
    unsigned long datetoday(short yyyy, short mm, short dd)
    {
        bool G = (yyyy < ICY) || (yyyy == ICY && mm < ICM) || (yyyy == ICY && mm == ICM && dd < ICD);
        //int A = (G?0:2-(yyyy/100) + (yyyy/400)); //
        int A = 2 - (yyyy / 100) + (yyyy / 400); //
        mm = (mm <= 2 ? (yyyy--, mm + 12) : mm);
        unsigned long rc = (DAYEAR * long(yyyy)) / 4L;
        unsigned long k = (DAYMON * long(mm + 1)) / 10000L;
        rc += k + dd + 1720997L + A;
        return rc;
    }
};

```

#define ICM 10  
месяц булла Inter Gravissimas

## Просмотр файла с результатами работы препроцессора:

```
#line 4 "f:\\наркевич\\к лекциям\\lp_lab07\\lp_lab07\\myfunc.cpp"
#line 1 "f:\\наркевич\\к лекциям\\lp_lab07\\lp_lab07\\myfunc.h"

#pragma once

namespace Myfunc
{
    struct DATE
    {
        short yyyy;
        short mm;
        short dd;
    };
    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2);
    unsigned long Distance(DATE d1, DATE d2);
}

#line 5 "f:\\наркевич\\к лекциям\\lp_lab07\\lp_lab07\\myfunc.cpp"
namespace Myfunc
{
    unsigned long datetoday(short yyyy, short mm, short dd)
    {
        bool G = (yyyy < 1582) || (yyyy == 1582 && mm < 19) || (yyyy == 1582 && mm == 19 && dd < 15);

        int A = 2 - (yyyy / 100) + (yyyy / 400);
        mm = (mm <= 2 ? (yyyy--, mm + 12) : mm);
        unsigned long rc = (1461L * long(yyyy)) / 4L;
        unsigned long k = (306001L * long(mm + 1)) / 10000L;
        rc += k + dd + 1720995L + A;
        return rc;
    };
    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2)
    {
        if (yyyy1 < 1 || yyyy2 < 1) throw "Date: год должен быть 1 или больше";
    }
}
```

```
// Myfunc.h
#pragma once
#define ICY 1582          // год  булла Inter Gravissimas
#define ICM 10           // месяц булла Inter Gravissimas
#define ICD 15           // день  булла Inter Gravissimas
#define DAYMON 306001L   // 30.6001 кол. дней в месяц
#define DAYEAR 1461L     // кол. дней в 4х годах 365.24*4

#define YMSG "Date: год должен быть 1 или больше"
#define YMSGG "Date:  месяц должен быть в интервале от 1 до 12"
#define MMSGG31 "Date:  день должен быть в интервале от 1 до 31"
#define MMSGG28 "Date:  день должен быть в интервале от 1 до 28"
#define MMSGG29 "Date:  день должен быть в интервале от 1 до 29"

namespace Myfunc
{
    struct DATE
    {
        short yyyy;
        short mm;
        short dd;
    };
    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2);
    unsigned long Distance(DATE d1, DATE d2);
};
```

```

// Myfunc.cpp
#include "stdafx.h"
#include <iostream>
#include "Myfunc.h"

namespace Myfunc
{
    unsigned long datetoday(short yyyy, short mm, short dd) { ... }
    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2)
    {
        if (yyyy1 < 1 || yyyy2 < 1) throw YMSG;
        if (mm1 < 1 || mm1 > 12 || mm2 < 1 || mm2 > 12) throw YMSG;
        if (dd1 < 1 || dd1 > 31 || dd2 < 1 || dd2 > 31) throw MMSG31;
        if (dd1 > 28 && yyyy1 % 4 > 0) throw MMSG28;
        if (dd1 > 28 && short yyyy1 == 0) throw MMSG29;
        return datetoday(1, dd1) - datetoday(yyyy2, mm2, dd2);
    };

    unsigned long Distance(DATE d1, DATE d2)
    {
        return distance(d1.yyyy, d1.mm, d1.dd, d2.yyyy, d2.mm, d2.dd);
    }
}

```

Обычно, директивы препроцессора располагаются в одной строке. Если требуется продолжить макрос на следующей строке, надо **явно** указать признак продолжения – символ обратный слэш (\).

### Функциональные макросы.

Макроопределения могут иметь параметры. Этот вид макроса называется функциональным.

Формальный аргумент *x* в определении макроса ERRMM(*x*) будет заменен значением фактического параметра *mm1*.

Директива препроцессора `#undef` аннулирует ранее определенный макрос. В нашем примере это:

```
#undef ERRMM
```

После этой директивы макрос становится неопределенным, и последующие ссылки на него будут приводить к **ошибке компиляции**.

Использование функциональных макроопределений увеличивает скорость выполнения программы, т.к. в ней отсутствуют вызовы функций. Но при этом происходит дублирование фрагментов программы, и размер ее может значительно увеличиться, если размер функциональных макросов достаточно велик.

**Внимание.** Использование скобок в макросах гарантирует правильную подстановку:

|                                       |                         |   |  |
|---------------------------------------|-------------------------|---|--|
| <code>#define sqr(X) X*X</code>       | <code>sqr(a)+1</code>   | → | <code>a * a + 1</code>                   |
|                                       | <code>sqr(a+1)+1</code> | → | <code>a + 1 * a + 1 + 1 // Ошибка</code> |
| <code>#define sqr(X) ((X)*(X))</code> | <code>sqr(a+1)+1</code> | → | <code>(a + 1) * (a + 1) + 1</code>       |

```

// Myfunc.cpp
#include "stdafx.h"
#include "Myfunc.h"

#include <iostream>

#define ERRMM(x)      (x < 1 || x > 12)      // ошибка в месяце
#define ERRDD(x)      (x < 1 || x > 31)      // ошибка в дне месяца
#define ERRDD28(x,y)  (x > 28 && y % 4 > 0)  // ошибка в дне месяца
#define ERRDD29(x,y)  (x > 28 && y % 4 == 0) // ошибка в дне месяца

namespace Myfunc
{
    unsigned long datetoday(short yyyy, short mm, short dd) { ... }
    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2)
    {
        #ifdef TESTPARM
            std::cout << "distance: yyyy1 = " << yyyy1 << " mm1 = " << mm1 << " dd1 = " << dd1 << std::endl;
            std::cout << "distance: yyyy2 = " << yyyy2 << " mm2 = " << mm2 << " dd2 = " << dd2 << std::endl;
        #endif
        if (yyyy1 < 1 || yyyy2 < 1) throw YMSG;
        if (ERRMM(mm1) || ERRMM(mm2)) throw MMSG;
        if (ERRDD(dd1) || ERRDD(dd2)) throw MMSG31;
        if (ERRDD28(dd1, yyyy1) || ERRDD28(dd2, yyyy2)) throw MMSG28;
        if (ERRDD29(dd1, yyyy1) || ERRDD29(dd2, yyyy2)) throw MMSG29;
        return datetoday(yyyy1, mm1, dd1) - datetoday(yyyy2, mm2, dd2);
    };

    unsigned long Distance(DATE d1, DATE d2) { ... }
}

#undef ERRMM
#undef ERRDD
#undef ERRDD28
#undef ERRDD29

```

## 8. Преппроцессор C++:

директивы условной компиляции: #ifdef, #ifndef, #if, defined()

Директивы условной компиляции #ifdef и #ifndef управляют компиляцией части исходного файла.

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

Директивы условной компиляции позволяют определять истинность некоторых условий и в зависимости от результата проверки определяют, какие блоки исходного кода будут переданы на обработку компилятору, и какие блоки будут удалены из компиляции.

**Пример применения:** такая возможность позволяет выполнить автоматическую настройку компиляции программы на разных платформах (операционная система + компилятор + процессор); получить из одного исходного файла нескольких вариантов программы (отладочные версии с промежуточным выводом результатов и рабочая версия).

Директива `#ifdef` проверяет определено ли в данный момент имя макроса `TESTPARM`. Результат проверки будет иметь значение "истина", если заданный макрос определен, в противном случае – "ложь":

