Лабораторная работа № 8. Стандартная библиотека шаблонов.

Оглавление

Часть 1. Основные определения STL

Библиотека стандартных шаблонов (Standard Template Library, STL) - наиболее совершенный инструмент языка программирования C++.

Ядро библиотеки стандартных шаблонов состоит из четырех основных элементов:

- контейнеров,
- итераторов,
- алгоритмов,
- функциональных объектов.

Контейнеры - это объекты, предназначенные для хранения других объектов. Контейнеры бывают различных типов: массивы, очереди, списки и т. д.

Алгоритмы - выполняют операции над содержимым контейнеров. Существуют алгоритмы для инициализации, сортировки, поиска или замены содержимого контейнеров.

Итераторы - это объекты, которые по отношению к контейнерам играют роль указателей. Они позволяют получать доступ к содержимому контейнера так, как указатель позволяет получать доступ к элементу массива.

Функциональные объекты - это объекты, действующие как функции. Они могут быть объектами класса или указателями на функции, включающими в себя имя функции, поскольку именно оно работает как указатель.

В настоящее время STL включает в себя 11 структур данных контейнеров, реализованных в виде шаблонов классов. Контейнеры, определенные в STL, представлены в табл.1.

Таблица 1. Контейнеры, определенные в STL

Контейнер	*Описание*	*Заголовок*
`bitset`	Множество битов	`<bitset>`
`deque`	Двунаправленный список	`<deque>`
`list`	Линейный список	`<list>`
`map`	Ассоциативный список для хранения пар ключ/значение, где с каждым ключом связано только одно значение	`<map>`
`multimap`	Ассоциативный список для хранения пар ключ/значение, где с каждым ключом связано два или более значений	`<map>`
`multiset`	Множество, в котором каждый элемент не обязательно уникален	`<set>`
`priority_queue`	Очередь с приоритетом	`<queue>`
`queue`	Очередь	`<queue>`
`set`	Множество, в котором каждый элемент уникален	`<set>`
`stack`	Стек	`<stack>`
`vector`	Динамический массив	`<vector>`

Методы, общие для всех контейнеров, приведены в табл.2.

Таблица 2. Методы, общие для всех контейнеров

Имя	*Назначение*
`size()`	Возвращает число элементов в контейнере
`empty()`	Возвращает `true`, если контейнер пуст
`max_size()`	Возвращает максимально допустимый размер контейнера
`begin()`	Возвращает итератор на начало контейнера
`end()`	Возвращает итератор на конец контейнера
`rbegin()`	Возвращает реверсивный итератор на конец контейнера (итерации происходят в обратном направлении)
`rend()`	Возвращает реверсивный итератор на начало контейнера (итерации происходят в обратном направлении)

Алгоритмы предназначены для обработки контейнеров. Каждый контейнер имеет собственный базовый набор операций, но стандартные алгоритмы обеспечивают более широкий спектр действий. Кроме того, они позволяют одновременно работать с двумя контейнерами, имеющими разные типы. Для обеспечения доступа к алгоритмам STL, нужно подключить заголовочный файл `<algorithm>`.

Основные алгоритмы STL приведены в табл.3.

Таблица 3. Основные алгоритмы STL

Алгоритм	*Действие*
`find`	Возвращает первый элемент с указанным значением
`count`	Считает количество элементов, имеющих указанное значение
`equel`	Сравнивает содержимое двух контейнеров и возвращает `true`, если все соответствующие элементы эквивалентны
`search`	Ищет последовательность, значений в одном контейнере, которая соответствует такой же последовательности в другом

Алгоритм	*Действие*
`copy`	Копирует последовательность, значений из одного контейнера в другой или в другое место того же контейнера
`swap`	Обменивает значения, хранящиеся в разных местах
`sort`	Сортирует значения в указанном порядке
`merge`	Комбинирует два сортированных диапазона значений для получения возможно большего диапазона
`accumulate`	Возвращает сумму элементов в заданном диапазоне
`for_each`	Выполняет указанную функцию для каждого элемента контейнера
`reverse`	Меняет на обратный порядок расположения элементов в контейнере

Часть 2. Строковый класс

В языке C++ встроенный строковый тип отсутствует, но для более удобной работы со строками был разработан строковый класс.

Для работы со строковым классом необходимо подключить к программе заголовочный файл `<string>`. Строковый класс является классом-контейнером, поэтому поддерживает все алгоритмы библиотеки стандартных шаблонов.

Формат оператора описания строки:

```
string имя;
```

Пример:

```
string str1, str2;
```

Ввод-вывод строк. Для ввода-вывода строк используется как уже известные объекты `cin` и `cout`, так и библиотечные функции C++.

Ввод-вывод строки с помощью стандартных потоков ввода-вывода `cin` и `cout`:

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
    string str;
    cin >> str;
    cout << str << endl;
    system("pause");
}
```

Строка вводится точно так же, как и переменные известных нам типов. Если запустите программу и введете строку из нескольких слов, выводится только первое слово. Это связано с тем, что ввод выполняется до первого пробельного символа (то есть пробела, знака табуляции или символа перевода строки `'\n'`).

Если требуется ввести строку, состоящую из нескольких слов, в одну строковую переменную, используется функция `getline`, первым параметром которой является поток `cin`, а второй параметр – имя переменной, которая получит строку, что введет пользователь с клавиатуры:

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
```

```
        string str;
        getline(cin, str);
        cout << str << endl;
        system("pause");
    }
```

Вывод строки происходит как вывод единого целого:

```
cout << "строка = " << str1;
```

Липпман Язык программирования C++ стр.129-130

Операции со строками. Со строками можно выполнять следующие арифметические операции:

`=`	- присваивание значения;
`+=`	- добавление в конец строки другой строки или символа;
`+`	- конкатенация двух строк, конкатенация строки и символа;
`==, !=`	- посимвольное сравнение;
`<, >, <=, >=`	- лексикографическое сравнение.

То есть можно скопировать содержимое одной строки в другую при помощи операции `S1 = S2`, сравнить две строки на равенство при помощи `S1 == S2`, сравнить строки в лексикографическом порядке при помощи `S1<S2`, или сделать сложение (конкатенацию) двух строк в виде `S = S1 + S2`.

Присвоение строк происходит с помощью операции присвоения:

```
str1 = "Mary";
str2 = str1;
```

Доступ к отдельным символам строки может происходить также как к элементам массива:

```
str1[2] = 'n'; // str1="Many";
```

Операция сцепления строк дописывает в строку еще одного слова или любую строку;

```
str2 = str1 + "Hello";
```

Получение длины строки. Методы `size()` и `length()` возвращают количество символов в строке. Рассмотрим пример:

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
	system("chcp 1251>null");
	string str = "Миру - мир";
	cout << str.length() << endl;
	system("pause");
}
```

Результат работы программы:

```
10
Для продолжения нажмите любую клавишу . . .
```

Пример 1. Даны два слова (две переменные). Сколько раз во втором слове встречается первая буква первого слова (не использовать `find`, `erase`, `substr`…).

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
```

```
        system("chcp 1251 > null");
        string s1, s2;
        int result = 0;
        cout << "Введите первое слово: ";
        getline(cin, s1);
        cout << "Введите второе слово: ";
        getline(cin, s2);
        for (int i = 0; i < s2.length(); i++)
              if (s1[0] == s2[i])
                    result++;
        cout << "\nРезультат:" << endl;
        cout << result << endl;
        system("pause");
        return 0;
}
```

Результат выполнения программы:

```
Введите первое слово: апельсин
Введите второе слово: абракадабра

Результат:
5
Для продолжения нажмите любую клавишу . . .
```

Пример 2. Даны 3 слова в 3-х разных переменных. Образовать новую последовательность символов, состоящую из первых букв каждого слова через пробел.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    system("chcp 1251 > null");
    string s1, s2, s3, result;
    cout << "Введите первое слово: ";
    getline(cin, s1);
    cout << "Введите второе слово: ";
    getline(cin, s2);
    cout << "Введите третье слово: ";
    getline(cin, s3);
    result = s1[0];
    result += " ";
    result += s2[0];
    result += " ";
    result += s3[0];
    cout << "\nРезультат:" << endl;
    cout << result << endl;
    system("pause");
    return 0;
}
```

Результат выполнения программы:

