

Лабораторная работа 14

Конструирование программного обеспечения

Первый этап разработки транслятора (обработка ошибок, обработка параметров, ввод и проверка входных данных, протоколирование)

Цель работы: разработать проект-приложение, предназначенное для вызова в консоли (из командной строки разработчика).

Назначение приложения: приложение принимает входной файл в кодировке Windows 1251. При посимвольном считывании осуществляется проверка символов на допустимость: каждый символ проверяется на соответствие таблице проверки. В процессе обработки входных параметров или считывании файла с исходным кодом могут возникать ошибки, которые фиксируются в протоколе.

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

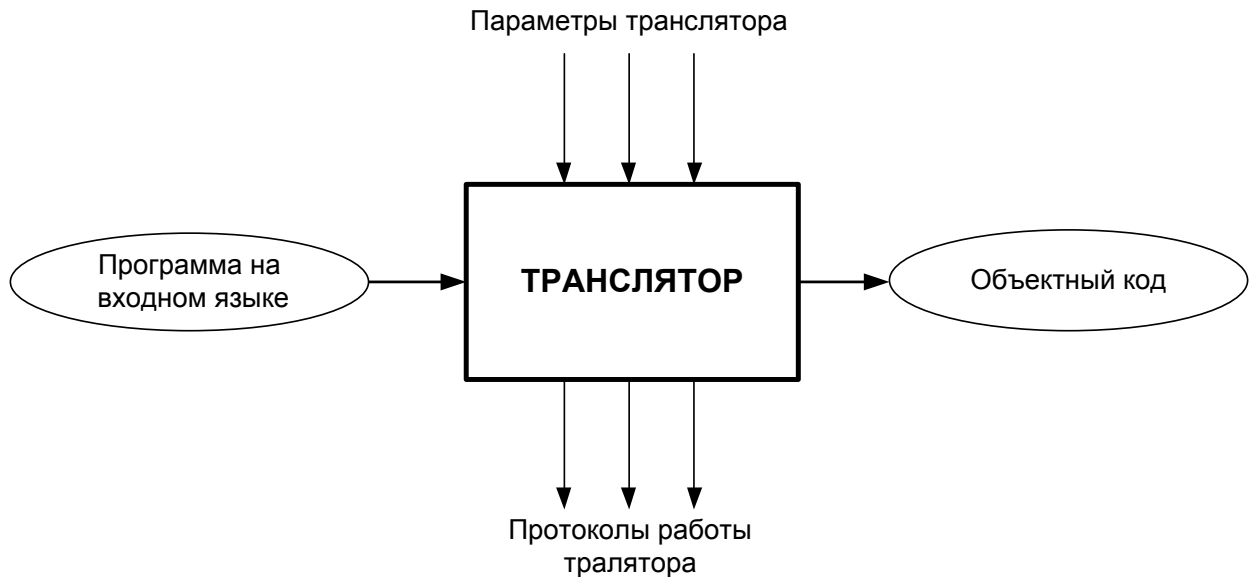
Приложение является многофайловым проектом и обеспечивает выполнение следующих функций:

- обработки и анализ входных параметров;
- ввод файла исходных кодов;
- проверка входных данных на допустимость;
- обработка ошибок;
- обеспечение работы с протоколом.

Для проверки работоспособности приложения в целом и функций, входящих в него, необходимо выполнить тестирование.

I. Введение. Первый этап разработки транслятора

Обобщенная структура транслятора:



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

Объектный код – эквивалентный код программы на целевом языке.

Объектный код:

- последовательность машинных команд;
- программа на языке ассемблера;
- программа на некотором другом языке (например, трансляция кода на TypeScript в код на языке программирования JavaScript).

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

II. Задание:

1. Используйте материал лекции № 11.
2. Создайте проект-приложение с именем **SE_Lab14**.
3. Ознакомьтесь с рисунком 1, демонстрирующим схему работы приложения SE_Lab14.

