Лабораторная работа №2. Модульная организация программ. Знакомство со стандартной библиотекой шаблонов языка C++

Модульная организация программ

Рано или поздно размер программы начинает превышать несколько сотен строк, а сама программа начинает содержать десятки функций, при этом поиск нужной функции становится затрудненным, и у программиста возникает естественное желание сгруппировать функции программы по их назначению.

При любом внесении изменений в единственный исходный файл проекта потребуется его полная перекомпиляция, что при большом размере файла замедлит не только работу редактора, но и увеличит время сборки программы.

Кроме того, код одних и тех же функций может совместно использоваться несколькими программах, и у программиста возникает необходимость сопровождения этого совместно используемого кода. Например, при исправлении ошибки в одной из функций программы, логично исправить ее и в остальных программах.

Упомянутые выше проблемы поможет решить вынесение логически связанного или совместно используемого кода в отдельные файлы.

Объявление и определение функции

Рассмотрим некоторую функцию Foo, выполняющую произвольные действия:

```
int Foo(int bar)
```

```
{
    // тело функции
    …
}
```

Строка **int Foo(int bar)** является **заголовком** функции Foo, а следующие за ним фигурные скобки с их внутренним содержимым – **телом** функции Foo. Заголовок функции содержит информацию о параметрах функции и типе ее возвращаемого значения, а тело функции – ее **реализацию**.

Заголовок функции совместно с ее телом образуют **определение** функции, а заголовок функции, следом за которым следует точка с запятой – ее **объявление**:

```
int Foo(int bar); // объявление функции Foo

int Foo(int bar) // определение функции Foo
{
    // тело функции
    …
}
```

Заметим, что внутри определения функции уже содержится ее объявление.

Для вызова функции необходимо написать ее имя, а следом за ней в фигурных скобках - значения передаваемых аргументов:

```
Foo(10);
int j = Foo(5);
int k = Foo(j + 10) + Foo(j);
```

Обязательное условие при этом: функция должна быть **объявлена** до своего первого использования. Заметьте, что для вызова функции (ее использования) достаточно одного лишь ее объявления, а тело функции знать не обязательно. Для того чтобы скомпилировать код функции компилятору, напротив, нужно знать ее **определение** целиком. Такое разделение позволяет написать функции, которые вызывают друг друга.

Например, следующая программа компилироваться не будет:

```
void Foo()
{
    …
    Bar();  // Ошибка – в данном месте программы функция Bar не объявлена
    …
}

void Bar()
{
    …
    Foo();  // Здесь все нормально, т.к. функция была объявлена выше
    …
}
```

Для исправления ошибки необходимо объявить функцию Bar() до тела функции Foo().

```
void Bar();

void Foo()
{
    …
```

```
    Bar();  // Теперь все нормально – функция Bar объявлена до ее использования
    …
}

void Bar()
{
    …
    Foo();
    …
}
```

В языке Си при совместном использовании объявления функции с ее определением следует помнить о том, что тип возвращаемого значения, количество и типы аргументов (но не обязательно их имена) в определении функции и в ее объявлении должны совпадать.

Язык Си++ в отличие от языка Си допускает **перегрузку** функций, при которой возможно существование различных функций с одним и тем же именем, но различным количеством и/или типами аргументов (но не типами возвращаемых значений). При несовпадении количества и/или типов аргументов между объявлением и определением функции они будут считаться разными функциями.

Мы знаем, что в определении функции уже содержится ее объявление. А что же произойдет, если некоторая функция будет лишь объявлена, но не определена? В этом случае компиляция файла завершится успешно, а вот компоновка программы завершится неудачно, т.к. для создания программы не будет хватать кода одной из объявленных функций. Существует лишь одно исключение: если функция, которая была объявлена, но не определена, в программе не используется, то компоновка программы завершится успешно. Пример:

```
// функция Foo не определена и используется в программе (см. функцию main),
// поэтому при компоновке программы будет ошибка
void Foo();

// функция Bar в программе не используется, поэтому отсутствие ее определения
// не влияет на возможность компоновки программы.
char Bar();
int main(int argc, char * argv[])
{
    Foo();
}
```

В других языках программирования, например, в C#, Java, ActionScript, JavaScript, PHP не требуется предварительное объявление функции и объявления функций, как таковые, в них отсутствуют.