```
// Myfunc.cpp
#include "stdafx.h"
#include "Myfunc.h"

#ifdef TESTPARM
    #include <iostream>
#endif

#define ERRMM(x)      (x < 1 || x > 12)      // ошибка в месяце
#define ERRDD(x)      (x < 1 || x > 31)      // ошибка в дне месяца
#define ERRDD28(x,y)  (x > 28 && y % 4 > 0)  // ошибка в дне месяца
#define ERRDD29(x,y)  (x > 28 && y % 4 == 0) // ошибка в дне месяца
```

```
unsigned long datetoday(short yyyy, short mm, short dd)
{
    #ifdef TESTPARM
        std::cout << "datetoday: yyyy = " << yyyy << " mm = " << mm << " dd = " << dd << std::endl;
    #endif
    bool G = (yyyy < ICY) || (yyyy == ICY && mm < ICM) || (yyyy == ICY && mm == ICM && dd < ICD);
    //int A = (G?0:2-(yyyy/100) + (yyyy/400)); // это правильно
    int A = 2 - (yyyy / 100) + (yyyy / 400); // так у у Microsoft
    mm = (mm <= 2 ? (yyyy--, mm + 12) : mm);
    unsigned long rc = (DAYEAR * long(yyyy)) / 4L;
    unsigned long k = (DAYMON * long(mm + 1)) / 10000L;
    rc += k + dd + 1720997L + A;
    return rc;
};
```

```
unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2)
{
    #ifdef TESTPARM
        std::cout << "distance: yyyy1 = " << yyyy1 << " mm1 = " << mm1 << " dd1 = " << dd1 << std::endl;
        std::cout << "distance: yyyy2 = " << yyyy2 << " mm2 = " << mm2 << " dd2 = " << dd2 << std::endl;
    #endif
    if (yyyy1 < 1 || yyyy2 < 1) throw YMSG;
    if (ERRMM(mm1) || mm2 < 1 || ERRMM(mm2)) throw YMSG;
    if (ERRDD(dd1) || ERRDD(dd2)) throw MMSG31;
    if (ERRDD28(dd1, yyyy1) || ERRDD28(dd2, yyyy2)) throw MMSG28;
    if (ERRDD29(dd1, yyyy1) || ERRDD29(dd2, yyyy2)) throw MMSG29;
    return datetoday(yyyy1, mm1, dd1) - datetoday(yyyy2, mm2, dd2);
};
```

```
unsigned long Distance(DATE d1, DATE d2)
{
    #ifdef TESTPARM
        std::cout << "Distance: d1 = " << d1.yyyy << ", " << d1.mm << ", " << d1.dd << std::endl;
        std::cout << "Distance: d2 = " << d2.yyyy << ", " << d2.mm << ", " << d2.dd << std::endl;
    #endif
    return distance(d1.yyyy, d1.mm, d1.dd, d2.yyyy, d2.mm, d2.dd);
}
```

```

#include "stdafx.h"
#include <iostream>
#include <Windows.h>
#include "Myfunc.h"

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    try
    {
        Myfunc::DATE fon1 = { 1903, 12, 28 }, fon2 = { 1957, 2, 9 };
        long d1 = Myfunc::Distance(fon2, fon1);
        std::cout << "Джон фон Нейман прожил " << d1 << " дней" << std::endl;
        Myfunc::DATE x1 = { 1, 1, 7 }, x2 = { 2017, 3, 18 };
        long d2 = Myfunc::Distance(x2, x1);
        std::cout << "От рождества Христова прошло " << d2 << " дней" << std::endl;
        long d3 = Myfunc::distance(2017, 3, 18, 1852, 12, 27);
        std::cout << "Со дня смерти Ады Лавлейс прошло " << d3 << " дней" << std::endl;
    }
    catch (char* e){
        std::cout << "Ошибка: " << e << std::endl;
    };
    system("pause");
    return 0;
}

```

Определение #define TESTPARM размещается в заголовочном файле “stdafx.h”.