```
Введите первое слово: скоро
Введите второе слово: наступит
Введите третье слово: зима

Результат:
с н з
Для продолжения нажмите любую клавишу . . .
```

Пример 3. Имеется некоторая последовательность символов. Образовать новую последовательность, включив в нее символы исходной, кроме символов «g» и «v» (использовать склейку «+») (не использовать `find`, `erase`, `substr`…)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
	system("chcp 1251 > null");
	string text, result = "";
	cout << "Введите текст: ";
	getline(cin, text);
	for (int i = 0; i < text.length(); i++)
		if (text[i] != 'g' && text[i] != 'v')
			result += text[i];
	cout << "\nРезультат:" << endl;
	cout << result << endl;
	system("pause");
	return 0;
}
```

Результат выполнения программы:

```
Введите текст: Python is a programming language that lets you work quickly and integrate systems more effectivaly.

Результат:
Python is a prorammin lanuae that lets you work quickly and interate systems more effectialy.
Для продолжения нажмите любую клавишу . . .
```

Метод c_str() - возвращает указатель на область памяти, в которой хранятся символы строки, возвращает значение типа char*. Возвращаемое значение можно рассматривать как С-строку и использовать в функциях, которые должны получать на вход С-строку.

Пример 4. Дан текст. Переставить в нем первую букву первого предложения и первую букву второго предложения (сначала найти номер первой точки) (не использовать find, erase, substr…)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
	system("chcp 1251 > null");
	string txt;
	cout << "Введите текст: ";
	getline(cin, txt);
	int i = 0, i_second=-1;
	while (1)
	{
		if (i >= txt.length())
			break;
		if (txt[i] == '.')
		{
			i++;
			while (txt[i] == ' ')
				i++;
			i_second = i;
			break;
		}
		i++;
```

```
	}
	char ch=txt[0];
	if (i != 0 && i_second != -1)
	{
		txt[0] = txt[i_second];
		txt[i_second] = ch;
	}
	cout << "\nРезультат:" << endl;
	cout << txt << endl;
	system("pause");
	return 0;
}
```

Результаты работы программы:

```
Введите текст: Похолодало.     Скоро зима.

Результат:
Сохолодало.     Пкоро зима.
Для продолжения нажмите любую клавишу . . .
```

Пример 5. Дан текст. Переписать в другую переменную только цифры, латинские буквы и пробелы (использовать склейку «+») (не использовать find, erase, substr…)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
	system("chcp 1251 > null");
	string txt, result="";
	cout << "Введите текст: ";
	getline(cin, txt);
	for (int i = 0; i < txt.length(); i++)
	{
		if (txt[i] >= '0' && txt[i] <= '9') // если символ - цифра
			result += txt[i];
		if (txt[i] >= 'a' && txt[i] <= 'z') // если символ - строчная буква
			result += txt[i];
		if (txt[i] >= 'A' && txt[i] <= 'B') // если символ - заглавная буква
			result += txt[i];
		if (txt[i] == ' ')                  // если символ - пробел
			result += txt[i];
	}
	cout << "\nРезультат:" << endl;
	cout << result << endl;
	system("pause");
	return 0;
}
```

Результаты работы программы:

```
Введите текст: все пройдет this too will pass 2021

Результат:
  this too will pass 2021
Для продолжения нажмите любую клавишу . . .
```

Передача строк в качестве параметров функций. Переменные типа `string` передаются в функцию также, как переменные другого типа.

Пример 6. Создать функцию `find` поиска в строке символа. Функция принимает два параметра: строку и символ, а возвращает или позицию первого включения символа в строку, или -1, если символ в строке не найден.

Используя функцию `find`, проверить наличие символа @ в строке, заданной пользователем.

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int  find(string s, char c)
{
	for (int i = 0; i<s.length(); i++)
		if (s[i] == c)
			return i;
	return -1;
}

int main()
{
	system("chcp 1251>null");
	int n = 0;
	string line;
	char ch='@';
	cout << "Введите текст: " << endl;
	getline(cin, line);
	if (find(line,ch)!=-1)
		cout << "Есть" << endl;
	else
		cout << "Нет" << endl;
	cout << endl;
	system("pause");
	return 0;
}
```

Пример 7. Создать функцию `find` поиска в строке символа. Функция принимает два параметра: строку и символ, а возвращает или позицию первого включения символа в строку, или -1, если символ в строке не найден.

Используя функцию `find`, из текстового файла вывести на консоль только строки, в которых встречается символ @.

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int  find(string s, char c)
{
	for (int i = 0; i<s.length(); i++)
		if (s[i] == c)
			return i;
	return -1;
}

int main()
{
	system("chcp 1251>null");
	int n = 0;
	string line;
	char ch='@';
	fstream F("text.txt");  //открываем файл в режиме чтения
	if (F)     //если открытие файла прошло корректно, то
```

```
	{ //цикл для чтения значений из файла; выполнение цикла прервется,
	  //когда достигнем конца файла, в этом случае F.eof() вернет истину.
		while (!F.eof())
		{
			getline(F, line); //чтение строки из потока F в line
			if (find(line,ch)!=-1)
				cout << line << endl;   //вывод line на экран
		}
		F.close();//закрытие потока
		cout << endl;
	} //если открытие файла прошло некорректно, то вывод
	  //сообщения об отсутствии такого файла
	else
		cout << " Файл не существует" << endl;
	system("pause");
	return 0;
}
```

Пример 8. Дан текстовый файл. Запишите в другой файл только такие строки исходного файла, которые не содержат чисел.
Создадим функцию, которая принимает в качестве параметра строку, а возвращает или true, если в строке есть числа, или false, если числа в строке не найдены.

```
#include <iostream>
#include <fstream>
#include <string>
#include "Header.h"
using namespace std;

bool is_digit(string line) {
	for (int i = 0; i < line.length(); i++)
		if (line[i] >= '0' && line[i] <= '9')
			return true;
	return false;
}

int main()
{
	system("chcp 1251> null");
	string line;
	ifstream in("text.txt");  //открываем файл в режиме чтения
	ofstream out("result.txt");  //открываем файл в режиме записи
	if (in && out)     //если открытие файлов прошло корректно, то
	{ //цикл для чтения значений из файла и записи; выполнение цикла прервется,
	  //когда достигнем конца файла, в этом случае in.eof() вернет истину.
		while (1)
		{
			getline(in, line); //чтение очередной строки из потока in в
переменную line
			if (is_digit(line))
			{
				out << line << endl;
				cout << line << endl;
			}
			if (in.eof()) break; // выход из цикла, если конец файла достигнут
		}
		in.close(); //закрытие потока
		out.close(); //закрытие потока
		cout << endl << endl;
	} //если открытие файла прошло некорректно, то вывод
	  //сообщения об отсутствии такого файла
	else
		cout << " Файл не существует" << endl;
	system("pause");
```

```
	return 0;
}
```

Пример 9. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, в начале каждой строки добавив “e-mail: “.

Создадим функцию, которая принимает в качестве параметра строку, а возвращает строку, у которой в начале добавлено “e-mail: “.

```
#include <iostream>
#include <fstream>
#include <string>
#include "Header.h"
using namespace std;
string email(string line) {
	line = "e-mail: " + line;
	return line;
}
int main()
{
	system("chcp 1251> null");
	string line;
	ifstream in("text.txt");  //открываем файл в режиме чтения
	ofstream out("result.txt");  //открываем файл в режиме записи
	if (in && out)     //если открытие файлов прошло корректно, то
	{ //цикл для чтения значений из файла и записи; выполнение цикла прервется,
	  //когда достигнем конца файла, в этом случае in.eof() вернет истину.
		while (1)
		{
			getline(in, line); //чтение очередной строки из потока in в
переменную line
			out << email(line) << endl;
			cout << email(line) << endl;
			if (in.eof()) break; // выход из цикла, если конец файла достигнут
		}
		in.close(); //закрытие потока
		out.close(); //закрытие потока
		cout << endl << endl;
	} //если открытие файла прошло некорректно, то вывод
	  //сообщения об отсутствии такого файла
	else
		cout << " Файл не существует" << endl;
	system("pause");
	return 0;
	}
```

Пример 10. Поиск подстроки

Написать программу, которая определяет, встречается ли в заданном текстовом файле заданная последовательность символов. Длина строки текста не превышает 80 символов, текст не содержит переносов слов, последовательность не содержит пробельных символов.