Заголовочные файлы

Компиляторы языков Си и Си++ являются компиляторами с раздельной компиляцией исходного кода, различные **единицы компиляции** в них компилируются независимо друг от друга. В частности, это позволяет поместить объявления и определения функций в раздельных файлах. Файлы, содержащие только объявления функций (их заголовки), а также определяемые пользователем типы данных называются **заголовочными файлами**[1].

[1] Заголовочным файлом является обычный текстовый файл с одним из следующих расширений[1]: .h, .hpp, .hxx. Вам ничто не мешает использовать другие расширения для заголовочных файлов, или вообще использовать файлы без расширения, как, например, файлы стандартной библиотеки шаблонов STL языка C++. Однако если Вы хотите, чтобы Ваш код был понятен другим программистам, следует использовать единый подход к именованию файлов.

Вы уже знакомы с директивой **#include**, используемой для подключения заголовочных файлов. Рассмотрим подробнее ее использование для подключения собственных заголовочных файлов.

Рассмотрим функцию Sqr(), выполняющую возведение вещественного числа в квадрат.

```
double Sqr(double arg)
{
    return arg * arg;
}
```

Поместим ее объявление в заголовочном файле **algebra.h**:

```
double Sqr(double arg);
```

а определение – в файл **algebra.cpp**:

```
#include "algebra.h"
double Sqr(double arg)
{
    return arg * arg;
}
```

Обратите внимание на наличие директивы **#include "algebra.h"** в файле algebra.cpp. В ряде случае можно обойтись и без нее, однако, если функции файла algebra.cpp могут вызывать друг друга, то желательно в начале файла algebra.cpp подключить файл algebra.h с объявлениями всех функций, чтобы порядок определения функций внутри algebra.cpp не имел значения. Выполнить данное подключение следует также в том случае, когда в файле algebra.cpp используются типы данных, объявленные в algebra.h.

Для использования функции sqr в основном файле нашей программы (например, в main.cpp) необходимо подключить заголовочный файл main.h при помощи директивы **#include**:

```
#include <stdio.h>
#include "algebra.h"
int main(int argc, char * argv[])
{
    printf("Sqrt(3)=%f\n", Sqr(3));
}
```

Весьма вероятно, что при компоновке данной программы возникнет ошибка:

```
error LNK2019: unresolved external symbol "double __cdecl Sqr(double)" (?Sqr@@YANN@Z)
referenced in function _main
```

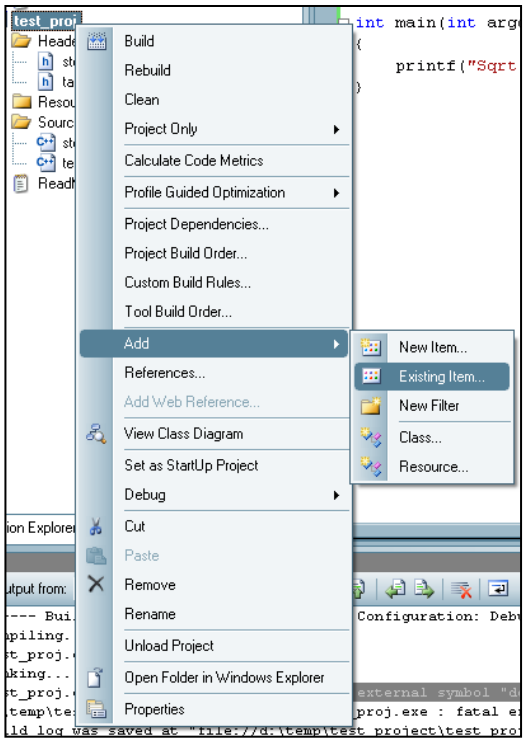
Причина ошибки заключается в том, что в проект изначально был добавлен лишь основной файл main.cpp, а файлы algebra.h и algebra.cpp – нет. В проекте содержалась лишь одна единица компиляции – файл main.cpp. Объявление функции Sqr() было включено в него при помощи директивы **#include**, а само определение этой функции – скомпилировано не было, т.к. содержится в файле algebra.cpp, не входящем в состав проекта.