```

// stdafx.h: включаемый файл для стандартных системных включаемых файлов
// или включаемых файлов для конкретного проекта, которые часто используются, но
// не часто изменяются
//

#pragma once

#include "targetver.h"

#include <stdio.h>
#include <tchar.h>

#define TESTPARM

```



Результат выполнения с *определенным* макросом TESTPARM:

```
D:\Adel\LPPrim\LP_Lab07\Debug\LP_Lab07.exe
Distance: d1 = 1957, 2, 9
Distance: d2 = 1903, 12, 28
distance: yyyy1 = 1957 mm1 = 2 dd1 = 9
distance: yyyy2 = 1903 mm2 = 12 dd2 = 28
datetoday: yyyy = 1957 mm = 2 dd = 9
datetoday: yyyy = 1903 mm = 12 dd = 28
Джон фон Нейман прожил 19402 дней
Distance: d1 = 2017, 3, 18
Distance: d2 = 1, 1, 7
distance: yyyy1 = 2017 mm1 = 3 dd1 = 18
distance: yyyy2 = 1 mm2 = 1 dd2 = 7
datetoday: yyyy = 2017 mm = 3 dd = 18
datetoday: yyyy = 1 mm = 1 dd = 7
От рождества Христова прошло 736399 дней
distance: yyyy1 = 2017 mm1 = 3 dd1 = 18
distance: yyyy2 = 1852 mm2 = 12 dd2 = 27
datetoday: yyyy = 2017 mm = 3 dd = 18
datetoday: yyyy = 1852 mm = 12 dd = 27
Со дня смерти Ады Лавлейс прошло 59981 дней
Для продолжения нажмите любую клавишу . . .
```

Результат выполнения с *неопределенным* макросом TESTPARM:

```
// stdafx.h: включаемый файл для стандартных системных включаемых файлов
// или включаемых файлов для конкретного проекта, которые часто используются, но
// не часто изменяются
//

#pragma once

#include "targetver.h"

