В.Л. Тарасов

Лекции по программированию на C++

Лекция 7

Препроцессор

Препроцессор изменяет исходный текст программы: удаляет комментарии и выполняет *директивы препроцессора*, которыми являются строки программы, начинающиеся знаком #.

Как правило, препроцессор встроен в компилятор, но существуют и автономные препроцессоры, с помощью которых можно получить файл, обработанный препроцессором.

7.1. Директивы препроцессора

Директивы препроцессора имеют следующий формат:

```
# [имя_директивы] [лексемы_препроцессора]
```

Препроцессор не относится к языку программирования, его назначение состоит в обработке исходного текста программы, поэтому *препроцессорные лексемы* не совпадают с рассмотренными ранее лексемами языка C++. Набор препроцессорных лексем включает: символьную константу, имя объекта-заголовка, идентификатор, знак операции, препроцессорное число, строку символов и каждый символ, не являющийся пробелом, который нельзя отнести к перечисленным выше.

При записи директив допускаются пробелы перед знаком #, между этим знаком и именем директивы, а также перед лексемами препроцессора, между ними и после них.

Директивы могут стоять в любом месте программы, они оказывают влияние от точки их появления до конца транслируемой компоненты.

Директивы препроцессора перечислены в табл.**7**.1.

Директивы препроцессора можно располагать в нескольких строках, используя в качестве знака переноса символ обратной наклонной черты (\), то есть можно, например, писать:

```
#include        \
<iostream>
```

Директива #pragma имеет следующий синтаксис:

```
#pragma ДИРЕКТИВА_pragma
```

Таблица 7.1. Директивы препроцессора

Имя	Назначение
#define	Определение макроса
#undef	Отмена определения макроса
#include	Подстановка текста из внешнего файла
#if	Компиляция, если выражение истинно
#ifdef	Компиляция, если макрос определен
#ifndef	Компиляция, если макрос не определен
#else	Альтернатива для #if, #ifdef, #ifndef
#elif	Составная директива else/if
#endif	Окончание группы компиляции по условию
#line	Замена новым значением номера строки или имени текущего файла
#error	Прерывает компиляцию с выдачей сообщения
#pragma	Действие определяется системой программирования
#	Пустая директива

Различные реализации языка C++ имеют различные наборы команд ДИРЕКТИВА_pragma, учитывающие конкретные особенности компилятора и операционной системы.

7.2. Макросы

Макросом или макроопределением называется идентификатор, связанный директивой #define с лексемой или последовательностью лексем. Определение макроса имеет вид:

```
#define МАКРОС  ЗНАЧЕНИЕ
```

Здесь МАКРОС – это имя макроса. В имени макроса пробелы не допускаются. Препроцессор заменяет всякое вхождение макроса в текст программы значением, связанным с ним директивой #define, всюду от точки его определения в программе до конца файла. Если макрос входит в состав символьной или строковой константы, то замена не производится. Например, в стандартном заголовочном файле cmath определены макросы M_E и M_PI для чисел e и π, округленных до 21 значащей цифры:

```
#define M_E         2.71828182845904523536
#define M_PI        3.14159265358979323846
```

Строку программы:

```
y = sin(M_PI / 3) / M_E;
```

препроцессор преобразует в строку:

```
y = sin(3.14159265358979323846 / 3) / 2.71828182845904523536;
```

которую затем обрабатывает компилятор.

Можно определять макросы с параметрами. Пример такого макроса приведен в программе 7.1.

Стандартом языка C++ предусмотрено несколько встроенных макросов, часть из них перечислена в табл.7.2.

Таблица 7.2. Встроенные макросы языка C++

Макрос	Описание
__DATE__	Строка, представляющая дату в форме mmm dd yyyy текущую дату
__FILE__	Строка, представляющая имя текущего файла
__LINE__	Целое, равное номеру текущей строки исходного файла
__TIME__	Строка, представляющая в форме hh:mm:ss текущее время

Кроме стандартных макросов, системы программирования могут включать специфичные для них макроопределения, которые можно изучить по документации.