Самое худшее, что Вы можете попытаться сделать – это заменить в файле **main.cpp** директиву **#include "algebra.h"** на **#include "algebra.cpp"**. Программа у Вас при этом соберется и даже заработает, но при этом будет содержать ряд недостатков, самые существенные из которых такие:

- Файл algebra.cpp станет, фактически частью файла main.cpp. При любом изменении файла algebra.cpp придется пересобирать файл main.cpp. С ростом количества «подключаемых» таким извращенным образом файлов скорость сборки проекта будет уменьшаться

- Мы не используем возможность раздельной компиляции исходников

Правильное решение будет заключаться в добавлении файлов algebra.h и algebra.cpp в состав проекта при помощи меню **Project→Add Existing Item**, либо контекстного меню Solution Explorer:



Отсутствующая в файле main.cpp функция Sqr() будет найдена в результате компоновки (она находится в объектом файле , получившемся в результате компиляции algebra.cpp). Кроме того, при изменении содержимого файла algebra.cpp перекомпиляции подвергнется только он, после чего программа будет просто скомпонована, что увеличит скорость ее сборки.

> У любопытного студента, вероятно, возник вопрос о том, почему в проект явно не добавляются не только заголовочные файлы стандартной библиотеки, но и соответствующие .cpp и .c файлы с их реализацией. Более того, при сборке проекта компилируются лишь файлы, входящие в состав проекта, а упоминаний о компилировании файлов стандартных библиотек никаких сообщений не выводится.
>
> Дело в том, что файлы стандартных библиотек, как правило, поставляются вместе с компилятором в уже собранном виде в формате .lib и .dll файлов. Пути к каталогам, содержащим данные файлы, хранятся в настройках среды Microsoft Visual Studio, рассмотренных в лабораторной работе №2, а использование стандартных библиотек записано в настройках проекта в момент его создания. Используемые программой функции стандартной библиотеки присоединяются к ней во время сборки программы.
>
> Если Вы планируете использовать в своих программах дополнительные внешние библиотеки сторонних производителей, Вам необходимо указать пути к каталогам с заголовочными и/или библиотечными файлами данных библиотек в настройках среды Visual Studio[2], а использование данных библиотек явно указать в настройках компоновщика в окне свойств вашего проекта.

Тем не менее, текущее решение содержит один недостаток, который мы сейчас продемонстрируем.

---

[2] Если Вы используете средства разработки, отличные от Microsoft Visual Studio, например, утилиту make компилятора gcc/g++, следуйте инструкциям и рекомендациям используемых средств разработки по подключению внешних библиотек.

Представим, что в один прекрасный момент нам понадобится работать с двухмерными векторами. Разместим в заголовочном файле **vector2d.h** объявление следующей структуры, не забыв включить этот файл в состав проекта:

```
struct Vector2d
{
    double x, y;
}
```

Допустим, что нам понадобится функция, вычисляющая скалярное произведение двухмерных векторов, которую мы разместим в файлах algebra.h и algebra.cpp.

algebra.h:

```
#include "vector2d.h"
double Sqr(double arg);
double DotProduct(const Vector2d & vector1, const Vector2d & vector2);
```

Функция DotProduct принимает аргументы по константной ссылке, чтобы избежать затрат на создание копии аргументов (для этого мы используем передачу по ссылке). Квалификатор **const** используется для того, чтобы указать компилятору, что внутри функции значения аргументов vector1 и vector2 модифицироваться не будут.

Передача аргументов в функцию по константной ссылке в ряде случаев выполняется быстрее, нежели их передача по значению, т.к. при этом не происходит создания копии аргументов. Единственная ситуация, при которой передача аргументов по значению выполняется быстрее – когда аргументы являются базовыми типами данных: целочисленные типы данных, указатели, как правило, вещественные.

> Базовые типы данных в большинстве случаев следует передавать в функцию по значению, а составные типы данных (структуры, классы, объединения) – по константной ссылке.

algebra.cpp:

```
#include "algebra.h"
double Sqr(double arg)
{
    return arg * arg;
}

double DotProduct(const Vector2d & vector1, const Vector2d & vector2)
{
    return vector1.x * vector2.x + vector1.y * vector2.y;
}
```

В файле main.cpp разместим следующий код:

```
#include <stdio.h>
#include "algebra.h"
#include "vector2d.h"

int main(int argc, char * argv[])
{
    Vector2d a = {1, 2};
    Vector2d b = {3, 4};
    printf("(%f, %f) * (%f, %f) = \n", a.x, a.y, b.x, b.y, DotProduct(a, b));
    return 0;
}
```

Обратите внимание, что здесь мы сознательно подключили заголовочный файл vector2d.h, хотя он уже был подключен в файле algebra.h. В реальных программах возможны зависимости между различными заголовочными файлами. Так или иначе, на этот раз компиляция программы завершится неудачно с ошибкой:

```
error C2011: 'Vector2d' : 'struct' type redefinition
```

Дело в том, что файл **vector2d.h** оказался включенным в состав файла **main.cpp** дважды – первый раз он был подключен из заголовочного файла **algebra.h**, а второй раз – явно в файле main.cpp. В результате, структура **Vector2d** оказалась объявленной дважды, что является недопустимым в языках C++ и C.

