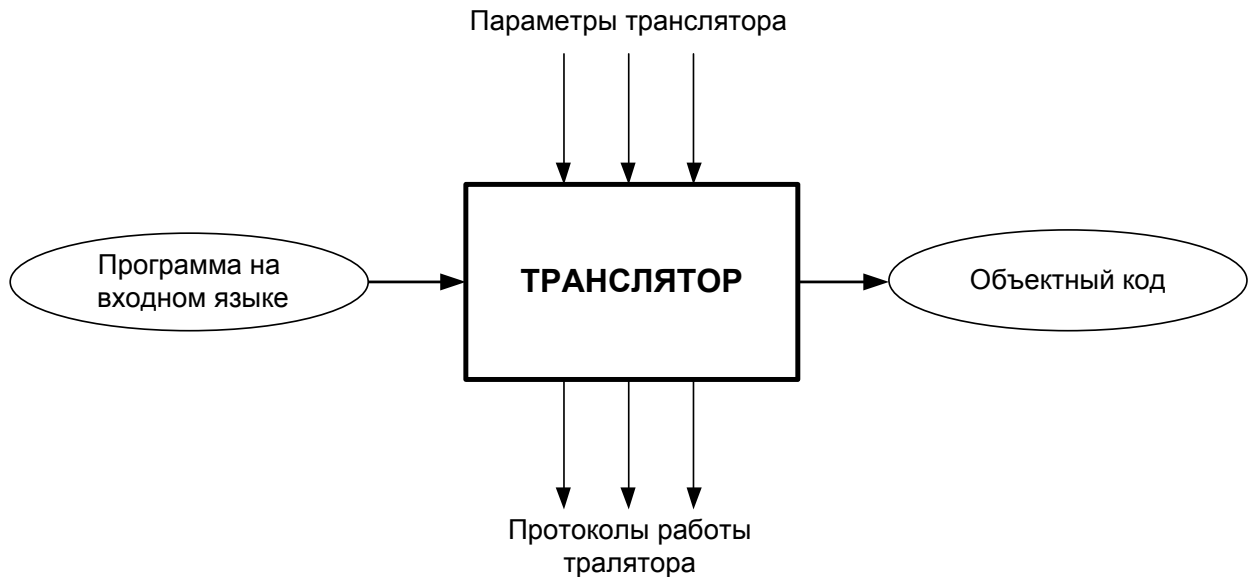


## Разбор лабораторной работы № 10

### 1. Введение. Первый этап разработки транслятора

Обобщенная структура транслятора:



Программа на входном языке (исходный код) – цепочка символов, составленная на исходном языке программирования.

Объектный код – код программы на целевом языке.

Объектный код:

- последовательность машинных команд;
- программа на языке ассемблера;
- программа на некотором другом языке (TypeScript → JavaScript).

Транслятор преобразует исходный код на одном языке программирования в исходный код на другом языке.

## 2. Задание лабораторной работы 10.

Требуется создать проект-приложение, структура которого представлена на рисунке 1.

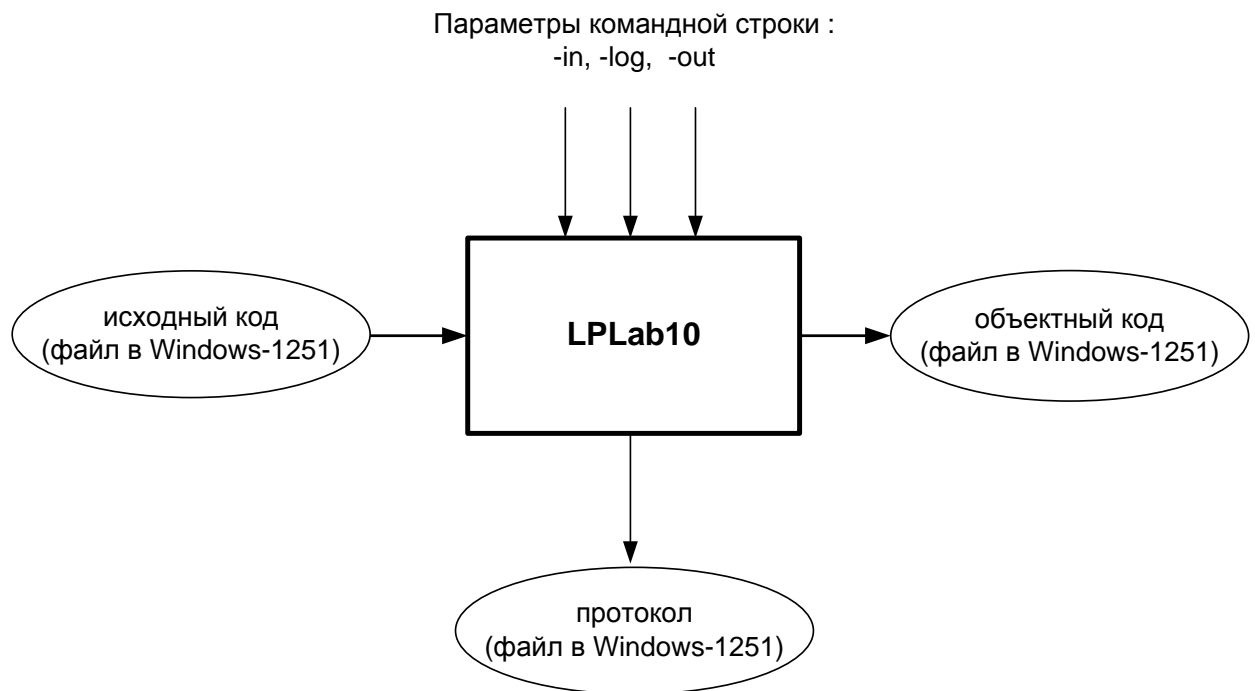


Рисунок 1 – Схема работы приложения **LPLab10**

Приложение принимает параметры, заданные ключами:  
-in:, -out:, -log:.