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

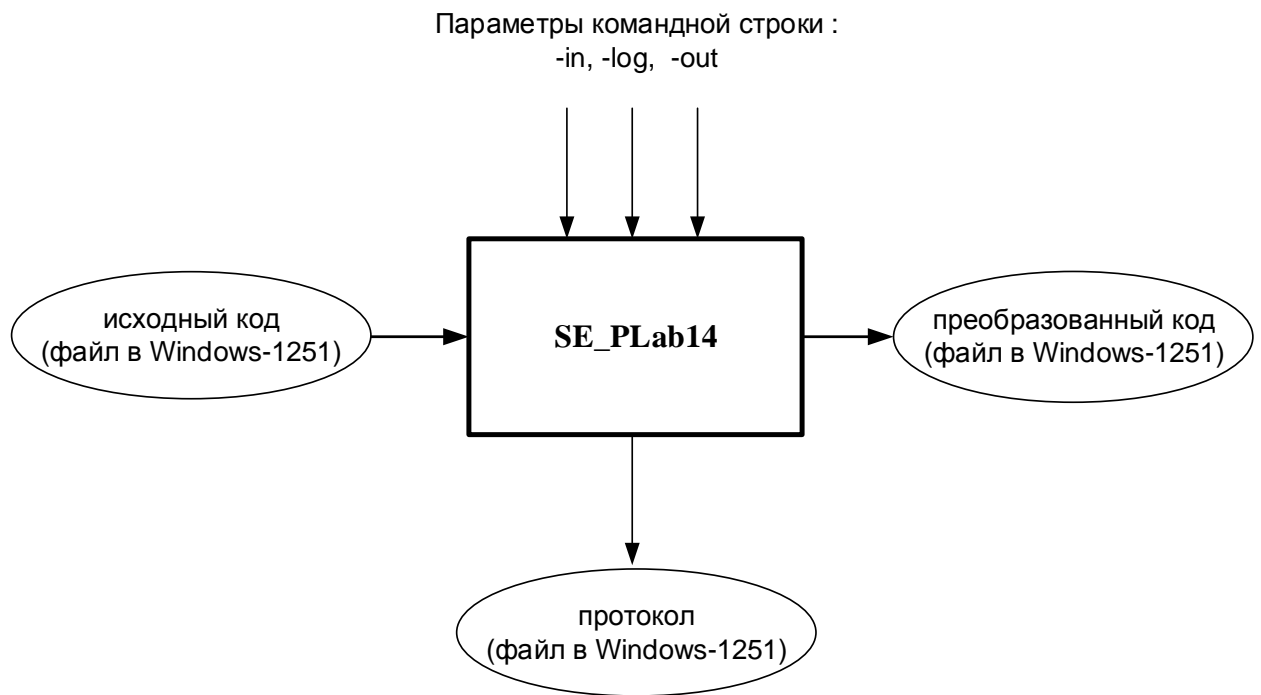


Рисунок 1 – Схема приложения **SE_Lab14**

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

Запуск приложения SE_Lab14 из командной строки разработчика:

```
Содержимое папки D:\Ade1\KPO_Lec\SE_PLab15\Debug
20.05.2022 23:21 <DIR> .
20.05.2022 23:21 <DIR> ..
20.05.2022 23:21      391 680 SE_PLab15.exe
20.05.2022 23:21      893 000 SE_PLab15.ilk
20.05.2022 23:21      749 568 SE_PLab15.pdb
                3 файлов      2 034 248 байт
                2 папок      1 052 368 896 байт свободно
D:\Ade1\KPO_Lec\SE_PLab15\Debug>SE_PLab15.exe -in:D:\in.txt -out:D:\out.txt -log:D:\log.txt
```

5. Параметр **-in:** – *обязательный* – это полное имя файла с исходным кодом.
6. Параметр **-out:** – *необязательный* – полное имя файла с преобразованным кодом. В том случае, если параметр **-out** не задан,

то имя файла, образуется от имени файла с исходным кодом, заданным в параметре **-in**: путем добавления расширения **.out**.

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

7. 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 SE_Lab14 предназначено для вызова в консоли.

III. Назначение.

Приложение **SE_Lab14** посимвольно считывает файл с исходным кодом в оперативную память. При считывании осуществляет проверку символов на допустимость.