Для решения данной проблемы нам на помощь придут **директивы условной компиляции**[3] языка.

К директивам условной компиляции относятся следующие директивы: #if, #else, #endif, #ifdef, #ifndef. С их подробным описанием вы можете ознакомиться, например, в MSDN, здесь же ограничимся примером исходного кода:

```
#if 1
// этот участок кода будет скомпилирован, т.к. аргумент директивы #if - ненулевой
#else
// этот участок кода скомпилирован не будет (#else действует как переключатель)
#endif // директива #endif завершает область действия директив #if/#else

// этот участок кода компилируется обычным образом (закончилось действие директив)

#define FOO 0
#if FOO
// этот участок кода скомпилирован не будет (значение FOO равно 0)
#endif      // #else можно опускать

#ifdef FOO
/* этот участок кода будет скомпилирован, т.к. директива #ifdef включает компиляцию, если аргумент был определен */
    #ifndef BAR // директивы условной компиляции могут быть вложены друг в друга
      /* этот участок кода также будет обработан, т.к. директива #ifndef включает компиляцию, если ее аргумент не был ранее определен */
    #endif
#else
    #if 1
    /* этот участок кода скомпилирован не будет, несмотря на ненулевое значение аргумента директивы #if, т.к. эта директива находится внутри блока #else-#endif, исключенного из обработки */
    #endif
#endif
```

При помощи директив условной компиляции мы можем «защитить» наши заголовочные файлы от повторного включения в одну и ту же единицу компиляции.

vector2d.h

```
#ifndef VECTOR_2D_INCLUDED_92834782347_
#define VECTOR_2D_INCLUDED_92834782347_
struct Vector2d
{
    double x, y;
}
#endif  // VECTOR_2D_INCLUDED_92834782347_
```

В начале заголовочного файла при помощи директивы **#ifndef** проверяется, не был ли ранее объявлен макрос **VECTOR_2D_INCLUDED_92834782347_**. Имя данного макроса выбрано таким образом, чтобы максимально свести к нулю вероятность его определения в других файлах - используется как неслучайная (VECTOR_2D_INCLUDED_), так и случайная (92834782347_) составляющие имени данного макроса. При

[3] Директивы условной компиляции – специальные инструкции препроцессора языков Си и Си++, позволяющие включить или исключить определенные участки исходного кода из дальнейшей обработки в зависимости от определенных условий.

самом первом включении данного заголовочного файла макрос окажется неопределенным и произойдет обработка оставшейся части исходного кода до директивы **#endif**. При повторном включении файла vector2d.h объявленный ранее макрос **VECTOR_2D_INCLUDED_92834782347_** исключит из обработки внутреннее содержимое директив #ifndef-#endif.

Аналогичным образом «защищается» от повторного включения и заголовочный файл **algebra.h**:

```
#ifndef ALGEBRA_INCLUDED_823472077342EB_
#define ALGEBRA_INCLUDED_823472077342EB_
#include "vector2d.h"
double Sqr(double arg);
double DotProduct(const Vector2d & vector1, const Vector2d & vector2);
#endif  // ALGEBRA_INCLUDED_823472077342EB_
```

Компилятор Microsoft Visual C++ предоставляет более удобную альтернативу данному способу защиты заголовочных файлов от повторного включения – директиву **#pragma once**, исключающую содержащий ее файл от повторного включения:

```
#pragma once
#include "vector2d.h"
double Sqr(double arg);
double DotProduct(const Vector2d & vector1, const Vector2d & vector2);
```

> Следует помнить о том, что другие компиляторы могут не поддерживать директиву **#pragma once**. Поэтому на нее не следует полагаться в том случае, когда исходный код планируется собирать под различные платформы различными компиляторами.

Использование структур

Структуры позволяют создавать составные типы данных, совместно используя несколько переменных для моделирования сложных сущностей предметной области с помощью более простых.