Запуск приложения LPLab10 из командной строки:

```
s\LP_Lab10x\Debug>  
s\LP_Lab10x\Debug>  
s\LP_Lab10x\Debug>LPLab10 -in:D:\infile.txt -out:D:\outfile.txt -log:D:\log.txt
```

Параметр **-in:** – обязательный – полное имя файла с исходным кодом.

Параметр **-out:** – необязательный – полное имя файла с объектным кодом. В том случае, если параметр -out не задан, то имя файла, образуется от имени файла с исходным кодом (**-in**) добавлением расширения **.out**.

Если задан параметр: **-in:D:\Folder1\infile.txt** и не задан параметр **-out**, то для файла с объектным кодом используется имя **D:\Folder1\infile.txt.out**.

Параметр **-log:** – необязательный – полное имя файла протокола. В том случае, если параметр -log не задан, то используется имя файла, образованное от имени файла с исходным кодом (**-in**) добавлением расширения **.log**.

Если задан параметр: **–in:D:\Folder1\infile.txt** и не задан параметр **–log**, то для файла протокола используется имя **D:\Folder1\infile.txt.log**.

### **Важно!**

Приложение LPLab10 предназначено для вызова в консоли.

Приложение **LPLab10**, *не должно формировать файл с объектным кодом.*

### **3. Назначение.**

Приложение **LPLab10** посимвольно считывает файл с исходным кодом в оперативную память. При считывании осуществляет проверку символов на допустимость.

В процессе обработки входных параметров или считывании файла с исходным кодом могут возникать ошибки, которые фиксируются в протоколе и/или выводятся на консоль.

### **4. Последовательность разработки приложения:**

- 1) функции для обработки ошибок;
- 2) функции для обработки входных параметров;
- 3) функции для ввода файла с исходным кодом;
- 4) функции для работы с протоколом.

### **5. Пространства имен**

Пространства имен и имена файлов с исходным кодом:

<b>Набор функций</b>	<b>Пространство имен (namespace)</b>	<b>Заголовочный файл (*.h)</b>	<b>Реализация (*.cpp)</b>
обработка ошибок	Error	Error.h	Error.cpp
обработка параметров	Parm	Parm.h	Parm.cpp
ввод исходного кода	In	In.h	In.cpp
работа с протоколом	Log	Log.h	Log.cpp

## 6. Обработка ошибок

Структура приложения LPLab10:

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

#include "Error.h" // обработка ошибок
#include "Parm.h" // обработка параметров
#include "Log.h" // работа с протоколом
#include "In.h" // ввод исходного файла

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    try
    {
        //обработка параметров
        //создание журнала
        //ввод исходного кода
    }
    catch(Error::ERROR e)
    {
        // запись информации об ошибке в протокол
        // или вывод на консоль (если протокол не создан)
    }
    return 0;
};
```

Разработать функции **geterror** и **geterrorin** по следующему описанию:

Наименование функции	Назначение
geterror	Используется в макросе <b>ERROR_THROW</b> . <b>Параметры:</b> <b>id</b> - код ошибки (int). <b>Выполняет:</b> проверяет допустимый диапазон <b>id</b> ; извлекает данные из таблицы ошибок и заносит данные в возвращаемую структуру <b>ERROR</b> . Если значение параметра <b>id</b> выходит за пределы допустимого диапазона ( $0 < id < \mathbf{ERROR\_MAX\_ENTRY}$ ), то формируется содержимое структуры <b>ERROR</b> , соответствующее ошибке с кодом 0. <b>Возврат:</b> заполненная структура <b>ERROR</b> .
geterrorin	Используется в макросе <b>ERROR_THROW_IN</b> . <b>Параметры:</b> <b>id</b> - код ошибки (int), <b>line</b> – номер строки (int, по умолчанию -1), <b>col</b> – позиция в строке (int, по умолчанию -1). <b>Выполняет:</b> проверяет допустимый диапазон <b>id</b> ; извлекает данные из таблицы ошибок и заносит данные в возвращаемую структуру <b>ERROR</b> . Если значение параметра <b>id</b> выходит за пределы допустимого диапазона ( $0 < id < \mathbf{ERROR\_MAX\_ENTRY}$ ), то формируется содержимое структуры <b>ERROR</b> соответствующее ошибки с кодом 0. <b>Возврат:</b> заполненная структура <b>ERROR</b> .

Пример программного кода, тестирующего функции **geterror** и **geterrorin**:

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

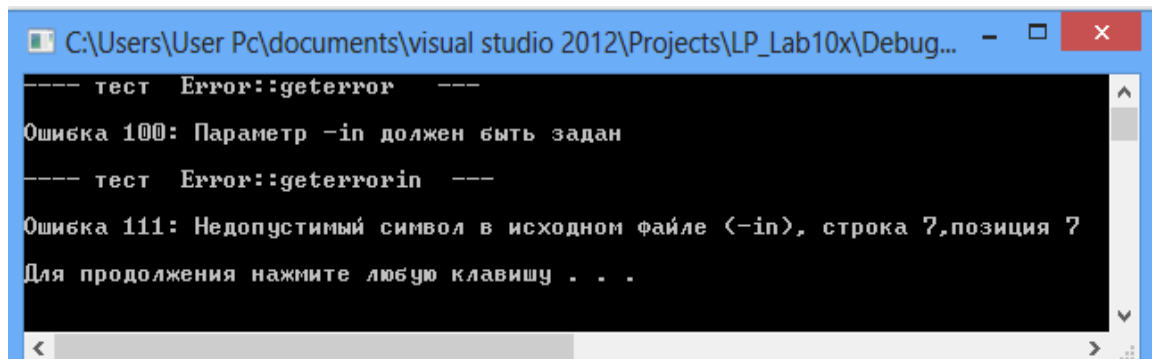
#include "Error.h" // обработка ошибок
#include "Parm.h" // обработка параметров
#include "Log.h" // работа с протоколом
#include "In.h" // ввод исходного файла

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    std::cout<<"---- тест Error::geterror ----" <<std::endl<<std::endl;
    try{ throw ERROR_THROW(100);}
    catch(Error::ERROR e)
    {
        std::cout<<"Ошибка " << e.id << ": " <<e.message<<std::endl<<std::endl;
    };

    std::cout<<"---- тест Error::geterrorin ----" <<std::endl<<std::endl;
    try{ throw ERROR_THROW_IN(111, 7, 7);}
    catch(Error::ERROR e )
    {
        std::cout<<"Ошибка " << e.id << ": " <<e.message
            <<" , строка " <<e.inext.line
            <<" ,позиция " <<e.inext.col<<std::endl<<std::endl;
    };

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

Пример выполнения теста функций **geterror** и **geterrorin**:



```
C:\Users\User Pc\documents\visual studio 2012\Projects\LP_Lab10x\Debug...
---- тест Error::geterror ----
Ошибка 100: Параметр -in должен быть задан
---- тест Error::geterrorin ----
Ошибка 111: Недопустимый символ в исходном файле (-in), строка 7, позиция 7
Для продолжения нажмите любую клавишу . . .
```

## Содержимое файла **Error.h**:

```
#pragma once
#define ERROR_THROW(id) Error::geterror(id);           // throw ERROR_THROW(id)
#define ERROR_THROW_IN(id, l, c) Error::geterrorin(id, l, c); // throw ERROR_THROW(id, строка, колонка)
#define ERROR_ENTRY(id, m) {id, m, {-1, -1}}           // элемент таблицы ошибок
#define ERROR_MAXSIZE_MESSAGE 200                     // максимальная длина сообщения об ошибке
#define ERROR_ENTRY_NODEF(id) ERROR_ENTRY(-id, "Неопределенная ошибка") // 1 неопределенный элемент таблицы ошибок
// ERROR_ENTRY_NODEF10(id) - 10 неопределенных элементов таблицы ошибок
#define ERROR_ENTRY_NODEF10(id) ERROR_ENTRY_NODEF(id+0), ERROR_ENTRY_NODEF(id+1), ERROR_ENTRY_NODEF(id+2), ERROR_ENTRY_NODEF(id+3), \
    ERROR_ENTRY_NODEF(id+4), ERROR_ENTRY_NODEF(id+5), ERROR_ENTRY_NODEF(id+6), ERROR_ENTRY_NODEF(id+7), \
    ERROR_ENTRY_NODEF(id+8), ERROR_ENTRY_NODEF(id+9)
// ERROR_ENTRY_NODEF100(id) - 100 неопределенных элементов таблицы ошибок
#define ERROR_ENTRY_NODEF100(id) ERROR_ENTRY_NODEF10(id+ 0), ERROR_ENTRY_NODEF10(id+10), ERROR_ENTRY_NODEF10(id+20), ERROR_ENTRY_NODEF10(id+30), \
    ERROR_ENTRY_NODEF10(id+40), ERROR_ENTRY_NODEF10(id+50), ERROR_ENTRY_NODEF10(id+60), ERROR_ENTRY_NODEF10(id+70), \
    ERROR_ENTRY_NODEF10(id+80), ERROR_ENTRY_NODEF10(id+90)
#define ERROR_MAX_ENTRY 1000                          // количество элементов в таблице ошибок

namespace Error
{
    struct ERROR // тип исключения для throw ERROR_THROW | ERROR_THROW_IN и catch(ERROR)
    {
        int id; // код ошибки
        char message[ERROR_MAXSIZE_MESSAGE]; // сообщение об ошибке
        struct IN // расширение для ошибок при обработке входных данных
        {
            short line; // номер строки (0, 1, 2, ...)
            short col; // номер позиции в строке (0, 1, 2, ...)
        } inext;
    };

    ERROR geterror(int id); // сформировать ERROR для ERROR_THROW
    ERROR geterrorin(int id, int line, int col); // сформировать ERROR для ERROR_THROW_IN
};
```

Содержимое файла **Error.cpp** (реализация функций **geterror** и **geterrorin** намерено скрыта):