Программа 7.1. Возможности препроцессора

Приводимая далее программа дает образцы использования различных директив препроцессора.

```
// Файл Preproc.cpp
// Демонстрация работы препроцессора

#include <iostream>

#define _USE_MATH_DEFINES  // Делает доступными M_PI и M_E

#include <cmath>
using namespace std;

/* greater_than(a,b): макрос с  параметрами, проверяющий, больше ли
первый аргумент второго. Аргументы в определении макроса рекомендуется
заключать в скобки во избежание ошибок */

#define greater_than(a,b) ((a) > (b) ? 1 : 0)

#define SMALL             // SMALL - пустой макрос
```

```
void main()
{
    int c = 5, d = 7;
    if(greater_than(c, d))
        cout << "\nc > d";
    else
        cout << "\nc <= d";
#undef greater_than             // Отмена определения greater_than
#ifdef  SMALL                   // Проверка, определен ли макрос SMALL
    cout << "\nSMALL is defined";       // Строка остается, если SMALL
#else                                   // определен
    cout << "\nSMALL is not defined";   // Строка остается, если SMALL
#endif                                  // не определен
#undef  SMALL               // Отмена определения макроса SMALL
#ifndef SMALL               // Проверка, что макрос SMALL не определен
    cout << "\nSMALL is not defined";   // Строка остается, если SMALL
#else                                   // не определен
    cout << "\nSMALL is defined ";      // Строка остается, если SMALL
#endif                                  // определен
    cout << "\nThe Name current file: " << __FILE__;  // Использование
    cout << "\nDate: " << __DATE__;                   // встроенных
    cout << "\nNumber the current Line "<< __LINE__;  // макросов
    cout << "\nTime " << __TIME__;
#define PROBLEM 0           // Определение макроса PROBLEM
#if PROBLEM                 // Проверка значения макроса PROBLEM
#error We have a problem    // Сюда не попадаем, если PROBLEM равно 0
#endif
    double y = sin(M_PI / 3) / M_E;
    cin.get();              // Ждем нажатия Enter
}
```

В результате работы данной программы на экране появилось следующее:

```
c <= d
SMALL is defined
SMALL is not defined
The Name current file:
d:\auser\methodworks\programming\progr_07_01_preproc\preproc.cpp
Date: Nov  5 2015
Number the current Line 33
Time 13:42:10
```

Если дать макросу `PROBLEM` значение, отличное от 0, например:

```
#define PROBLEM 1
```

то компиляция прервется на строке:

```
#error We have a problem
```

с выдачей сообщения:

```
We have a problem
```

7.3. Препроцессор в Visual C++

Компилятор Visual C++ можно настроить на создание файла с результатами работы препроцессора. Для этого надо открыть окно свойств проекта командой **Проект**, **Свойства**, в окне свойств открыть ветку **Свойства конфигурации**, **C/C++**, **Препроцессор** и для свойства **Создавать файл препроцессора** выбрать один из режимов создания, например, **Без номеров строк** (рис.7.1).

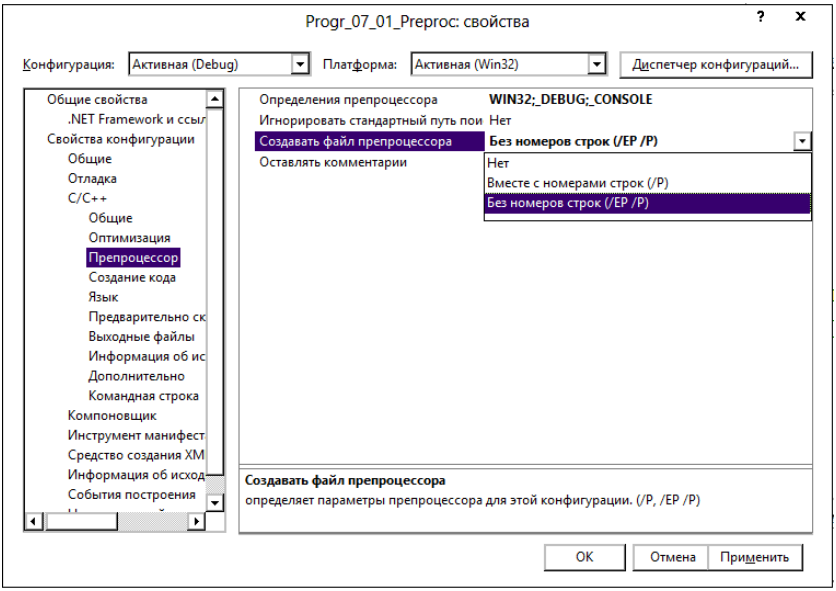

Рис. 7.1. Создание файла препроцессора

При компиляции cpp-файла препроцессор создает файл с расширением `.i` (рис.7.2).