Рассмотрим структуру Complex, моделирующую комплексное число, а также несколько функций для обработки комплексных чисел:

```
#include <math.h>
struct Complex
{
    double re;
    double im;
};

Complex Add(Complex const& arg1, Complex const& arg2)
{
    Complex result = { arg1.re + arg2.re, arg1.im + arg2.im};
    return re;
}

double GetLength(Complex const& arg)
{
    return sqrt(arg.re * arg.re + arg.im * arg.im);
}
```

В реальной программе определение структуры Complex, а также объявления функций их обработки следует вынести в заголовочный файл, например, с именем complex.h, не забыв разместить в нем директивы, защищающие от его повторного включения, а реализацию функций работы с комплексными числами поместить в файл complex.cpp. В этом случае подключение стандартного заголовочного файла math.h, содержащего объявление функции **sqrt**, следует выполнить в файле complex.cpp.

Пример использования структуры Complex:

```
#include <stdio.h>
#include "complex.h"

int main(int argc, char * argv[])
{
    Complex a = {2, 3};
    Complex b = {3.5, -8.8};
    Complex c = Add(a, b);
    printf("Length(c) = %f\n", GetLength(c));
    return 0;
}
```

Практические задания

Для получения оценки «**удовлетворительно**» необходимо выполнить все обязательные задания и набрать не **менее 80 баллов**.

Для получения оценки «**хорошо**» необходимо выполнить все обязательные задания и часть дополнительных, набрав не **менее 180 баллов.**

Для получения оценки «**отлично**» необходимо выполнить все обязательные задания и часть дополнительных, набрав не **менее 300 баллов**.

Внимание, дополнительные задания принимаются только после успешной защиты обязательных заданий.

Обязательные задания

Задание 1. 20 баллов

Ознакомьтесь с возможностями класса **vector** библиотеки STL, а также с работой алгоритмов sort, transform, for_each, и других.

Разработайте программу, выполняющую считывание массива чисел с плавающей запятой, разделяемых пробелами, из стандартного потока ввода в vector, обрабатывающую его согласно заданию Вашего варианта и выводящую в стандартный поток полученный массив (разделенный пробелами). В программе должны быть выделены функции, выполняющие считывание массива, его обработку и вывод результата.

Для тестирования функции, выполняющей обработку массива и выдачу результата, должны быть разработаны тесты, при помощи макроса assert проверяющие корректность ее работы на некотором разумном наборе входных параметров.

Вариант	Выводимое значение
1	Прибавить к каждому элементу массива среднее арифметическое его положительных элементов
2	Каждый элемент массива должен быть умножен на минимальный элемент исходного массива
3	Умножить элементы массива, делящиеся на 3 без остатка, на среднее арифметическое элементов массива, делящихся на 2 без остатка
4	Разделить элементы массива на половину максимального элемента
5	Умножить каждый отрицательный элемент массива на произведение максимального и минимального элементов исходного массива
6	Умножить каждый элемент массива на максимальный элемент исходного массива и разделить на минимальный элемент исходного массива
7	Прибавить к каждому элементу массива сумму трех минимальных элементов массива
8	Элементы, стоящие на четных позициях массива умножить на 2, а из элементов, стоящих на нечетных позициях, вычесть сумму всех неотрицательных элементов

Т.к. вещественные числа представляются в памяти лишь приблизительно, необходимо при подсчете суммы цифр числа принимать в расчет лишь 3 знака после запятой.

Бонус 10 баллов за сортировку элементов массива в порядке возрастания

Бонус начисляется за вывод элементов массива в порядке возрастания их значений

Задание 2

Ознакомьтесь с возможностями класса string (точнее, шаблона basic_string) библиотеки STL и выполните задание, соответствующее номеру Вашего варианта.

Для тестирования разрабатываемой функции должны быть разработаны тесты, при помощи макроса assert проверяющие корректность ее работы на некотором разумном наборе входных параметров.

Вариант	Задание	Балл
1	Разработайте функцию **std::string RemoveExtraSpaces(std::string const& arg)**, удаляющую из строки, переданной в параметре arg, лишние пробелы. Лишними считаются все пробелы в начале и конце строки, а также дополнительные пробелы между словами. Разработайте с ее использованием функцию, выполняющую удаление лишних пробелов из каждой входного потока символов и вывод результирующих строк в выходной поток **Внимание, реализация данной функции должна иметь сложность O(n).**	30
2	Разработайте функцию **std::string TrimBlanks(std::string const& str)**, выполняющую отрезание пробелов в начале и в конце строки str, и возвращающую результирующую строку Разработайте на ее основе программу, выполняющую отрезание пробелов в начале и конце каждой строки, поступающей со стандартного потока ввода, и выводящую результат в стандартный поток вывода **Внимание, реализация данной функции должна иметь сложность O(n).**	20

Задание 3

Ознакомьтесь с возможностями контейнера std::map библиотеки STL и выполните задание, соответствующее номеру Вашего варианта.