I. ***Исходные данные и результаты***

Исходные данные:

1. Текстовый файл неизвестного размера, состоящий из строк длиной не более 80 символов. Поскольку по условию переносы отсутствуют, можно ограничиться поиском заданной последовательности в каждой строке отдельно. Следовательно, необходимо помнить только одну текущую строку файла. Для ее хранения выделим строковую переменную `line`.

2. Последовательность символов для поиска, вводимая с клавиатуры. Поскольку по условию задачи она не содержит пробельных символов, ее длина также не должна быть более 80 символов, иначе поиск завершится неудачей. Для ее хранения также выделим строковую переменную `word`.

Результатом работы программы является сообщение либо о наличии в каждой строке заданной последовательности, либо об ее отсутствии. Представим варианты сообщений в программе в виде строковых констант.

Для работы с файлом потребуется служебная переменная соответствующего типа.

II. ***Алгоритм решения задачи***

1. Построчно считывать текст из файла.

Для каждой строки проверять, содержится ли в ней заданная последовательность.

Печатать сообщение о наличии заданной последовательности или её отсутствии в каждой строке.

Если файл пустой, напечатать сообщение об отсутствии заданной последовательности и завершить программу.

III. ***Программа и тестовые примеры***

Листинг 10

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
	system("chcp 1251>null");
	string word,line;                  //2
	cout << "Введите слово для поиска:";
	cin >> word;
	ifstream fin("text.txt");          //3
	if (!fin)
	{
		cout << "Ошибка открытия файла." << endl;
		system("pause");
		return 1;                      //4
	}
	int Nomer = 0;                     //1
	while (getline(fin, line))         //5
	{

		cout << Nomer;
		if (line.find(word) == -1)  //6
		{
			cout << " - отсутствует!" << endl;
		}
		else
			cout << " - есть!" << endl;
		Nomer++;
	}
	if (!Nomer) cout << "Отсутствует!" << endl;
	system("pause");
	return 0;
}
```

Рассмотрим помеченные операторы. В операторе 1 описывается переменная `Nomer` для хранения номера текущей строки. В операторе 2 описывается переменная `line` для размещения очередной строки файла и переменная `word` для размещения искомой последовательности символов.

В операторе 3 определяется объект `fin` класса входных потоков `ifstream`. С этим объектом можно работать так же, как со стандартными объектами `cin` и `cout`, то есть использовать операции помещения в поток `<<` и извлечения из потока `>>`.

Предполагается, что файл с именем `text.txt` находится в том же каталоге, что и текст программы, иначе следует указать полный путь, дублируя символ обратной косой черты, так как иначе он будет иметь специальное значение, например:

```
ifstream fin("c:\\prim\\cpp\\text.txt");      // 3
```

В операторе 4 проверяется успешность создания объекта `fin`. Файлы, открываемые для чтения, проверять нужно обязательно! В операторе 5 организуется цикл чтения из файла в переменную `line`.

Функция `getline` успешном чтении строки возвращает значение `true`, при достижении конца файла вернет значение `false`, завершающее цикл.

Для анализа строки в операторе 6 применяется метод `line.find(word)`, который осуществяет поиск подстроки `word` в строке `line`. В случае успешного поиска функция возвращает указатель на найденную подстроку, в случае неудачи – значение - 1.

В качестве тестового примера приготовьте текстовый файл, состоящий из нескольких строк. Для тестирования программы следует запустить ее по крайней мере два раза: введя с клавиатуры слово, содержащееся в файле, и слово, которого в нем нет. Файл должен быть в кодировке ANSI, иначе из-за разных кодировок слова, вводимого с клавиатуры, и текстового файла слово никогда не будет найдено в файле.

Даже такую простую программу мы рекомендуем вводить и отлаживать по шагам. Предлагаемая последовательность отладки:

1. Ввести «скелет» программы (директивы `#include`, функцию `main()`, операторы 1-4). Добавить контрольный вывод введенного слова. Запустив программу, проверить ввод слова и успешность открытия файла. Выполнить программу, задав имя несуществующего файла, для проверки вывода сообщения об ошибке. Удалить контрольный вывод слова.
2. Проверить цикл чтения из файла: добавить оператор 5 с его завершающей фигурной скобкой, внутри цикла поставить контрольный вывод прочитанной строки:

   ```
   cout << line << endl;
   ```

   Удалить контрольный вывод строки.
3. Дополнить программу операторами проверки и вывода сообщений.

Часть 3. Контейнер `vector`

Основные сведения. Шаблонный класс `vector` входит в библиотеку STL (Standard Template Library). Вскоре вы убедитесь, что можно очень много сделать, если знать основные принципы работы класса `vector`, не разбираясь в его реализации.

В классе `vector` поддерживаются динамические массивы. Динамическим массивом называется массив, размеры которого могут увеличиваться по мере необходимости. Класс `vector` оформлен в виде шаблона, что позволяет эффективно использовать его с разными типами. Другими словами, можно создать вектор объектов `double`, вектор объектов `int`, вектор объектов `string` и т. д. При помощи шаблона можно создать «класс чего угодно». Чтобы сообщить компилятору, с каким типом данных будет работать класс (в данном случае — какие элементы будут храниться в векторе), укажите имя нужного типа в угловых скобках `<...>`. Так, вектор объектов `string` обозначается `vector<string>`. Такая запись определяет специализированный вектор, в котором могут храниться только объекты

string. Если попытаться занести в него объект другого типа, компилятор выдаст сообщение об ошибке.

Поскольку класс vector представляет собой «контейнер», то есть предназначается для хранения однотипных элементов, в нем должны быть предусмотрены средства для сохранения и извлечения элементов. Новые элементы добавляются в конец вектора функцией push_back() (помните, что эта функция принадлежит классу, поэтому, чтобы вызвать ее для конкретного объекта, нужно отделить ее имя от имени объекта символом точки). Извлечь элементы из вектора можно, используя индексацию: в классе выполнена перегрузка индексации. Благодаря перегрузке операторов программист работает с вектором как с массивом.

Учитывая все сказанное, рассмотрим пример использования векторов. Для этого следует включить в программу заголовочный файл <vector>:

```
// Копирование всего содержимого файла в вектор строк
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;
int main()
{
	system("chcp 1251>nul");
	vector<string> v;
	ifstream in("Source.cpp");
	string line;
	while (getline(in, line))
		v.push_back(line); // Занесение строки в конец вектора
		// Нумерация строк:
	for (int i = 0; i < v.size(); i++)
		cout << i << ": " << v[i] << endl;
} ///:-
```

Мы открываем файл и последовательно читаем его строки в объекты string. Объекты заносятся в конец вектора v. После завершения цикла while все содержимое файла будет находиться в памяти внутри объекта v.

Условие проверки цикла for означает, что для продолжения работы цикла счетчик i должен быть меньше количества элементов в векторе v (количество элементов определяется функцией size() класса vector).

Откомпилируйте и запустите программу, и вы увидите результат — строки выходного файла окажутся пронумерованными.

Подход оператора >> к работе с потоками позволяет легко изменить программу так, чтобы файл разбивался не по строкам, а по словам, разделенным пропусками:

```
// Разбиение файла по словам, разделенный пропусками
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;
void main()
{
	system("chcp 1251>nul");
	vector<string> words;
	ifstream in("Source.cpp");
	string word;
	while (in >> word)
		words.push_back(word);
	for (int i = 0; i < words.size(); i++)
		cout << words[i] << endl;
}
```

Следующее выражение обеспечивает ввод очередного «слова»: `while(in>>word)`. Когда условие цикла становится ложным, это означает, что был достигнут конец файла.

Чтобы убедиться в том, как просто работать с классом `vector`, рассмотрим следующий пример, в котором создается вектор с элементами типа `int`:

```
// Создание вектора для хранения целых чисел
	#include <iostream>
	#include <vector>
	using namespace std;
	int main()
	{
		vector<int> v;
		for (int i = 0; i < 10; i++)
			v.push_back(i);
		for (int i = 0; i < v.size(); i++)
			cout << v[i] << ", ";
		cout << endl;
		for (int i = 0; i < v.size(); i++)
			v[i] = v[i] * 10; // Присваивание
		for (int i = 0; i < v.size(); i++)
			cout << v[i] << ", ";
		cout << endl;
		return 0;
	}
```

Для класса `vector` определяются следующие операторы сравнения: `=`, `<`, `<=`, `!=`, `>`, `>=`.

Наиболее важными функциями-членами класса `vector` являются функции `size()`, `begin()`, `end()`, `push_back()`, `insert()` и `erase()`. Функция `size()` возвращает текущий размер вектора. Эта функция особенно полезна, поскольку позволяет узнать размер вектора во время выполнения программы. Помните, вектор может расти по мере необходимости, поэтому размер вектора необходимо определять не в процессе компиляции, а в процессе выполнения программы. Функция `begin()` возвращает итератор начала вектора. Функция `end()` возвращает итератор конца вектора. Как уже говорилось, итераторы очень похожи на указатели и с помощью функций `begin()` и `end()` можно получить итераторы (читай: указатели) начала и конца вектора. Функция `push_back()` помещает значение в конец вектора. Если это необходимо для размещения нового элемента, вектор удлиняется. В середину вектора элемент можно добавить с помощью функции `insert()`. Вектор можно инициализировать. В любом случае, если в векторе хранятся элементы, то с помощью оператора индекса массива к этим элементам можно получить доступ и их изменить. Удалить элементы из вектора можно с помощью функции `erase()`.

Пример 1. Как вы знаете, в C++ массивы и указатели очень тесно связаны. Доступ к массиву можно получить либо через оператор индексирования, либо через указатель. По аналогии с этим в библиотеке стандартных шаблонов имеется тесная связь между векторами и итераторами. Доступ к членам вектора можно получить либо через оператор индексирования, либо через итератор. В следующем примере показаны оба этих подхода.

```
	#include <iostream>
	#include <vector>
	using namespace std;

	  int main()
	{
		setlocale(0,"rus");
		vector<int>v; // создание вектора нулевой длины
		int i;
		// помещение значений в вектор
		for (i = 0; i<10; i++) v.push_back(i);
		// доступ к содержимому вектора
```

```
	// с использованием оператора индекса
	for(i = 0; i<10; i++) cout << v[i] << " ";
	cout << endl;
	// доступ к вектору через итератор
	vector<int>::iterator p = v.begin();
	while(p != v.end()) {
		cout << *p << " ";
		p++;
	}
	cout << '\n';
	system("pause");
	return 0;
}
```

Результат выполнения программы:

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
Для продолжения нажмите любую клавишу . . .
```

В этой программе сначала создается вектор `v` нулевой длины. Далее с помощью функции-члена `push_back()` к концу вектора `v` добавляются некоторые значения и размер вектора `v` увеличивается. Обратите внимание на объявление итератора `p`. Тип `iterator` определяется с помощью класса-контейнера. То есть, чтобы получить итератор для выбранного контейнера, объявить его нужно именно так, как показано в примере: просто укажите перед типом `iterator` имя контейнера. С помощью функции-члена `begin()` итератор инициализируется, указывая на начало вектора. Возвращаемым значением этой функции как раз и является итератор начала вектора. Теперь, применяя к итератору оператор инкремента, можно получить доступ к любому выбранному элементу вектора. Этот процесс совершенно аналогичен использованию указателя для доступа к элементам массива. С помощью функции-члена `end()` определяется факт достижения конца вектора. Возвращаемым значением этой функции является итератор того места, которое находится сразу за последним элементом вектора, Таким образом, если итератор `p` равен возвращаемому значению функции `v.end()`, значит, конец вектора был достигнут.

Пример 2. Помимо возможности размещения элементов в конце вектора, с помощью функции-члена `insert()` их можно вставлять в его середину. Удалять элементы из вектора можно с помощью функции-члена `erase()`. В примере представлена демонстрация функций `insert()` и `erase()`.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
	system("chcp 1251>null");
	vector<float>v(5, 1); // создание пятиэлементного вектора из единиц
	// вывод на экран исходных размера и содержимого вектора
	cout << "Размер = " << v.size() << endl;
	cout << "Исходное содержимое:\n";
	for (unsigned int i = 0; i<v.size(); i++) cout << v[i] << " ";
	v.clear();
	for (float i = 0; i < 10; i += 1.1)
		v.push_back(i);
	cout << "\nИсходное содержимое:\n";
	for (unsigned int i = 0; i<v.size(); i++) cout << v[i] << " ";
	vector<float>::iterator p = v.begin();
	p += 2; // указывает на третий элемент
			// вставка в вектор на то место,
			// куда указывает итератор, десяти новых элементов,
			// каждый из которых равен 9
	v.insert(p, 3, 9);
```

```
        // вывод на экран размера
        //и содержимого вектора после вставки
        cout << "\n\nРазмер после вставки = " << v.size();
        cout << "\nСодержимое после вставки:\n";
        for (unsigned int i = 0; i<v.size(); i++) cout << v[i] << " ";
        cout << endl;
        // удаление вставленных элементов
        p = v.begin();
        p += 2; // указывает на третий элемент
        v.erase(p, p + 10); // удаление следующих десяти элементов
                                        // за элементом, на который указывает
                                        // итератор p
                                        // вывод на экран размера
                                        // и содержимого вектора после удаления
        cout << "\nРазмер после удаления – " << v.size();
        cout << "\nСодержимое после удаления:\n";
        for (unsigned int i = 0; i<v.size(); i++) cout << v[i] << " ";

        cout << '\n';
        system("pause");
        return 0;
}
```

Результат выполнения программы:

```
Размер = 5
Исходное содержимое:
1 1 1 1 1
Исходное содержимое:
0 1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9

Размер после вставки = 13
Содержимое после вставки:
0 1.1 9 9 9 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9

Размер после удаления – 3
Содержимое после удаления:
0 1.1 9.9
Для продолжения нажмите любую клавишу . . .
```

Пример 3. В следующем примере вектор используется для хранения объектов класса, определённого программистом. Обратите внимание, что в классе определяются конструктор по умолчанию и перегруженные версии операторов < и ==.

```
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;

class Demo {
        double d;
public:
        Demo() { d = 0.0; }
        Demo(double x) { d = x; }
        Demo &operator- (double x) {
                d = x; return *this;
        }
        double getd() { return d; }
        friend bool operator< (Demo a, Demo b) {
                return a.getd() < b.getd();
        }
        friend bool operator==(Demo a, Demo b) {
                return a.getd() == b.getd();
        }
```

```
};
int main() {
	setlocale(0, "rus");
	vector<Demo>v = { Demo(6.1), Demo(3.9), Demo(2.7), Demo(1.2)};
	unsigned int i;
	cout << setprecision(3);
	for (i = 0; i<10; i++)
		v.push_back(Demo(i / 3.0));
	for (i = 0; i<v.size(); i++)
		cout << v[i].getd() << '\t';
	cout << endl;
	for (i = 0; i<v.size(); i++)
		v[i] = v[i].getd() * 2.1;
	for (i = 0; i<v.size(); i++)
		cout << v[i].getd() << '\t';
	cout << endl;
	system("pause");
	return 0;
}
```

Результат выполнения программы:

```
6.1     3.9     2.7     1.2     0       0.333   0.667   1       1.33    1.67
2       2.33    2.67    3
12.8    8.19    5.67    2.52    0       0.7     1.4     2.1     2.8     3.5
4.2     4.9     5.6     6.3
Для продолжения нажмите любую клавишу . . .
```

В этой программе сначала создается вектор `v`, в который сразу записывается четыре объекта класса `Demo`. Затем в вектор в цикле добавляется ещё десять объектов. Следующий цикл выводит на экран содержимое поля `d` каждого объекта, при этом используется метод `getd()`. Ешё следующий цикл меняет значение поля `d` каждого объекта с помощью индексации и присваивания. И последний цикл снова выводит содержимое вектора на экран.

Пример 4. Следующий пример – это доработанный пример 3. Здесь используется функция вывода вектора на экран.