```
#include "stdafx.h"
#include "Error.h"
namespace Error
{
    // серии ошибок:  0 - 99 - системные ошибки
    //                  100 - 109 - ошибки параметров
    //                  110 - 119 - ошибки открытия и чтения файлов
    ERROR errors[ERROR_MAX_ENTRY] = //таблица ошибок
    {
        ERROR_ENTRY(0, "Недопустимый код ошибки"),    // код ошибки вне диапазона 0 - ERROR_MAX_ENTRY
        ERROR_ENTRY(1, "Системный сбой"),
        ERROR_ENTRY_NODEF(2), ERROR_ENTRY_NODEF(3), ERROR_ENTRY_NODEF(4), ERROR_ENTRY_NODEF(5),
        ERROR_ENTRY_NODEF(6), ERROR_ENTRY_NODEF(7), ERROR_ENTRY_NODEF(8), ERROR_ENTRY_NODEF(9),
        ERROR_ENTRY_NODEF10(10), ERROR_ENTRY_NODEF10(20), ERROR_ENTRY_NODEF10(30), ERROR_ENTRY_NODEF10(40), ERROR_ENTRY_NODEF10(50),
        ERROR_ENTRY_NODEF10(60), ERROR_ENTRY_NODEF10(70), ERROR_ENTRY_NODEF10(80), ERROR_ENTRY_NODEF10(90),
        ERROR_ENTRY(100, "Параметр -in должен быть задан"),
        ERROR_ENTRY_NODEF(101), ERROR_ENTRY_NODEF(102), ERROR_ENTRY_NODEF(103),
        ERROR_ENTRY(104, "Превышена длина входного параметра"),
        ERROR_ENTRY_NODEF(105), ERROR_ENTRY_NODEF(106), ERROR_ENTRY_NODEF(107),
        ERROR_ENTRY_NODEF(108), ERROR_ENTRY_NODEF(109),
        ERROR_ENTRY(110, "Ошибка при открытии файла с исходным кодом (-in)"),
        ERROR_ENTRY(111, "Недопустимый символ в исходном файле (-in)"),
        ERROR_ENTRY(112, "Ошибка при создании файла протокола(-log)"),
        ERROR_ENTRY_NODEF(113), ERROR_ENTRY_NODEF(114), ERROR_ENTRY_NODEF(115),
        ERROR_ENTRY_NODEF(116), ERROR_ENTRY_NODEF(117), ERROR_ENTRY_NODEF(118), ERROR_ENTRY_NODEF(119),
        ERROR_ENTRY_NODEF10(120), ERROR_ENTRY_NODEF10(130), ERROR_ENTRY_NODEF10(140), ERROR_ENTRY_NODEF10(150),
        ERROR_ENTRY_NODEF10(160), ERROR_ENTRY_NODEF10(170), ERROR_ENTRY_NODEF10(180), ERROR_ENTRY_NODEF10(190),
        ERROR_ENTRY_NODEF100(200), ERROR_ENTRY_NODEF100(300), ERROR_ENTRY_NODEF100(400), ERROR_ENTRY_NODEF100(500),
        ERROR_ENTRY_NODEF100(600), ERROR_ENTRY_NODEF100(700), ERROR_ENTRY_NODEF100(800), ERROR_ENTRY_NODEF100(900)
    };
    ERROR geterror(int id){ ... }
    ERROR geterrorin(int id, int line = -1, int col = -1){ ... }
};
```

## 7. Обработка входных параметров

Содержимое файла **Parm.h**:

```
#pragma once
#define PARM_IN L"-in:"           // ключ для файла исходного кода
#define PARM_OUT L"-out:"        // ключ для файла объектного кода
#define PARM_LOG L"-log:"        // ключ для файла журнала
#define PARM_MAX_SIZE 300        // максимальная длина строки параметра
#define PARM_OUT_DEFAULT_EXT L".out" // расширение файла объектного кода по умолчанию
#define PARM_LOG_DEFAULT_EXT L".log" // расширение файла протокола по умолчанию

namespace Parm                    // обработка входных параметров
{
    struct PARM                   // входные параметры
    {
        wchar_t in[PARM_MAX_SIZE]; // -in: имя файла исходного кода
        wchar_t out[PARM_MAX_SIZE]; // -out: имя файла объектного кода
        wchar_t log[PARM_MAX_SIZE]; // -log: имя файла протокола
    };

    //int _tmain(int argc, _TCHAR* argv[])
    PARM getparm(int argc, _TCHAR* argv[]); // сформировать struct PARM на основе параметров функции main
};
```

Разработать функцию **getparm** по следующему описанию:

Наименование функции	Назначение
getparm	<p>Используется для записи значений входных параметров (<b>-in:</b>, <b>-out:</b>, <b>-log:</b>) в структуру <b>PARM</b>.</p> <p><b>Параметры:</b> <b>argc</b> – количество параметров (int, <math>\geq 1</math>), <b>argv</b> – массив указателей на нуль-терминальные строки со значениями параметров, (<b>_TCHAR*</b> – указатель на строку <b>wchar_t</b>)</p> <p><b>Выполняет:</b> проверяет наличие параметра <b>-in:</b>; если параметр не задан генерируется исключение (<b>ERROR_THROW</b>) с кодом ошибки <b>100</b>;</p> <p>если не задано значения <b>-out:</b> и <b>-log:</b>, то формирует значения по умолчанию (см п.5-8);</p> <p>проверяет длину строки каждого входного параметра; если длина строки превышает значение <b>PARM_MAX_SIZE</b> (рис.6), то генерируется исключение (<b>ERROR_THROW</b>) с кодом ошибки <b>104</b>;</p> <p><b>Возврат:</b> заполненная структура <b>PARM</b>.</p>

**Указание:** используйте функции **wscpy\_s**, **wcsncat\_s**, **wcslen**, **wcsstr**, **wcslen** стандартной библиотеки.



Пример программного кода, тестирующего функцию **getparm**:

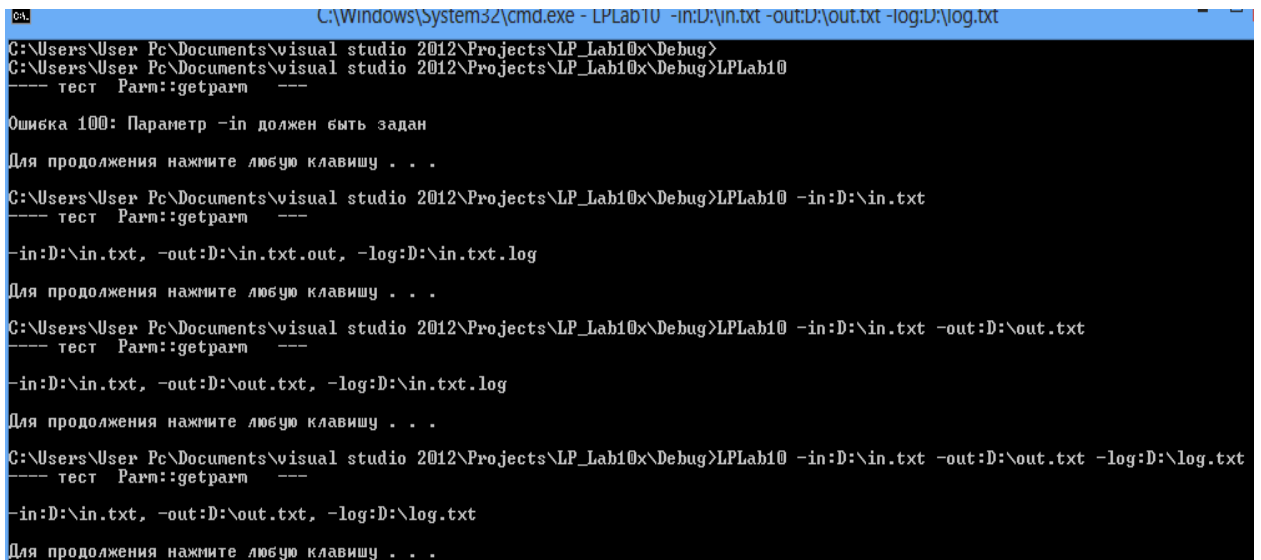
```
#include <wchar>