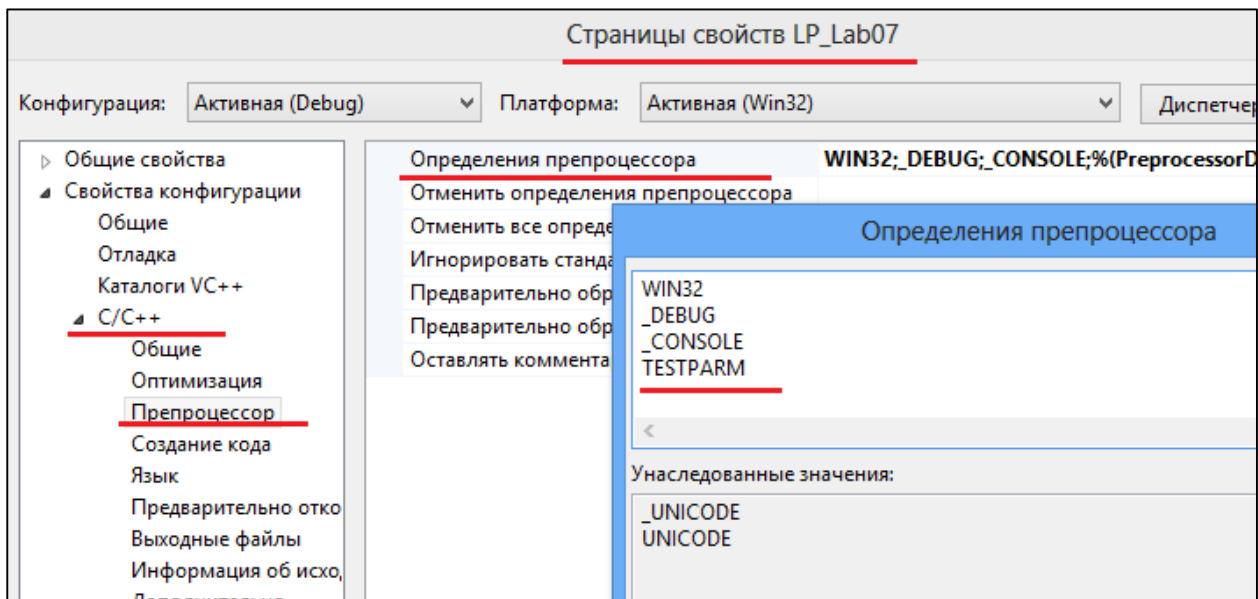
#include <stdio.h>
#include <tchar.h>

// #define TESTPARM
```

```
C:\Windows\system32\cmd.exe
Джон фон Нейман прожил 19402 дней
От рождества Христова прошло 736399 дней
Со дня смерти Ады Лавлейс прошло 59981 дней
Для продолжения нажмите любую клавишу . . .
```

Определить макрос можно в IDE.

Свойства\_проекта → C/C++ → Препробессор → вводим определение макроса TESTPARM для препробессора:



Результат выполнения с *определенным* в IDE макросом TESTPARM:

```
C:\Windows\system32\cmd.exe

Distance: d1 = 1957, 2, 9
Distance: d2 = 1903, 12, 28
distance: уууу1 = 1957 mm1 = 2 dd1 = 9
distance: уууу2 = 1903 mm2 = 12 dd2 = 28
datetoday: уууу = 1957 mm = 2 dd = 9
datetoday: уууу = 1903 mm = 12 dd = 28
Джон фон Нейман прожил 19402 дней
Distance: d1 = 2017, 3, 18
Distance: d2 = 1, 1, 7
distance: уууу1 = 2017 mm1 = 3 dd1 = 18
distance: уууу2 = 1 mm2 = 1 dd2 = 7
datetoday: уууу = 2017 mm = 3 dd = 18
datetoday: уууу = 1 mm = 1 dd = 7
От рождества Христова прошло 736399 дней
distance: уууу1 = 2017 mm1 = 3 dd1 = 18
distance: уууу2 = 1852 mm2 = 12 dd2 = 27
datetoday: уууу = 2017 mm = 3 dd = 18
datetoday: уууу = 1852 mm = 12 dd = 27
Со дня смерти Ады Лавлейс прошло 59981 дней
Для продолжения нажмите любую клавишу . . .
```

```

// Myfunc.cpp
#include "stdafx.h"
#include "Myfunc.h"

#if defined (TESTPARM) || defined (TESTRET)
    #include <iostream>
#endif

#define ERRMM(x)      (x < 1 || x > 12)      // ошибка в месяце
#define ERRDD(x)      (x < 1 || x > 31)      // ошибка в дне месяца
#define ERRDD28(x,y)  (x > 28 && y % 4 > 0)  // ошибка в дне месяца
#define ERRDD29(x,y)  (x > 28 && y % 4 == 0) // ошибка в дне месяца

namespace Myfunc
{
    unsigned long datetoday(short yyyy, short mm, short dd)
    {
        #ifndef TESTPARM
            std::cout << "datetoday: yyyy = " << yyyy << " mm = " << mm << " dd = " << dd << std::endl;
        #endif

        bool G = (yyyy < ICY) || (yyyy == ICY && mm < ICM) || (yyyy == ICY && mm == ICM && dd < ICD);
        //int A = (G?0:2-(yyyy/100) + (yyyy/400)); // это правильно
        int A = 2 - (yyyy / 100) + (yyyy / 400); // так у Microsoft
        mm = (mm <= 2 ? (yyyy--, mm + 12) : mm);
        unsigned long rc = (DAYEAR * long(yyyy)) / 4L;
        unsigned long k = (DAYMON * long(mm + 1)) / 10000L;
        rc += k + dd + 1720997L + A;

        #ifndef TESTRET
            std::cout << "datetoday: " << rc << std::endl;
        #endif

        return rc;
    }
};

```

Добавляем еще один макрос TESTRET, который используется для управления макросом TESTPARM:

```

// stdafx.h: включаемый файл для стандартных системных включаемых файлов
// или включаемых файлов для конкретного проекта, которые часто используются, но
// не часто изменяются
//