```
#include <iostream>
#include <string.h>
#include <vector>
#include <iomanip>
using namespace std;
class Demo {
	double d;
public:
	Demo() { d = 0.0; }
	Demo(double x) { d = x; }
	Demo &operator- (double x) {
		d = x; return *this;
	}
	double getd() { return d; }
	bool operator< (Demo& a) {
		return getd() < a.getd();
	}
	bool operator==(Demo& a) {
		return getd() == a.getd();
	}
};
void print(vector<Demo> v) {
	unsigned int i;
	for (i = 0; i<v.size(); i++)
		cout << v[i].getd() << '\t';
}
int main() {
```

```
setlocale(0, "rus");
vector<Demo>v = { Demo(6.1),Demo(3.9),Demo(2.7),Demo(1.2)};
unsigned int i;
cout << setprecision(3);
for (i = 0; i<10; i++)
	v.push_back(Demo(i / 3.0));
print(v);
cout << endl;
for (i = 0; i<v.size(); i++)
	v[i] = v[i].getd() * 2.1;
print(v);
cout << endl;
system("pause");
return 0;
}
```

Часть 4. Контейнер `list`

Основные сведения. Шаблонный класс `list` входит в библиотеку STL (Standard Template Library).

В библиотеке `STL` (напомним, что это – библиотека шаблонных классов - `Standard Template Library`) описывается шаблонный класс `list`, который позволяет описывать списки переменных любого типа. Формальное определение шаблона дано ниже:

```
template <class Type, class Allocator=allocator<Type>> class list { … };
```

Для работы объектами этого класса необходимо подключить заголовочный файл:

```
#include <list>
```

Для описания списков предусмотрены разные конструкторы. Примеры описания списков даны ниже:

```
// Списки list описания и инициализация
list <int> l0; // Пустой список l0
list <int> l1(3); // Список с тремя элементами равными 0
list <int> l2(5, 2); // Список из пяти элементов равными 2
list <int> l3(l2); // Список l3 на основе списка l2
list <int>::iterator l3_Iter; // Описание итератора l3_Iter
l3_Iter = l3.begin();  // Итераторные вычисления на начало
l3_Iter++; l3_Iter++; // Итераторные вычисления продвинуть на два объекта
list <int> l4(l3.begin(), l3_Iter); // Список l4 на основе первых двух элементов L3
```

Функцию печати опишем в заголовочном файле так, чтобы в ней в данной функции использован прямой итератор для класса `list`, а для индексации при печати элементов списка использовалась вспомогательная целая переменная:

```
void lPrint(list<int>& l)
{
	list<int>::iterator iter;
	int i = 0;
	if (!l.empty())
	{
		for (iter = l.begin(); iter != l.end(); iter++, i++)
		{
			cout << "l[" << i << "] = " << *iter << endl;
		}
	}
	else
		cout << "Список пуст!" << endl;
}
```

Для описания пустого списка `l` (типа `list`) также нужно указать тип `int`

(инстанцировать шаблонный объект):

```
list <int> l; // Пустой список l
cout<< "Добавление:" << endl;
l.push_back( 1 ); // Добавление в конец списка
l.push_back( 2 ); // Добавление в конец списка
l.push_front( 5 ); // Добавление в начало списка
lPrint(l); // Собственная функция печати целого списка
```

В функции печати используется итератор `iter`, который позволяет продвигаться по списку (`iter++`). Для установки итератора на первый элемент используется метод `begin`. Остановка просмотра проверяется методом `end`. Для проверки пустого списка используется метод `empty`. Для дальнейшей демонстрации возможностей списков будем использовать эту функцию. В первом фрагменте получим результат:

```
Добавление:
l[0] = 5
l[1] = 1
l[2] = 2
```

Перечень методов класса список (`list`) приведен в разделе Справочные материалы, подробное описание методов вы можете найти в документации, литературе и справочной системе MSDN [5]. Рассмотрим еще несколько примеров использования объектов класса список и его методов. Для копирования списков может быть использован метод слияния (`assign`), а для очистки списка может быть использован метод удаления (`erase` или `clear`):

```
// Копирование и очистка
l20.erase(l20.begin(), l20.end());
lPrint(l20);
l20.assign(l.begin(), l.end());
cout << "После копирования l в l20:" << endl;
lPrint(l20);
l20.clear();
cout << "После очистки l20:" << endl;
lPrint(l20);
```

В этом фрагменте получим результат:

```
Список пуст!
После копирования l в l20:
l[0] = 5
l[1] = 1
l[2] = 2
После очисткиl20:
Список пуст!
```

Получить текущую размерность списка можно методом **size**:

```
cout << "Число элементов в списке =  " << l.size( )<< endl;
```

Получим результат:

```
Число элементов в списке =  3
```

Манипулировать с содержимым списка (вставка, удаление и т.д.) можно с помощью методов (`insert`, `remove`, `erase`, `remove_if` и др.). Ниже приводиться работающий фрагмент программы, иллюстрирующий эти действия (комментарии даны в тексте программы):

```
// Вставка
iter = l.begin();
iter++; iter++; // Вставка после второго
cout << "Вставка:" << endl;
l.insert(iter, 100);
lPrint(l);
// Удаление по числу
l.remove(100);
cout << "Удаление по числу 100:" << endl;
```

```
lPrint(l);
// Удаление по номеру
l.erase(l.begin()); // Из головы списка
cout << "Удаление по номеру 0:" << endl;
lPrint(l);
// Добавим и удалим по итератору
l.push_back(20);
l.push_back(31);
cout << "Перед удалением:" << endl;
lPrint(l);
iter = l.begin();
iter++; iter++; // Сдвиг на два элемента
l.erase(iter);
cout << "Удаление по итератору 2 (3-й элемент):" << endl;
// Удаление по условию (предикату)
lPrint(l);
list <int> l22 = l;
cout << "Удаление четных чисел:" << endl;
cout << " До удаления:" << endl;
lPrint(l22);
l22.remove_if(is_odd<int>());
cout << " После удаления:" << endl;
lPrint(l22);
```

В последнем случае нужно создать специальный класс с булевским оператором шаблонного типа (унарный одноместный предикат - функция):

```
template <class T>
class is_odd : public std::unary_function<T, bool>
{
public:
	bool operator( ) (T& val)
	{
		return (val % 2) != 1;
	}
} // Условие предиката – четное = 0
```

Это позволяет установить для каждого элемента списка значение условия для его удаления. В нашем случае выполняется проверка на честность параметра элемента, передаваемого в функцию. В задании на ЛР необходимо для своего целого списка объявить шаблонный класс для условия удаления элементов с заданным значением переменной.

Для предыдущего фрагмента удаления четных значений из списка l2 (по условию), созданного на основе l, получим:

```
Вставка:
l[0] = 5
l[1] = 1
l[2] = 100
l[3] = 2
Удаление по числу 100:
l[0] = 5
l[1] = 1
l[2] = 2
Удаление по номеру 0:
l[0] = 1
l[1] = 2
Перед удалением:
l[0] = 1
l[1] = 2
l[2] = 20
```

```
l[3] = 31
Удаление по итератору 2 (3-й элемент):
l[0] = 1
l[1] = 2
l[2] = 31
Удаление четных чисел:
До удаления:
l[0] = 1
l[1] = 2
l[2] = 31
После удаления:
l[0] = 1
l[1] = 31
```

Сортировка списка выполняется методом `sort`, при этом может быть указан параметр или не указан. Если параметр не задан, то сортировка выполняется в порядке возрастания ключевой переменной (`less`). При задании параметра сортировки `greater` сортировка выполняется в порядке убывания основной переменной списка. Примеры и результат приведены ниже.