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

IV. 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.

V. Пространства имен

Пространства имен и имена файлов с исходным кодом:

Набор функций	Пространство имен (namespace)	Заголовочный файл (*.h)	Реализация (*.cpp)
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

VI. Обработка ошибок

Структура приложения SE_Lab14:

```
#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;
};
```

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

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

	<p>Если значение параметра id выходит за пределы допустимого диапазона ($0 < id < \text{ERROR_MAX_ENTRY}$), то формируется содержимое структуры ERROR соответствующее ошибки с кодом 0.</p> <p>Возврат: заполненная структура ERROR.</p>
--	---

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;
};
```

2. Выполните тест и убедитесь в работоспособности функций **geterror** и **geterrorin** и макросов **ERROR_THROW** и **ERROR_THROW_IN**.

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

```
D:\Adel\KPO_Lec\SE_PLab15\Debug\SE_PLab15.exe
---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
```

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){ ... }
};
```


VII. 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
};
```

1. Разработать функцию **getparm** по следующему описанию:

Наименование функции	Назначение
getparm	<p>Используется для записи значений входных параметров (-in:, -out:, -log:) в структуру PARM.</p> <p>Параметры: argc – количество параметров (int, >=1), argv – массив указателей на нуль-терминальные строки со значениями параметров, (_TCHAR* – указатель на строку wchar_t)</p> <p>Выполняет: проверяет наличие параметра -in:; если параметр не задан генерируется исключение (ERROR_THROW) с кодом ошибки 100;</p> <p>если не задано значения -out: и -log:, то формирует значения по умолчанию (см п.2);</p> <p>проверяет длину строки каждого входного параметра; если длина строки превышает значение PARM_MAX_SIZE, то генерируется исключение (ERROR_THROW) с кодом ошибки 104;</p> <p>Возврат: заполненная структура PARM.</p>

Указание: используйте функции **wcsncpy_s**, **wcsnecat_s**, **wcslen**, **wcsstr**, **wcslen** стандартной библиотеки для широких строк.

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*:

```
D:\Adel\KP0_Lec\SE_PLab15\Debug>SE_PLab15
--тест Parm::getparm ---

Ошибка 100 : Параметр -in должен быть задан

Для продолжения нажмите любую клавишу . . .
```

Первый запуск – вызов без параметров → выводится ошибка 100.

```
D:\Adel\KP0_Lec\SE_PLab15\Debug>SE_PLab15 -in:in.txt
--тест Parm::getparm ---

-in:in.txt, -out:in.txt.out, log:in.txt.log

Для продолжения нажмите любую клавишу . . .
```

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

```
D:\Adel\KP0_Lec\SE_PLab15\Debug>SE_PLab15 -in:in.txt -out:out.txt
--тест Parm::getparm ---

-in:in.txt, -out:out.txt, log:in.txt.log

Для продолжения нажмите любую клавишу . . .
```

```
D:\Ade1\KP0_Lec\SE_PLab15\Debug>SE_PLab15 -in:in.txt -out:out.txt -log:log.txt
--тест Parm::getparm ---

-in:in.txt, -out:out.txt, log:log.txt

Для продолжения нажмите любую клавишу . . .
```

И так далее.

2. Выполните тест и убедитесь в работоспособности функций **getparm**.

VIII. Ввод файла исходного кода

1. Разработать функцию **getin** по следующему описанию:

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

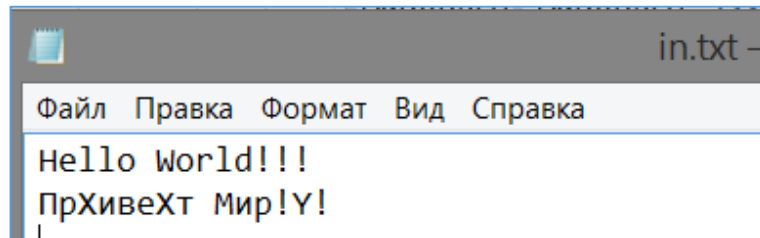
// таблица проверки входной информации, индекс = код (Windows-1251) символа
// значения IN::F - запрещенный символ, IN::T - разрешенный символ, IN::I - игнорировать (не вводить),
//      если 0 <= значение < 256 - то вводится данное значение
#define IN_CODE_TABLE {\
    IN::F, IN::F, IN::F, IN::F, IN::F, 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::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::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::T, 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 **!**.

2. 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::PARM 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*:

```
D:\Adel\KPO_Lec\SE_PLab15\Debug>SE_PLab15.exe -in:in.txt
---тест In:getin---

Hello World!!!
Привет Мир!!!
Всего символов: 28
Всего строк: 1
Пропущено: 2
Для продолжения нажмите любую клавишу . . .
```

Тест2 функции *getin*

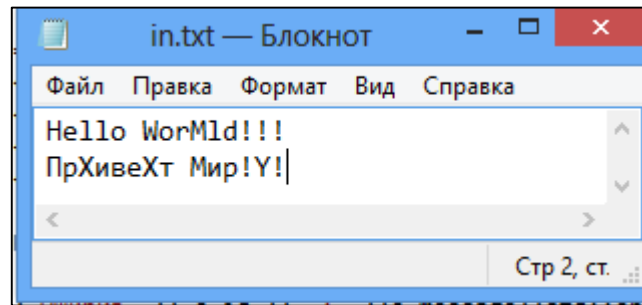
Программный код, тестирующий функцию *getin*:

```
#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;
};
```

3. Исходный файл (содержит недопустимый символ):



Пример выполнения тестирования функции **getin**:

```
D:\Adel\KPO_Lec\SE_PLab15\Debug>SE_PLab15.exe -in:in.txt
---тест In:getin---

Ошибка 111: Недопустимый символ в исходном файле (-in)

Строка 1 позиция 10

Для продолжения нажмите любую клавишу . . .
```

4. Выполните тесты и убедитесь в работоспособности функций **getin**.
5. Сформируйте проверочную таблицу таким образом, чтобы допускался ввод только букв, входящих в вашу фамилию и имя на русском и английском языках, а также цифр входящих в год вашего рождения. Кроме того, буква А (русская буква) должна заменяться на разрешенный символ – (минус), буква Х (английская буква должна игнорироваться).
6. Протестируйте функцию **getin** (табл.4) на собственных данных.

IX. Работа с протоколом

Содержимое файла **Log.h**:

```
#pragma once
#include <fstream>
#include "In.h"
#include "Parm.h"
#include "Error.h"

namespace Log          // Работа с протоколом
{
    struct LOG          // протокол
    {
        wchar_t logfile[PARM_MAX_SIZE]; // имя файла протокола
        std::ofstream* stream;          // выходной поток протокола
    };