#include "Error.h"    // обработка ошибок
#include "Parm.h"     // обработка параметров
#include "Log.h"      // работа с протоколом
#include "In.h"       // ввод исходного файла

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    std::cout<<"---- тест Parm::getparm ----" <<std::endl<<std::endl;
    try
    {
        Parm::PARAM parm = Parm::getparm(argc, argv);
        std::wcout<<"-in:"<<parm.in<<" , -out:"<<parm.out<<" , -log:"<<parm.log<< std::endl <<std::endl;
    }
    catch(Error::ERROR e)
    {
        std::cout<<"Ошибка "<< e.id << " : "<<e.message<<std::endl<<std::endl;
    };

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

Пример выполнения теста функции **getparm**:



```
C:\Windows\System32\cmd.exe - LPLab10 -in:D:\in.txt -out:D:\out.txt -log:D:\log.txt
C:\Users\User Pc\Documents\visual studio 2012\Projects\LP_Lab10\Debug>
C:\Users\User Pc\Documents\visual studio 2012\Projects\LP_Lab10\Debug>LPLab10
---- тест Parm::getparm ----
Ошибка 100: Параметр -in должен быть задан
Для продолжения нажмите любую клавишу . . .
C:\Users\User Pc\Documents\visual studio 2012\Projects\LP_Lab10\Debug>LPLab10 -in:D:\in.txt
---- тест Parm::getparm ----
-in:D:\in.txt, -out:D:\in.txt.out, -log:D:\in.txt.log
Для продолжения нажмите любую клавишу . . .
C:\Users\User Pc\Documents\visual studio 2012\Projects\LP_Lab10\Debug>LPLab10 -in:D:\in.txt -out:D:\out.txt
---- тест Parm::getparm ----
-in:D:\in.txt, -out:D:\out.txt, -log:D:\in.txt.log
Для продолжения нажмите любую клавишу . . .
C:\Users\User Pc\Documents\visual studio 2012\Projects\LP_Lab10\Debug>LPLab10 -in:D:\in.txt -out:D:\out.txt -log:D:\log.txt
---- тест Parm::getparm ----
-in:D:\in.txt, -out:D:\out.txt, -log:D:\log.txt
Для продолжения нажмите любую клавишу . . .
```

Первый запуск – вызов без параметров → выводится ошибка 100.

Второй запуск – вызов с одним заданным параметром: **-in:D:\in.txt**.

И так далее.

## 8. Ввод файла исходного кода

Разработать функцию **getin** по следующему описанию:

Наименование функции	Назначение
getin	<p>Используется для ввода и проверки информации из файла с исходными кодами.</p> <p><b>Параметры:</b> infile – имя входного файла (<b>wchar_t*</b>)</p> <p><b>Выполняет:</b> посимвольно вводит данные из файла, заданного параметром; проверяет каждый символ на соответствие таблице проверки; подсчитывает и записывает в структуру <b>IN</b> количество введенных строк и символов, а также пропущенных символов; записывает в структуру <b>IN</b> таблицу проверки, символ может быть введен (обозначен в таблице <b>IN:T</b>), пропущен (<b>IN:I</b>), заменен (в таблице значение от <b>0</b> до <b>255</b>); если в таблице проверки символу соответствует значение <b>IN:F</b>, то генерируется исключение (<b>ERROR_THROW_IN</b>, код ошибки <b>111</b>), которое фиксирует в структуре <b>ERROR</b> номер строки (отсчет от 0) и номер позиции в строке (отсчет от 0), в котором обнаружен запрещенный символ; если возникает ошибка при открытии файла выходного потока, генерируется исключение (<b>ERROR_THROW</b>, код ошибки <b>110</b>).</p> <p><b>Возврат:</b> заполненная структура <b>IN</b>.</p>