```
// Сортировка
cout << "До сортировки списка!" << endl;
lPrint(l);
l.sort(greater<int>()); // Сортировка по убыванию
cout << "После greater сортировки списка!" << endl;
lPrint(l);
l.sort(less<int>()); // Явная сортировка по возрастанию
cout << "После less сортировки списка!" << endl;
lPrint(l);
// Список l22
cout << "До сортировки списка (l22)!" << endl;
l22.push_front(555); // Добавление в начало списка
lPrint(l22);
l22.sort(); // Сортировка по возрастанию по-умолчанию ( возрастание)
less
cout << "После сортировки списка(по-умолчанию)!" << endl;
lPrint(l22);
```

Получим результат:

```
До сортировки списка!
l[0] = 1
l[1] = 2
l[2] = 31
После greater сортировки списка!
l[0] = 31
l[1] = 2
l[2] = 1
После less сортировки списка!
l[0] = 1
l[1] = 2
l[2] = 31
До сортировки списка (l22)!
l[0] = 555
l[1] = 1
l[2] = 31
После сортировки списка( по-умолчанию)!
l[0] = 1
l[1] = 31
l[2] = 555
```

Переупорядочение списка, изменение порядка на обратный, выполняется методом `reverse`. Пример:

```
cout << "До сортировки списка!" << endl;
lPrint(l);
cout << "После reverse сортировки списка!" << endl;
l.reverse();
lPrint(l);
```

Получим результат:

```
До сортировки списка!
l[0] = 1
l[1] = 2
l[2] = 31
После reverse сортировки списка!
l[0] = 31
l[1] = 2
l[2] = 1
```

Очистка списка (clear):

```
l.clear( );
cout<< "После очистки списка!" << endl;
lPrint(l);
```

Получим результат:

```
Пустой список!
```

Использование класса list для включения пользовательских объектов

В рамках ЛР необходимо продемонстрировать использование методов класса список - list для простых объектов класса, который определяется вариантом задания. Отличие заключается в том, что здесь используются нестандартные переменные (не стандартный тип int). Использование своего класса в качестве содержимого списка имеет ряд особенностей. На примере простого класса **Point** рассмотрим их. Пусть создан простой класс:

```
class Point {
public:
	int x;
	int y;
	Point() { x = 0; y = 0; };
	Point(int a, int b) { x = a; y = b; };
	Point(const Point & p) { x = p.x; y = p.y; };
	Point & operator=(const Point & p) { x = p.x; y = p.y; return *this; };
	friend ostream & operator <<(ostream &  out, Point &  obj);
	friend bool operator==(const Point & p1, const Point & p2);
	friend bool operator>(const Point & p1, const Point & p2);
	friend bool operator<(const Point & p1, const Point & p2);
};
// Перегурузка операции для сортировки
bool operator<(const Point & p1, const Point & p2)
{
	return (p1.x < p2.x);
};
// Перегурузка операции для сортировки
bool operator>(const Point & p1, const Point & p2)
{
	return (p1.x > p2.x);
};
// Перегурузка операции сравнения для удаления по объекту
bool operator==(const Point & p1, const Point & p2)
{
	return ((p1.x == p2.x) && (p1.y == p2.y));
};
// Перегурузка операции для увывода объекта
```

```
ostream & operator <<(ostream &  out, Point &  obj)
{
	out << "{ x = " << obj.x << " y = " << obj.y << " } " << endl;
	return out;
};
```

После этого мы можем описать разнообразные списки этих объектов, используя инстанцирование и итераторы. Ниже приведены примеры вариантов использования конструкторов:

```
	list<Point> lP;
	list<Point>::iterator PIter;
	Point P1(11, 22);
	Point P2(51, 52);
	cout << "КЛАСС POINT!!!" << endl;
	lP.push_back(P1);
	lP.push_back(P2);
	lP.push_front(*new Point(31, 32));
	lPPrint(lP);
```

Функция печати списков из `Point` имеет вид:

```
// Функция печати для Point
void lPPrint(list<Point>& l)
{
	list<Point>::iterator iter;
	int i = 0;
	if (!l.empty())
	{
		for (iter = l.begin(); iter != l.end(); iter++, i++)
		{
			cout << "list[" << i << "] = " << *iter << endl;
		}
	}
	else
		cout << "Список пуст!" << endl;
}
```

Тогда получим следующий результат:

```
КЛАСС POINT!!!
list[0] = { x = 31 y = 32 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 51 y = 52 }
```

Можно описать списки на основе разных конструкторов, а также итератор для навигации по ним:

```
	list <Point> lp; // Пустой список lp0
	list <Point> lp1(3); // Список с тремя элементами равными 0
	list <Point> lp2(5, P2); // Список из пяти элементов равными P2
	list <Point> lp3(lp2); // Список l3 на основе списка l2
	list <Point>::iterator lp3_Iter;
	lp3_Iter = lp3.begin();
	lp3_Iter++; lp3_Iter++;
	list <Point> lp4(lp3.begin(), lp3_Iter); // Новый список lp4 на основе
первых двух элементов lp3
	cout << "КОНСТРУКТОРЫ:  КЛАСС POINT!!!" << endl;
	cout << "lp:" << endl;
	lPPrint(lp);
	cout << "lp1:" << endl;
	lPPrint(lp1);
	cout << "lp2:" << endl;
	lPPrint(lp2);
	cout << "lp3:" << endl;
```

```
lPPrint(lp3);
cout << "lp4:" << endl;
lPPrint(lp4);
```

Получим после выполнения этого фрагмента программы:

```
КОНСТРУКТОРЫ:  КЛАСС POINT!!!
lp:
Список пуст!
lp1:
list[0] = { x = 0 y = 0 }
list[1] = { x = 0 y = 0 }
list[2] = { x = 0 y = 0 }
lp2:
list[0] = { x = 51 y = 52 }
list[1] = { x = 51 y = 52 }
list[2] = { x = 51 y = 52 }
list[3] = { x = 51 y = 52 }
list[4] = { x = 51 y = 52 }
lp3:
list[0] = { x = 51 y = 52 }
list[1] = { x = 51 y = 52 }
list[2] = { x = 51 y = 52 }
list[3] = { x = 51 y = 52 }
list[4] = { x = 51 y = 52 }
lp4:
list[0] = { x = 51 y = 52 }
list[1] = { x = 51 y = 52 }
```

Все другие методы класса `list` из STL также можно использовать для построения списков на основе класса `Point`. Рассмотрим и другие примеры. Метод позволяет `assign` добавить в новый список элементы из другого списка, а метод `clear`, очистить список:

```
// Копирование и очистка
//list <Point> lp20; // - Можно и так описать список lp20
typedef list<Point> ListPoint; // Зададим новый тип - ListPoint список точек
ListPoint lp20;
cout<< "Исходный список lP:" << endl;
lPPrint(lP); // см. выше пример
lPPrint(lp20);
lp20.assign( lP.begin( ), --lP.end( ) ); // копируем без последнего элемента списка
cout<< "После копирования lP в lp20:" << endl;
lPPrint(lp20);
lp20.clear( );
cout<< "После очистки lp20:" << endl;
lPPrint(lp20);
```

Получим результат, при копировании всего списка без последнего элемента (`--lP.end( )`):

```
Исходный список lP:
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
l[2] = { x = 51 y = 52 }
Список пуст!
После копирования lP в lp20:
l[0] = { x = 31 y = 32 }
l[1] = { x = 1 y = 2 }
После очистки lp20:
Список пуст!
```

Манипулировать с содержимым списка (вставка, удаление и т.д.) можно с помощью методов, рассмотренных выше для стандартных типов: `insert`, `erase`, `remove_if` и др. Примеры:

```
// Вставка по итератору
```

```
PIter = lP.begin();
PIter++; PIter++;  //итератор переместим на 2 элемента
cout << "Вставка: до вставки 2-го:" << endl;
lPPrint(lP);
cout << "Вставка: после вставки 2-го:" << endl;
lP.insert(PIter, P1);  // 11-22
lPPrint(lP);
//////////////////
// Удаление по итератору
lP.erase(lP.begin());
lP.push_back(*new Point(20, 62));
cout << "Удаление по итератору (0 - первого):" << endl;
lPPrint(lP);
// Удаление по простой предикат для удаления x = 20
lP.remove_if(is_X20<Point>()); // Работает!!!
cout << "После удаления x = 20 списка!" << endl;
lPPrint(lP);
```

Простой унарный предикат (задается в виде булевской унарной функции шаблонного класса) для удаления точек с условием `x = 20` выглядит так:

```
// Класс для предиката
template <class T> class is_X20 : public std::unary_function<T, bool>
{
public:
    bool operator( ) (T& val)
    {
        return (val.x == 20); // Возвращается истина, если объект в списке имеет x = 20
    }
}
```

В результате выполнения данного фрагмента программы получим:

```
Вставка: до вставки 2-го:
list[0] = { x = 20 y = 32 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 51 y = 52 }
Вставка: после вставки 2-го:
list[0] = { x = 20 y = 32 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 11 y = 22 }
list[3] = { x = 51 y = 52 }
Удаление по итератору (0 - первого):
list[0] = { x = 11 y = 22 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 51 y = 52 }
list[3] = { x = 20 y = 62 }
После удаления x = 20 списка!
list[0] = { x = 11 y = 22 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 51 y = 52 }
```

Функция `remove` (удалить по объекту) требует описания перегруженной операции равенства, присваивания и явных конструкторов копирования. Например:

```
Point(const Point & p){ x = p.x ; y = p.y ;};
Point & operator=(const Point & p){ x = p.x ; y = p.y ; return *this;}; // Конструктор копии
friend bool operator==(const Point & p1 , const Point & p2) ;
// ...
// Перегрузка операции сравнения для удаления по объекту
bool operator==(const Point & p1 , const Point & p2)
{
    return ( (p1.x==p2.x) && (p1.y==p2.y) );
};
```

Тогда можно выполнить следующий фрагмент текста с функцией `remove`:

```
// Вставка по итератору
  PIter = lP.begin( );
  PIter++; PIter++;  //итератор переместим на 2 элемента
  cout<< "Вставка: до вставки 2-го:" << endl;
  lPPrint(lP);
  cout<< "Вставка: после вставки 2-го (P1):" << endl;
  lP.insert( PIter, P1 ); // <11,22>
  lPPrint(lP);
///
   cout<< "Размер списка:" <<lP.size() << endl; // Размер списка
   lP.insert( PIter, P2 );
 lP.insert( PIter, *new Point(51, 61));
  cout<< "Вставка: после вставки 2-го(P2 вставка 2-х):" << endl;
 lPPrint(lP);
 lP.remove( P2 );
 cout<< "Удаление по объекту P2 = <51,52> (после):" << endl;
 lPPrint(lP);
```

В результате выполнения данного фрагмента программы получим (удаляются точки с координатами <51,52>):

```
Вставка: до вставки 2-го:
list[0] = { x = 20 y = 32 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 51 y = 52 }
Вставка: после вставки 2-го <11,22> (P1):
list[0] = { x = 20 y = 32 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 11 y = 22 }
list[3] = { x = 51 y = 52 }
Размер списка:4
Вставка: после вставки 2-го(P2 вставка 2-х):
list[0] = { x = 20 y = 32 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 11 y = 22 }
list[3] = { x = 51 y = 52 }
list[4] = { x = 51 y = 61 }
list[5] = { x = 51 y = 52 }
Удаление по объекту P2 = <51,52> (после):
list[0] = { x = 20 y = 32 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 11 y = 22 }
list[3] = { x = 51 y = 61 }
```

Функция `remove_if` тоже требует описания унарных предикатов. Пусть мы имеем доступную глобальную переменную (GP), описанную так:

```
// Глобальная переменная
Point GP(51,52);
```

Тогда можно описать унарный предикат вида:

```
// Унарная функция для предикта условного удаления по объекту
template <class T> class is_XA : public std::unary_function<T, bool> // Предикат на ее основе
{
public:
  bool operator( ) (  T& val   )
  {
  return ( val == GP );    } // Использование перегруженной операции "= ="
};
//
```

После этого можно выполнить следующий фрагмент программы:

```
// Удаление по условию равенства объектов – предикат - is_XA
```

```
cout<< " Вставка по итератору  P2 (до):" << endl;
lP.insert( PIter, P2 );  // P2 = <51,52>
Print_list(lP);
GP.x = 51; GP.y = 52;
lP.remove_if( is_XA<Point>( )  );
cout<< "Удаление по условию = Point(51,52) P2 (после):" << endl;
Print_list(lP);
```

Получим в результате

```
Вставка по итератору  P2 (до):
list[0] = { x = 11 y = 22 }
list[1] = { x = 51 y = 52 }
list[2] = { x = 11 y = 22 }
list[3] = { x = 51 y = 61 }
Удаление по условию = Point(51,52) P2 (после):
list[0] = { x = 11 y = 22 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 51 y = 61 }
```

Функции `sort` требуют описания перегруженных операций сравнения (“>”,”<”) для объекта типа `Point`. Для упрощения предиката зададим сортировку только по параметру класса `x`. Например, дружественной функцией:

```
//…
friend bool operator>(const Point & p1 , const Point & p2) ;
friend bool operator<(const Point & p1 , const Point & p2) ;
//…
// Перегурузка операции для сортировки
bool operator<(const Point & p1 , const Point & p2)
{
        return ( p1.x < p2.x );
}
// Перегурузка операции для сортировки
bool operator>(const Point & p1 , const Point & p2)
{
        return ( p1.x > p2.x );
}
```

Пример использования метода `sort` показан ниже. Используемый ниже метод `reverse` позволяет изменить порядок расположения объектов на обратный порядок. После этого можно выполнить следующий фрагмент программы:

```
// Сортировка
lP.push_front( *new Point(20 ,32) );
cout<< "Обратный порядок (до):" << endl;
lPPrint(lP);
lP.reverse( ); // Изменение порядка на обратный
cout<< "Обратный порядок (после):" << endl;
lPPrint(lP);
cout<< "Сортировка по возрастанию x:" << endl;
lP.sort( ); // По умолчанию less - сортировкапо возрастанию
lPPrint(lP);
cout<< "Сортировка по убыванию x:" << endl;
lP.sort( greater<Point>( ) );
lPPrint(lP);
```

Получим в результате:

```
Обратный порядок (до):
list[0] = { x = 20 y = 32 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 11 y = 22 }
list[3] = { x = 51 y = 61 }
Обратный порядок (после):
```

```
list[0] = { x = 51 y = 61 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 11 y = 22 }
list[3] = { x = 20 y = 32 }
Сортировка по возрастанию x:
list[0] = { x = 11 y = 22 }
list[1] = { x = 11 y = 22 }
list[2] = { x = 20 y = 32 }
list[3] = { x = 51 y = 61 }
Сортировка по убыванию x:
list[0] = { x = 51 y = 61 }
list[1] = { x = 20 y = 32 }
list[2] = { x = 11 y = 22 }
list[3] = { x = 11 y = 22 }
```

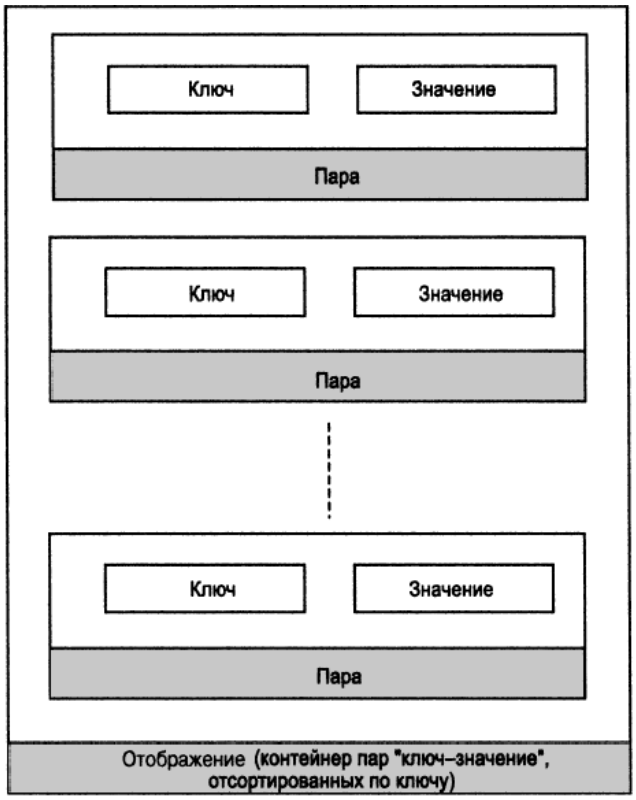
Предлагаю студентам разобраться с использованием других предикатов самостоятельно. Особенностью использования собственных объектов является необходимость перегрузки операций вывода. Другой особенностью класса `list` является автоматическое корректное удаление динамических объектов, созданных в программе и включенных в массив.

Часть 4. Контейнер `map` (отображение)

Основные сведения. Шаблонный класс `map` входит в библиотеку STL (Standard Template Library). Вскоре вы убедитесь, что можно очень много сделать, если знать основные принципы работы класса `map`, не разбираясь в его реализации.

Ассоциативные контейнеры (отображения), хранящие данные в отсортированном виде, сродни словарю. В результате вставка в них осуществляется медленнее, чем в последовательные контейнеры, но когда дело доходит до поиска, преимущества ассоциативных контейнеров оказываются существенными.

Отображение хранит пары “ключ-значение” с уникальными ключами, отсортированными по значениям ключей.



Класс `pair` (пара) стандартной библиотеки C++ позволяет нам определить одним объектом пару значений, если между ними есть какая-либо семантическая связь. Эти значения могут быть одинакового или разного типа. Например, инструкция