#pragma once

#include "targetver.h"

#include <stdio.h>
#include <tchar.h>

#define TESTPARM
#define TESTRET

```

## 9. Препроцессор C++:

директивы: #if, #else

*Директива условной компиляции* #if позволяет подавить компиляцию части исходного файла.

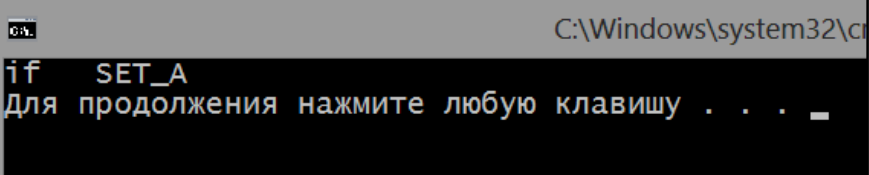
Каждая директива #if в исходном файле должна иметь соответствующую закрывающую директиву #endif.

```
#include "stdafx.h"
#include <iostream>

#define SET_A

int _tmain(int argc, _TCHAR* argv[])
{
    #ifdef SET_A
        std::cout << "if SET_A" << std::endl;
    #else
        std::cout << "else SET_A" << std::endl;
    #endif

    system("pause");
    return 0;
}
```

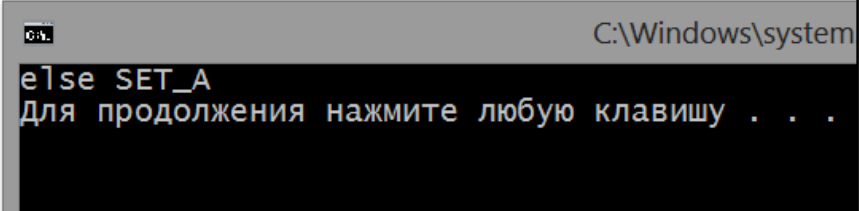


```
#include "stdafx.h"
#include <iostream>

// #define SET_A

int _tmain(int argc, _TCHAR* argv[])
{
    #ifdef SET_A
        std::cout << "if SET_A" << std::endl;
    #else
        std::cout << "else SET_A" << std::endl;
    #endif

    system("pause");
    return 0;
}
```



```
#include "stdafx.h"
#include <iostream>

// #define SET_A
#define SET_B

int _tmain(int argc, _TCHAR* argv[])
{
    #ifdef SET_A
        std::cout << "if SET_A" << std::endl;
    #elif defined(SET_B)
        std::cout << "elif SET_B" << std::endl;
    #else
        std::cout << "else " << std::endl;
    #endif

    system("pause");
    return 0;
}
```

C:\Windows\system...  
elif SET\_B  
Для продолжения нажмите любую клавишу . . .

```
#include "stdafx.h"
#include <iostream>

// #define SET_A
#define SET_B
#define SET_C
#define SET_D

int _tmain(int argc, _TCHAR* argv[])
{
    #ifdef SET_A
        std::cout << "if SET_A" << std::endl;
    #endif
    #if !defined(SET_A)
        std::cout << "elif !SET_A" << std::endl;
    #endif
    #if defined(SET_B) && defined(SET_C)
        std::cout << "if SET_B && SET_C" << std::endl;
    #endif
    #if defined(SET_B) || defined(SET_C)
        std::cout << "if SET_B || SET_C" << std::endl;
    #endif

    system("pause");
    return 0;
}
```

C:\Windows\system...  
elif !SET\_A  
if SET\_B && SET\_C  
if SET\_B || SET\_C  
Для продолжения нажмите любую клавишу . . .

## 10. Препроцессор C++:

директива `#error`

Синтаксис:

`#error текст`

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