Для тестирования разрабатываемой функции должны быть разработаны тесты, при помощи макроса assert проверяющие корректность ее работы на некотором разумном наборе входных параметров.

Вариант	Задание	Балл
1	Разработайте программу, выполняющую подсчет частоты встречаемости слов, поступающих со стандартного потока вывода и выводящую слова их частоты их встречаемости в стандартный поток вывода. Словом считается последовательность из одного и более символов, разделенная последовательностью из одного и более символов разделителей (пробелы, табуляции, символы конца строки). Для подсчета частоты встречаемости символов используйте отображение **слово→частота встречаемости**. **Дополнительно можно получить до 10 баллов,** если программа будет способна обнаруживать русские и английские слова, записанные в разном регистре символов, т.е. считать слова **HeLLo** и **heLLO** одинаковыми.	20

Задание 4.

Ознакомьтесь с возможностями контейнера std::set и выполните задание, соответствующее номеру Вашего варианта.

Для тестирования разрабатываемой функции должны быть разработаны тесты, при помощи макроса assert проверяющие корректность ее работы на некотором разумном наборе входных параметров.

Вариант	Задание	Балл
1	Разработайте функцию **std::set<int> CrossSet(std::set<int> const& set1, std::set<int> const& set2)**, возвращающую результат пересечения[4] двух множеств целых чисел. С ее использованием разработайте программу, выводящую в стандартный поток вывода элементы двух множеств целых чисел и результат их пересечения. **Первое множество** – множество чисел от 1 до N, делящихся без остатка на сумму своих цифр. **Второе множество** – множество целых чисел то 1 до N, сумма цифр которых является четной. Параметр N передается пользователем в виде аргумента командной строки. **Подсказка**: используйте алгоритм **set_intersection**.	30
2	Разработайте функцию **std::set<int> GeneratePrimeNumbersSet(int upperBound)**, возвращающую множество всех простых чисел, не превышающих значения upperBound. С ее использованием разработайте программу, выводящую в стандартный поток вывода элементы множества простых чисел, не превышающие значения, переданного приложению через обязательный параметр командной строки. Максимальное значение верхней границы множества принять равным 100 миллионам.	100

Дополнительные задания

Задание 5.

Ознакомьтесь с возможностями класса string (точнее, шаблона basic_string) библиотеки STL и выполните задание, соответствующее номеру Вашего варианта.

Для тестирования разрабатываемых функций должны быть разработаны тесты, при помощи макроса assert проверяющие корректность ее работы на некотором разумном наборе входных параметров.

Вариант	Задание	Балл
1	Разработайте функцию **std::string FindAndReplace(std::string const& subject, std::string const& search, std::string const& replace)**, возвращающую результат замены всех вхождений подстроки **search** в строке **subject** на строку **replace**. В случае, если искомая строка пустая, замены строк производиться не должно. Разработайте на ее основе программу, заменяющую все вхождения искомой строки в стандартном потоке ввода на строку-заменитель и выводящую результат в стандартный поток вывода. Синтаксис командной строки: **replace.exe** <search-string> <replace-string>	40
2	Разработайте функцию **std::string HtmlEncode(std::string const& text)**, выполняющую кодирование специальных символов строки text соответствующими сущностями HTML: • " (двойная кавычка) заменяется на ***"*** • ' (апостроф) заменяется на **'** • < (знак меньше) заменяется на ***<*** • > (знак больше) заменяется на ***>*** • & (амперсанд) заменяется на ***&*** Разработайте на ее основе программу, выполняющую html-кодирование текста,	40

[4] Пересечением двух множеств является множество, содержащее элементы, присутствующие одновременно и в первом и во втором множестве

	поступающего со стандартного потока ввода, и выводящую результат в стандартный поток вывода.	
3	Разработайте функцию **std::string HtmlDecode(std::string const& html)**, выполняющую декодирование HTML-сущностей строки **html**, перечисленных в варианте 3, обратно в их символьное представление. Разработайте на ее основе программу, выполняющую декодирование html-сущностей текста, поступающего со стандартного потока ввода, и выводящую результат в стандартный поток вывода.	40