**Указание:** используйте потоковый ввод **ifstream** для посимвольного ввода данных.

## Содержимое файла **In.h**:

```
#pragma once
#define IN_MAX_LEN_TEXT 1024*1024      // максимальный размер исходного кода = 1MB
#define IN_CODE_ENDL '\n'              // символ конца строки

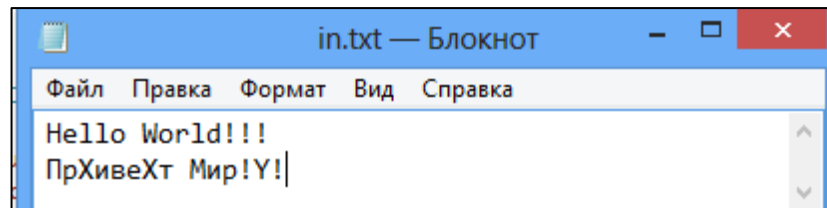
// таблица проверки входной информации, индекс = код (Windows-1251) символа
// значения IN::F - запрещенный символ, IN::T - разрешенный символ, IN::I - игнорировать (не вводить),
//      если 0 <= значение < 256 - то вводится данное значение
#define IN_CODE_TABLE {\
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::T, IN::F, IN::F, IN::I, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::T, IN::I, '!', IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::T, IN::F, IN::F, IN::T, \
    IN::F, IN::F, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::F, IN::F, IN::F, IN::F, IN::F, IN::T, IN::F, IN::F, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
    IN::T, IN::F, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, \
}

namespace In
{
    struct IN          // исходные данные
    {
        enum {T = 1024, F = 2048, I = 4096};    // T - допустимый символ, F - недопустимый, I - игнорировать, иначе заменить
        int size;          // размер исходного кода
        int lines;         // количество строк
        int ignor;         // количество проигнорированных символов
        unsigned char* text; // исходный код (Windows - 1251)
        int code [256];     // таблица проверки: T, F, I новое значение
    };
    IN getin(wchar_t infile[]); // ввести и проверить входной поток
};
```

## Тест1 функции **getin**

Проверочная таблица допускает ввод букв, входящих в строки **Hello World!** **Привет Мир** и символ \n (конец строки), игнорирует английскую букву **X** и символ с кодом **0x0d**, а также заменяет английскую букву **Y** на **!**.

Исходный файл:



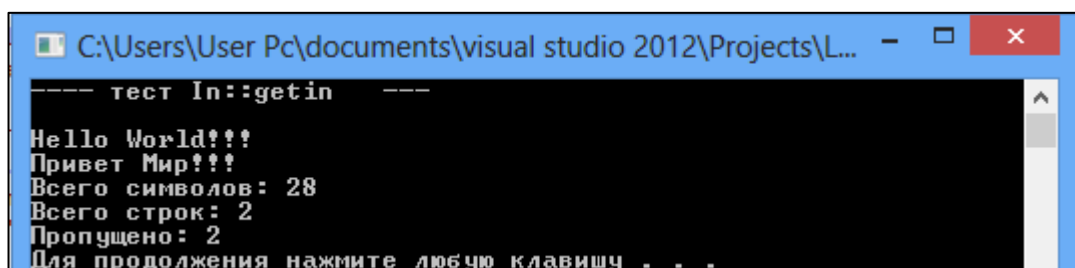
Программный код, тестирующий функцию **getin**:

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

#include "Error.h"    // обработка ошибок
#include "Parm.h"     // обработка параметров
#include "Log.h"      // работа с протоколом
#include "In.h"       // ввод исходного файла

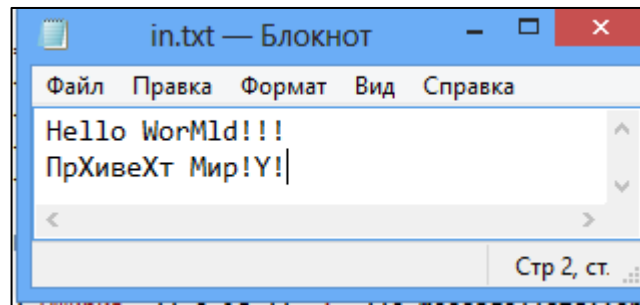
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    std::cout<<"---- тест In::getin  ---" <<std::endl<<std::endl;
    try
    {
        Parm::PARAM parm = Parm::getparm(argc, argv);
        In::IN in = In::getin(parm.in);
        std::cout<<in.text<<std::endl;
        std::cout<<"Всего символов: "<< in.size<<std::endl;
        std::cout<<"Всего строк: "<< in.lines<<std::endl;
        std::cout<<"Пропущено: "<< in.ignor<<std::endl;
    }
    catch(Error::ERROR e)
    {
        std::cout<<"Ошибка "<< e.id << ": "<<e.message<<std::endl<<std::endl;
    };
    system("pause");
    return 0;
};
```

Пример выполнения тестирования функции **getin**:



## Тест2 функции **getin**

Исходный файл (содержит не допустимый символ):



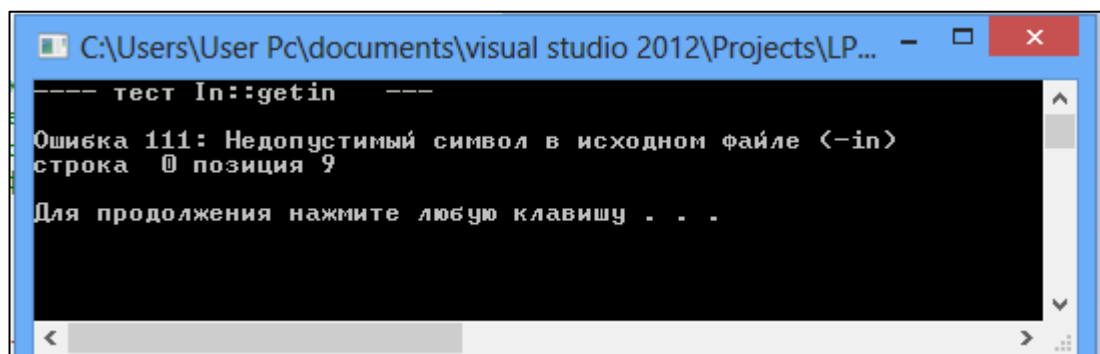
Программный код, тестирующий функцию **getin**:

```
#include <wchar>

#include "Error.h"    // обработка ошибок
#include "Parm.h"     // обработка параметров
#include "Log.h"      // работа с протоколом
#include "In.h"       // ввод исходного файла

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    std::cout<<"---- тест In::getin ----" <<std::endl<<std::endl;
    try
    {
        Parm::PARAM parm = Parm::getparm(argc, argv);
        In::IN in = In::getin(parm.in);
        std::cout<<in.text<<std::endl;
        std::cout<<"Всего символов: "<< in.size<<std::endl;
        std::cout<<"Всего строк: "<< in.lines<<std::endl;
        std::cout<<"Пропущено: "<< in.ignor<<std::endl;
    }
    catch(Error::ERROR e)
    {
        std::cout<<"Ошибка "<< e.id << ": "<<e.message<<std::endl;
        std::cout<<"строка "<< e.inext.line << " позиция "<<e.inext.col
            <<std::endl<<std::endl;
    }
    system("pause");
    return 0;
};
```

Пример выполнения тестирования функции **getin**:



## 9. Работа с протоколом

Содержимое файла **Log.h**:

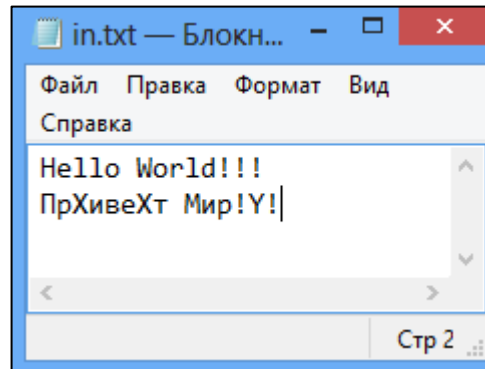
```
#pragma once
#include <fstream>
#include "In.h"
#include "Parm.h"
namespace Log // Работа с протоколом
{
    struct LOG // протокол
    {
        wchar_t logfile[PARM_MAX_SIZE]; // имя файла протокола
        std::ofstream * stream; // выходной поток протокола
    };

    static const LOG INITLOG = {L"", NULL}; // структура для начальной инициализации LOG
    LOG getlog(wchar_t logfile[]); // сформировать структуру LOG
    void WriteLine(LOG log, char* c, ...); // вывести в протокол конкатенацию строк
    void WriteLine(LOG log, wchar_t* c, ...); // вывести в протокол конкатенацию строк
    void WriteLog(LOG log); // вывести в протокол заголовок
    void WriteParm(LOG log, Parm::PARM parm); // вывести в протокол информацию о входных параметрах
    void WriteIn(LOG log, In::IN in); // вывести в протокол информацию о входном потоке
    void WriteError(LOG log, Error::ERROR error); // вывести в протокол информацию об ошибке
    void Close(LOG log); // закрыть протокол
};
```

Разработать функции, описанные в таблице:

Наименование функции	Назначение
getlog	Используется для создания и открытия потокового вывода протокола. <b>Параметры:</b> logfile – имя входного файла ( <b>wchar_t*</b> ) <b>Выполняет:</b> открывает (создает) выходной поток; если поток не создан, генерируется исключение ( <b>ERROR_THROW</b> , код ошибки <b>112</b> ); записывает данные в структуру <b>LOG</b> . <b>Возврат:</b> заполненная структура <b>LOG</b> . <b>Указание:</b> примените потоковый вывод <b>ofsream</b>
WriteLine (две функции)	Используется для вывода одной строки в протокол <b>Параметры:</b> структура <b>LOG</b> , переменное число параметров типа <b>char*</b> , последний параметр должен быть пустой строкой. <b>Параметры:</b> структура <b>LOG</b> , переменное число параметров типа <b>wchar_t*</b> , последний параметр должен быть пустой строкой. <b>Выполняет:</b> осуществляет конкатенацию всех строк заданных параметрами, формирует строку и выводит ее в протокол. <b>Возврат:</b> функция ничего не возвращает <b>Указание:</b> для преобразования строки <b>wchar_t*</b> в строку <b>char*</b> примените функцию <b>wstombs</b>
WriteLog	Используется для вывода заголовка протокола <b>Параметры:</b> структура <b>LOG</b> . <b>Выполняет:</b> выводит строку заголовка в протокол. <b>Возврат:</b> функция ничего не возвращает. <b>Указание:</b> для получения текущей даты и времени в формате строки используйте функции <b>time</b> , <b>localtime_s</b> и <b>strftime</b> .
WriteParm	Используется для вывода в протокол информации о входных параметрах <b>Параметры:</b> структура <b>LOG</b> и структура <b>PARM</b> . <b>Выполняет:</b> выводит в протокол информацию о параметрах. <b>Возврат:</b> функция ничего не возвращает.
WriteIn	Используется для вывода в протокол информации о входных данных. <b>Параметры:</b> структура <b>LOG</b> и структура <b>IN</b> . <b>Выполняет:</b> выводит в протокол информацию о входных данных. <b>Возврат:</b> функция ничего не возвращает.
WriteError	Используется для вывода в протокол или на консоль информации об ошибке. <b>Параметры:</b> структура <b>LOG</b> и структура <b>IN</b> . <b>Выполняет:</b> выводит в протокол информацию об ошибке; если протокол не открыт, выводит информацию на консоль. <b>Возврат:</b> функция ничего не возвращает.
Close	Используется для закрытия выходного потока протокола. <b>Параметры:</b> структура <b>LOG</b> . <b>Выполняет:</b> закрывает выходной поток. <b>Возврат:</b> функция ничего не возвращает.

Пример исходного файла:



Программный код, тестирующий функции:

```
#include <iostream>
#include <locale>
#include <wchar>

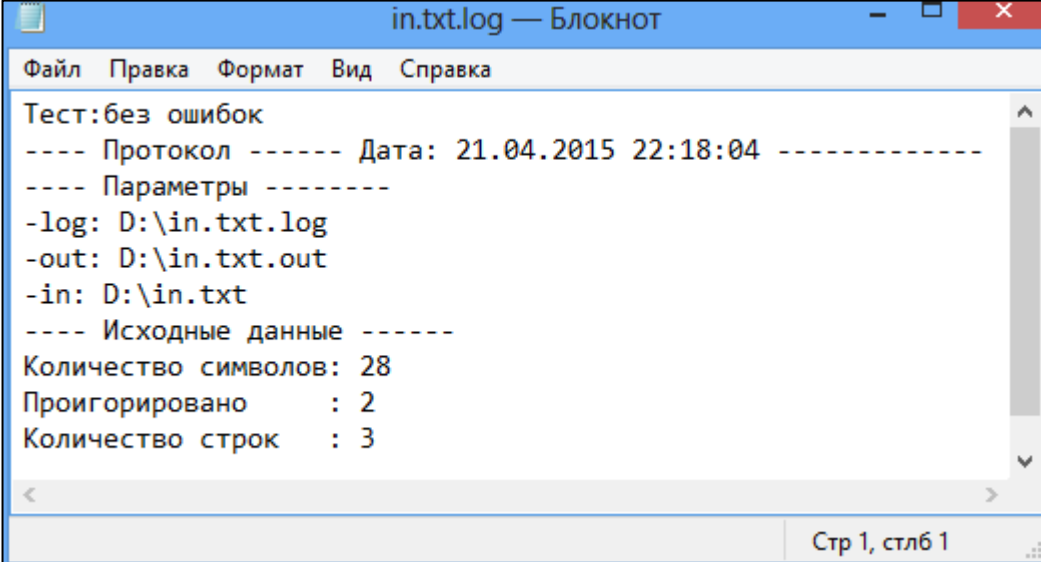
#include "Error.h"    // обработка ошибок
#include "Parm.h"     // обработка параметров
#include "Log.h"      // работа с протоколом
#include "In.h"       // ввод исходного файла

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    Log::LOG log = Log::INITLOG;
    try
    {
        Parm::PARM parm = Parm::getparm(argc, argv);
        log = Log::getlog(parm.log);
        Log::WriteLine(log, "Тест:", "без ошибок ", "");
        Log::WriteLog(log);
        Log::WriteParm(log, parm);
        In::IN in = In::getin(parm.in);
        Log::WriteIn(log, in);
        Log::Close(log);
    }
    catch(Error::ERROR e)
    {
        Log::WriteError(log, e);
    };

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

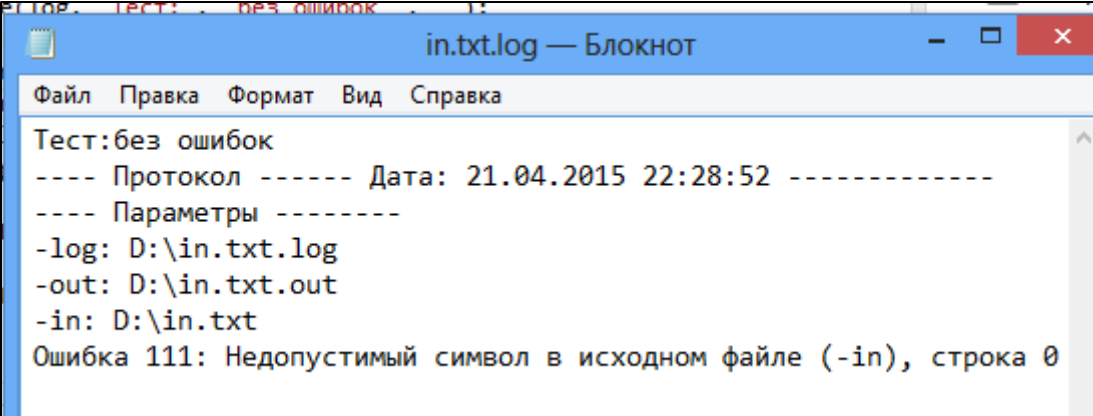


Протокол выполнения без ошибок:



```
in.txt.log — Блокнот
Файл  Правка  Формат  Вид  Справка
Тест:без ошибок
---- Протокол ----- Дата: 21.04.2015 22:18:04 -----
---- Параметры -----
-log: D:\in.txt.log
-out: D:\in.txt.out
-in: D:\in.txt
---- Исходные данные -----
Количество символов: 28
Проигорировано      : 2
Количество строк    : 3
Стр 1, стлб 1
```

Протокол выполнения с ошибками:



```
in.txt.log — Блокнот
Файл  Правка  Формат  Вид  Справка
Тест:без ошибок
---- Протокол ----- Дата: 21.04.2015 22:28:52 -----
---- Параметры -----
-log: D:\in.txt.log
-out: D:\in.txt.out
-in: D:\in.txt
Ошибка 111: Недопустимый символ в исходном файле (-in), строка 0
```