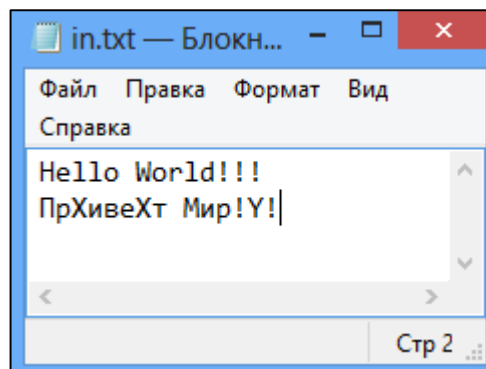
    static const LOG INITLOG { L"", NULL }; // структура для начальной инициализации LOG
    LOG getlog(wchar_t logfile[]);          // сформировать структуру LOG
    void WriteLine(LOG log, char* c, ...);  // вывести в протокол конкатенацию строк
    void WriteLine(LOG log, wchar_t* c, ...); // вывести 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);
};
```

1. Разработать функции, описанные в таблице:

Наименование функции	Назначение
getlog	Используется для создания и открытия потокового вывода протокола. Параметры: logfile – имя входного файла (wchar_t*) Выполняет: открывает (создает) выходной поток; если поток не создан, генерируется исключение (ERROR_THROW , код ошибки 112); записывает данные в структуру LOG . Возврат: заполненная структура LOG . Указание: примените потоковый вывод ofstream
WriteLine (две функции)	Используется для вывода одной строки в протокол Параметры: структура LOG , переменное число параметров типа char* , последний параметр должен быть пустой строкой. Параметры: структура LOG , переменное число параметров типа wchar_t* , последний параметр должен быть пустой строкой. Выполняет: осуществляет конкатенацию всех строк, заданных параметрами, формирует строку и выводит ее в протокол. Возврат: функция ничего не возвращает Указание: для преобразования строки wchar_t* в строку char* примените функцию wstombs
WriteLog	Используется для вывода заголовка протокола Параметры: структура LOG . Выполняет: выводит строку заголовка в протокол. Возврат: функция ничего не возвращает.

	Указание: для получения текущей даты и времени в формате строки используйте функции time , localtime_s и strftime .
WriteParm	Используется для вывода в протокол информации о входных параметрах Параметры: структура LOG и структура PARM . Выполняет: выводит в протокол информацию о параметрах. Возврат: функция ничего не возвращает.
WriteIn	Используется для вывода в протокол информации о входных данных. Параметры: структура LOG и структура IN . Выполняет: выводит в протокол информацию о входных данных. Возврат: функция ничего не возвращает.
WriteError	Используется для вывода в протокол или на консоль информации об ошибке. Параметры: структура LOG и структура IN . Выполняет: выводит в протокол информацию об ошибке; если протокол не открыт, выводит информацию на консоль. Возврат: функция ничего не возвращает.
Close	Используется для закрытия выходного потока протокола. Параметры: структура LOG . Выполняет: закрывает выходной поток. Возврат: функция ничего не возвращает.

Пример исходного файла:



Программный код, тестирующий функции:

```

Log::LOG log = Log::INITLOG;
try
{
    Parm::PARM parm = Parm::getparm(argc, argv);
    log = Log::getlog(parm.log);
    Log::Writeline(log, (char*)"Тест:", (char*)" без ошибок \n", "");
    Log::Writeline(log, (wchar_t*)L"Тест:", (wchar_t*)L" без ошибок \n", L "");
    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);
};

```

2. Запустите приложение **SE_Lab14** из командной строки разработчика и убедитесь в его работоспособности. Выполните все тесты и получите протоколы выполнения.

Запуск приложения из командной строки разработчика:

```
Содержимое папки D:\Ade1\KP0_Lec\SE_PLab15\Debug
21.05.2022 00:40 <DIR> .
21.05.2022 00:40 <DIR> ..
21.05.2022 00:20      32 in.txt
21.05.2022 00:36     248 in.txt.log
21.05.2022 00:35    378a368 SE_PLab15.exe
21.05.2022 00:35    824a540 SE_PLab15.ilc
21.05.2022 00:35    724a992 SE_PLab15.pdb
                5 файлов      1a928a180 байт
                2 папок      1a051a983a872 байт свободно

D:\Ade1\KP0_Lec\SE_PLab15\Debug>SE_PLab15 -in:in.txt
Для продолжения нажмите любую клавишу . . .
```

Протокол выполнения без ошибок:

```
in.txt.log — Блокнот
Файл  Правка  Формат  Вид  Справка
Тест: без ошибок
Тест: без ошибок
---- Протокол ----- 21.05.2022 00:42:56 -----
---- Параметры -----
-log: in.txt.log
-out: in.txt.out
-in: in.txt
---- Исходные данные -----
Количество символов: 29
Проигнорировано      : 2
Количество строк     : 2
```

Протокол выполнения с ошибками:

```
in.txt.log — Блокнот
Файл  Правка  Формат  Вид  Справка
Тест: без ошибок
Тест: без ошибок
---- Протокол ----- 21.05.2022 00:47:08 -----
---- Параметры -----
-log: in.txt.log
-out: in.txt.out
-in: in.txt
Ошибка 111: Недопустимый символ в исходном файле (-in), строка 0, позиция 11
```