4	Разработайте функцию
	bool ParseURL(std::string const& url, Protocol & protocol, int & port, std::string & host, std::string & document),
	выполняющую разбор строки url, и извлечение из нее информации об используемом протоколе, номере порта, имени хоста и имени документа.
	В случае успеха функция должна возвращать true, в случае ошибки – false.
	Protocol – это перечислимый тип, задающий один из известных программе протоколов:
	enum Protocol
	{
	HTTP,
	HTTPS,
	FTP
	};
	Валидным (допустимым) URL-ом программа должна считать строку в следующем формате:
	протокол://***хост***[:***порт***][/***документ***], где
	протокол – http, https или ftp (**в любом регистре**),
	порт – положительное число от 1 до 65535
	(в квадратных скобках указаны опциональные элементы URL-а)
	Если порт не указан, то он должен считаться равным номеру порта по умолчанию для данного протокола (для HTTP – это 80, для HTTPS – 443, для FTP – 21).
	
	Разработайте на ее основе программу, распознающую допустимые URL-строки (разделяемые символами конца строки) в стандартном потоке ввода и выводящую в стандартный поток вывода информацию **о каждом из них** в следующем формате:
	<*Исходный URL*>
	HOST: <*хост*>
	PORT: <*порт*>
	DOC: <*документ*>
	Например, для URL-а
	http://www.mysite.com/docs/document1.html?page=30&lang=en#title
	должно быть выведено:
	`http://www.mysite.com/docs/document1.html?page=30&lang=en#title`
	`HOST: www.mysite.com`
	`PORT: 80`
	`DOC: docs/document1.html?page=30&lang=en#title1`
		60

Задание 6.

Ознакомьтесь с возможностями контейнера std::map библиотеки STL и выполните задание, соответствующее номеру Вашего варианта.

Вариант	**Задание**	**Балл**

<table>
<tr><td>1</td><td>Разработайте программу-словарь, осуществляющую перевод слов и словосочетаний, поступающих со стандартного потока ввода, с английского языка на русский с использованием заданного файла словаря и выводящую результат перевода в стандартный поток вывода.
Если вводимое слово или словосочетание, отсутствует в словаре, программа должна попросить пользователя ввести перевод и запомнить его, в случае, если пользователь ввел непустую строку.
Для выхода из диалога с программой пользователь должен ввести строку, состоящую из трех точек. Перед выходом программа спрашивает о необходимости сохранить изменения в файле словаря, в том случае, если в словарь были добавлены фразы во время текущей сессии работы с программой.
Имя файла словаря передается программе с помощью параметра командной строки.
Формат файла словаря задан в приложении к лабораторной работе в каталоге tests\translator.
Пример диалога пользователя с программой:</td><td>90</td></tr>
<tr><td></td><td>>cat
кот, кошка
>ball
мяч
>meat
Неизвестное слово “meat”. Введите перевод или пустую строку для отказа.
>мясо
Слово “meat” сохранено в словаре как “мясо”.
>meat
мясо
>The Red Square
Неизвестное слово “The Red Square”. Введите перевод или пустую строку для отказа.
>Красная Площадь
Слово “The Red Square” сохранено в словаре как “Красная Площадь”.
>lkkvksmdv
Неизвестное слово “lkkvksmdv”. Введите перевод или пустую строку для отказа.
>
Слово “lkkvksmdv”проигнорировано.
>Тут пользователь нажимает Ctrl+Z, а затем Enter, чтобы ввести символ конца файла с клавиатуры.
В словарь были внесены изменения. Введите Y или y для сохранения перед выходом.
>y
Изменения сохранены. До свидания.</td><td></td></tr>
<tr><td></td><td>Дополнительно можно получить до 10 баллов, если программа будет способна осуществлять перевод английских слов, вводимых пользователем в произвольном регистре символов. Например, слова CaT, при известном переводе для слова cat.</td><td></td></tr>
</table>

Задание 7.