```
#include "stdafx.h"
#include <iostream>

// #define SET_A
#define SET_B
#define SET_C
#define SET_D

int _tmain(int argc, _TCHAR* argv[])
{
    #if defined(SET_B) && defined(SET_C)
        #error SET_B && SET_C
    #endif
    #ifdef SET_A
        #error SET_A
    #endif

    system("pause");
    return 0;
}
```

Ошибок: 2 | Предупреждений: 0

|   | Описание                                       |
|---|--|
| 1 | error C1189: #error : SET_B && SET_C           |
| 2 | IntelliSense: директива #error: SET_B && SET_C |

## 11. Препроцессор C++:

предопределенные макросы

Компилятор C++ автоматически определяет некоторые макросы, например:

`__LINE__`

Этот макрос заменяется номером текущей строки в форме десятичной целой константы.

Несмотря на то, что он называется предопределенным макросом, значение его меняется динамически в зависимости от местоположения макроса.

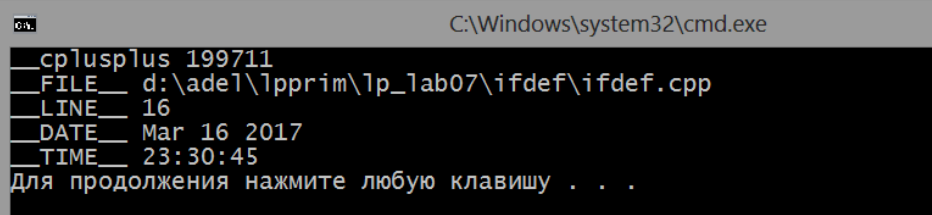
Этот макрос в сочетании с макросом `__FILE__` можно использовать при генерации сообщения об ошибке для вывода несоответствия, обнаруженного программой. В этом случае сообщение будет содержать номер строки с именем исходного файла, в котором была обнаружена ошибка.

```
#include "stdafx.h"
#include <iostream>

// #define SET_A
#define SET_B
#define SET_C
#define SET_D

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout << "__cplusplus" << __cplusplus << std::endl; // 199711L - стандарт C++
    std::cout << "__FILE__" << __FILE__ << std::endl;        // файл с кодом
    std::cout << "__LINE__" << __LINE__ << std::endl;        // тек. строка
    std::cout << "__DATE__" << __DATE__ << std::endl;        // тек. дата
    std::cout << "__TIME__" << __TIME__ << std::endl;        // тек. время

    system("pause");
    return 0;
}
```



## 12. Препроцессор C++:

операторы препроцессора # и ##.

**Стрингификация (#)** – это преобразование фрагмента кода в строковую константу, т.е. преобразование аргумента в строку. Например, в результате стрингификации STR(hello) аргумент преобразуется в символьную строку "hello".

**Конкатенация (##)** (оператор конкатенации) – это конкатенация двух строковых констант. При работе с макросами, это означает объединение двух лексических единиц в одну более длинную. Например, один аргумент макроса может быть объединен с другим аргументом или с каким-либо текстом.

```
#include "stdafx.h"
#include <iostream>
#include <locale>

#define STR(x) #x //стрингификация
#define ZZZ(y) #y //стрингификация
#define CONSTR1(a, b) STR(a##b) //конкатенация + стрингификация
#define CONSTR2(a, b) STR(b##a) //конкатенация + стрингификация
#define APPLY(x, y) x(y)
#define CON1(a,b) a##b //конкатенация
#define CON2(a,b) b##a //конкатенация

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    std::cout << "STR(hello) = " << STR(hello) << std::endl;
    std::cout << "ZZZ(привет) = " << ZZZ(привет) << std::endl;
    std::cout << "CONSTR1(sss, xxxx) = " << CONSTR1(sss, xxxx) << std::endl;
    std::cout << "CONSTR2(sss, xxxx) = " << CONSTR2(sss, xxxx) << std::endl;
    std::cout << "APPLY(STR, tttt) = " << APPLY(STR, tttt) << std::endl;
    std::cout << "CON1(S, TR)(12345) = " << CON1(S, TR)(12345) << std::endl;
    std::cout << "CON2(TR, S)(54321) = " << CON2(TR, S)(54321) << std::endl;

    system("pause");

    return 0;
}

#ifdef SET_A
// std::cout
```