Для файла `Progr_07_01_Preproc.cpp` из программы 7.1 препроцессор создает файл `Progr_07_01_Preproc.i` из 57223 строк. Большая часть их - содержимое заголовочных файлов. Далее приведена часть этого файла с обработанным препроцессором текстом функции `main()`.

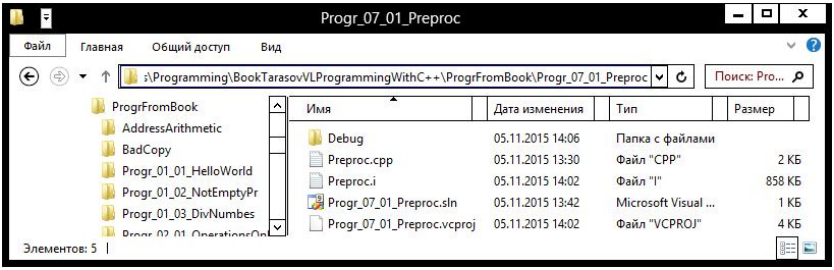

Рис. 7.2. Файл "i", созданный препроцессором

```
using namespace std;

void main()
{
    int c = 5, d = 7;
    if(((c) > (d) ? 1 : 0))
        cout << "\nc > d";
    else
        cout << "\nc <= d";


    cout << "\nSMALL is defined";




    cout << "\nSMALL is not defined";



    cout << "\nThe Name current file: " <<
"d:\\auser\\methodworks\\programming\\programmingwithc++\\progrfrombook\\pro
gr_07_01_preprocessorpossibilities\\progr_07_01_preproc.cpp";
    cout << "\nDate: " << "Nov  4 2015";
    cout << "\nNumber the current Line "<< 33;
    cout << "\nTime " << "18:16:30";




    double y = sin(3.14159265358979323846 / 3) / 2.71828182845904523536;
    cin.get();
}
```

Видно, что все комментарии и директивы препроцессора удалены, вместо макросов подставлены их значения.

Директива препроцессора #pragma once

Для обеспечения однократного включения заголовочных файлов в Visual C++ имеется директива:

```
#pragma once
```

которая автоматически вставляется в начало заголовочных файлов, включаемых в проект при работе в среде Visual Studio.

Программа 7.2. Препроцессор в Visual Studio

Создадим в среде Visual Studio непустой проект консольного приложения ShowPragma. Автоматически создаваемый файл stdafx.h имеет содержание:

```
// Файл stdafx.h

#pragma once

#include "targetver.h"

#include <stdio.h>
#include <tchar.h>
const int N = 100;
```

Как видим, здесь в начало файла автоматически вставлена директива #pragma once. Дополнительно добавлено определение константы N.

В автоматически создаваемый файл ShowPragma.cpp автоматически вставляется файл stdafx.h. Попробуем сделать это трижды.

```
// Файл ShowPragma.cpp

#include "stdafx.h"
#include "stdafx.h"
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

Несмотря на троекратное включение файла stdafx.h, при построении ошибки не возникает, так как директива #pragma once, которая есть в файле stdafx.h, обеспечивает его однократное включение. Закомментируем эту директиву в файле stdafx.h:

```
//#pragma once

#include "targetver.h"

#include <stdio.h>
```

```
#include <tchar.h>
const int N = 100;
```

Теперь при построении проекта в одной и той же строке возникают две одинаковые ошибки о переопределении константы `N` (рис.7.3). Глядя на определении этой константы в файле `stdafx.h`, трудно догадаться в чем состоит ошибка, так как, в действительности, ошибка вызвана повторным включение этого файла в файл `ShowPragma.cpp`.

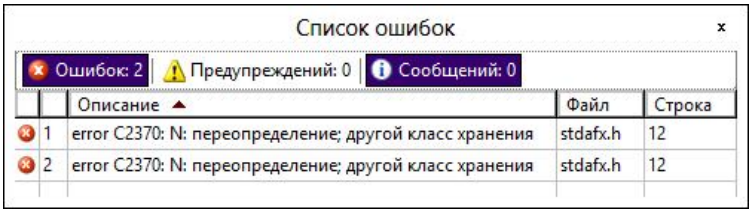

Рис. 7.3. Две одинаковые ошибки в одной и той же строке

Таким образом, препроцессор изменяет текст программ, что может привести к трудно обнаруживаемым ошибкам, поэтому следует с осторожностью использовать препроцессор.