Ознакомьтесь с возможностями контейнера std::set и выполните задание, соответствующее номеру Вашего варианта.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант	Задание	Балл
1	В спортивной школе обучается некоторое количество учеников. Имена учеников – уникальны. Каждый ученик посещает одну или более спортивных секций школы. У тренера каждой секции есть список учеников его секции, заданный в виде текстового файла, в котором в каждой строке записано ФИО ученика. **Задача**: на основе файлов со списками учеников каждого тренера построить список всех учеников школы.	50

	Программа принимает с командной строки имена файлов со списками учеников каждой секции (количество секций в школе, а, следовательно, параметров командной строки – произвольно) и выводит в output список всех учащихся школы. **Указания**: используйте контейнер std::set<std::string> для хранения множества имен учеников школы. В данное множество добавляйте имена учащихся, считываемых из списка учеников каждой секции. После заполнения множества выведите в output его содержимое.	
2	Для повышения культуры общения в чате необходимо написать программу-фильтр, удаляющую из сообщений участников чата недопустимые слова. Список недопустимых слов задается в текстовом файле (слова разделены последовательностью из одного и более пробелов, табуляций или символов конца строки), имя которого передается программе при помощи аргумента командной строки. Программа из каждой вводимой из стандартного потока ввода строки должна удалять присутствующие в ней недопустимые слова и выводить обработанный результат в стандартный поток ввода. Словом считается последовательность из одного и более символов, разделенных последовательностью из одного и более символов-разделителей (пробелы, табуляции, символы конца строки, знаки препинания, знаки арифметических операций, скобки). **Указания**: считайте недопустимые слова во множество строк и при обработке текста проверяйте вхождение каждого слова текста в данное множество. **Дополнительно можно получить 10 баллов**, если фильтрация слов будет производиться **с игнорированием регистра символов**, в котором они записаны. Т.е. если недопустимым словом является слово «дурак», то должны фильтроваться слова «ДуРак», «дурак», «ДУРАК» и подобные.	60

Задание 8 – 150 баллов

Разработайте функцию

std::string ExpandTemplate(std::string const& tpl, std::map<std::string, std::string> const& params), возвращающую результат подстановки параметров, заданных при помощи аргумента params (шаблонный параметр=>значение) в строку tpl. Пустые строки, выступающие в роли ключа в params, должны игнорироваться. Любой шаблонный параметр может встречаться в строке tpl произвольное количество раз. При наличии нескольких возможных вариантов подстановки должен выбираться параметр, имеющий наибольшую длину. Та часть строки tpl, в которую уже была выполнена подстановка, не должна модифицироваться.

Для тестирования функции ExpandTemplate должны быть разработаны тесты, при помощи макроса assert проверяющие корректность ее работы на некотором разумном наборе входных параметров.

Пример использования данной функции и ожидаемые результаты ее работы:

```
using namespace std;

int main()
{
  {
    string const tpl = "Hello, %USER_NAME%. Today is {WEEK_DAY}.";
    map<string, string> params;
    params["%USER_NAME%"] = "Ivan Petrov";
    params["{WEEK_DAY}"] = "Friday";
    assert(ExpandTemplate(tpl, params) == "Hello, Ivan Petrov. Today is Friday.");
  }

  {
    string const tpl = "Hello, %USER_NAME%. Today is {WEEK_DAY}.";
```

```
    map<string, string> params;
    params["%USER_NAME%"] = "Super %USER_NAME% {WEEK_DAY}";
    params["{WEEK_DAY}"] = "Friday. {WEEK_DAY}";
    assert(ExpandTemplate(tpl, params) ==
      "Hello, Super %USER_NAME% {WEEK_DAY}. Today is Friday. {WEEK_DAY}.");
  }

  {
    string const tpl = "-AABBCCCCCABC+";
    map<string, string> params;
    params["A"] = "[a]";
    params["AA"] = "[aa]";
    params["B"] = "[b]";
    params["BB"] = "[bb]";
    params["C"] = "[c]";
    params["CC"] = "[cc]";
    assert(ExpandTemplate(tpl, params) ==
      "-[aa][bb][cc][cc][c][a][b][c]+");
  }
}
```

С использованием данной функции разработайте приложение expand_template, позволяющее заменить вхождения определенных строк в одном файле на определяемые пользователем и записать результат в выходной файл. Синтаксис командной строки приложения:

expand_template.exe <input-file> <output-file> [<param> <value> [<param> <value> …]]

Пример:
expand_template.exe input.txt output.txt "тепло" "холодно" "соленый" "сладкий" "день" "ночь"
записывает в выходной файл результат замены всех вхождений подстроки «тепло» на «холодно», «соленый» на «сладкий», «день» на «ночь», найденных во входном файле.

Ссылки

1. Николай Джосьютис – Стандартная библиотека Си++
2. Скотт Мейерс – Эффективное использование STL. Библиотека программиста
3. Standard C++ Library Reference (MSDN)