```
pair <string, int> pr("Key", 10);
```

создает объект `pr` типа `pair`, состоящий из двух значений.
Отдельные части пары могут быть получены с помощью членов `first` и `second`:

```
#include <iostream>
#include <string>
using namespace std;

void main() {
	pair <string, int> pr("Key", 10);
	cout << "pr: " << pr.first << ' ' << pr.second << endl;
	system("pause");
}
```

Итак, `map` содержит пары (объекты класса-шаблона `pair`), каждая пара состоит из *ключа* (`first`) и *значения* (`second`), ассоциированного с ключом.

Создание экземпляра класса `map`

Создать экземпляра класса `map` можно следующими способами:

1. объявить экземпляр класса `map` без инициализации. Для этого перед названием переменной указывается название класса. После названия переменной внутри угловых скобок через запятую задаются типы данных ключа и значения. В этом случае объект не содержит элементов. Пример объявления без инициализации:

```
map <string, int> m;
```

2. указать объект класса `map` внутри круглых скобок или после оператора `=`:

```
map <string, int> m1;
map <string, int> m2(m1);
map <string, int> m3 = m1;
```

Вставка элементов

Вставить элемент можно, указав ключ внутри квадратных скобок. Значение элемента задается после оператора присваивания. Если ключ уже существует, то вместо вставки элемента производится изменение значения. Пример:

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

void main() {
	map <string, int> m;
	map <string, int>::iterator i;
	m["one"] = 0;		// заполнение в контейнер
	m["two"] = 2;		// заполнение в контейнер
	m["one"] = 1;		// изменение значения по ключу

	for(i=m.begin(); i!=m.end(); i++) {
		cout << i->first << " -> " << i->second << "; ";
	}
} // one -> 1; two -> 2;
```

Над двумя объектами класса `map` определены операции `==`, `!=`, `<`, `<=`, `>` и `>=`. Кроме того, один объект можно присвоить другому объекту. Пример:

```
map <int, int> m1, m2;
map <int, int>::iterator i;
m1[0] = 10;  // заполнение контейнера
m1[1] = 20;  // заполнение контейнера
m2 = m1;
for(i=m2.begin(); i!=m2.end(); i++) {
      cout << i->first << ' ' << i->second << "; ";

}// 0 10; 1 20;
```

Обратите внимание, как в примере производится перебор всего содержимого отображения и вывод каждой пары «*ключ/значение*». Для этого используется итератор `i` :

```
map <int, int>::iterator i;
```

разыменование которого дает пару (объект `pair`) с ключом и значением. Доступ к компонентам пары осуществляется через переменные `first` и `second`:

```
cout << i->first << ' ' << i->second << "; ";
```

Как уже упоминалось, в ассоциативном контейнер можно хранить только уникальные ключи. Дублирования ключей не допускается.

Как правило, для любого объекта, заданного в качестве ключа, должны быть определены конструктор по умолчанию и несколько операторов сравнения.

Удаление элементов

Для удаления элементов из отображения предназначены следующие методы:

1. `erase()` - удаляет один элемент или элементы из диапазона.
   Пример удаления элемента с указанным ключом:
   ```
   void main(){
      map <string, int> m;
      map <string, int>::iterator i;
      m["one"] = 1;
      m["two"] = 2;
      m.erase("one");
      for (i=m.begin(); i!=m.end(); ++i)     {
            cout << i->first << " -> " << i->second << "; ";
      } // two -> 2;
   }
   ```
   Пример удаления элемента, на который указывает итератор:
   ```
      void main() {
            map <string, int> m;
            map <string, int>::iterator i;
            m["one"] = 1;       // заполнение в контейнер
            m["two"] = 2;
            m.erase(--m.end());
            for (i = m.begin(); i != m.end(); i++) {
                  cout << i->first << " -> " << i->second << "; ";
            }
      }  // one -> 1;
   ```
2. `clear()` – удаление всех элементов.
3. `empty()` – возвращает значение `true`, если контейнер не содержит элементов, и `false` – в противном случае.
4. `size()` – возвращает количество элементов в контейнере. Пример:
   ```
      void main(){
            map<string, int> m;
            map<string, int>::iterator i;
            m["one"] = 1;
            m["two"] = 2;
            m["third"] = 3;
   ```

```
        m["four"] = 4;

        cout << "size: " << m.size() << endl; // 4
        m.clear();
        if ( m.empty() )   {
              cout << "Нет элементов\n";
        }
        cout << "size: " << m.size() << endl; // 0
}
```

Доступ к элементам

К любому элементу можно обратиться как к элементу массива. Достаточно указать его ключ внутри квадратных скобок. Можно как получить значение, так и изменить его. Если производится присваивание значения по ключу и ключа не существует, то производится вставка элемента, если элемент уже существует, то его значение изменяется на новое. Если производится получение значения по ключу и ключа не существует, то элемент вставляется со значением по умолчанию для типа, а затем это значение возвращается. Пример:

```
void main(){
      map< string, int > m;
      map< string, int >::iterator i;
      m["one"] = 1;       // Вставляем новый элемент
      m["two"] = 2;
      m["one"] = 100; // Изменяем значение существующего элемента
      cout << m["one"] << endl; //     100
      cout << m["two"] << endl; //     2
      cout << m["key"] << endl; //     0 (элемент вставлен)

      for (i=m.begin(); i!=m.end(); ++i)     {
            cout << i->first << " -> " << i->second << "; ";
      } // key -> 0; one -> 100; two -> 2;
}
```

Для доступа к элементам предназначены следующие методы:

1. `count(Keyval)` – возвращает количество элементов, у которых ключ соответствует значению `Keyval`.
2. `find(Keyval)` – возвращает итератор, установленный на элемент, ключ которого соответствует значению `Keyval`. Если элемент не найден, то метод возвращает итератор, указывающий на позицию после последнего элемента.

Пример 1. В следующей программе на примере ассоциативного контейнера, предназначенного для хранения десяти пар ключ/значение, иллюстрируются основы использования ассоциативных контейнеров. Ключом здесь является символ, а значением — целое. Пары ключ/значение хранятся следующим образом:

```
A    0
B    1
C    2
 . . .
J    9
```

Поскольку пары хранятся именно таким образом, то, когда пользователь набирает на клавиатуре ключ (т. е. одну из букв от `A` до `J`), программа выводит на экран соответствующее этому ключу значение.

```
#include <iostream>
#include <string.h>
```

```
#include <map>
using namespace std;

int main() {
	system("chcp 1251 > null");
	map<char, int> m;
	int i;
	// размещение пар в ассоциативном списке
	for (i = 0; i < 10; i++) {
		m['A'+i] = i;
	}
	char ch;
	cout << "Введите ключ: ";
	cin >> ch;
	map<char, int>::iterator p;
	// поиск значения по заданному ключу
	p = m.find(ch);
	if (p != m.end())
		cout << p->second;
	else
		cout << "Такого ключа в ассоциативном списке нет\n";

	cout << '\n';
	system("pause");
	return 0;
} ///:-
```

После того как ассоциативный контейнер инициализирован парами ключ/значение, найти нужное значение по заданному ключу можно с помощью функции `find()`. Функция `find()` возвращает итератор соответствующего ключу элемента или итератор конца ассоциативного контейнера, если указанный ключ не найден. Когда соответствующее ключу значение найдено, оно выводится в консоль.

Пример 2. Так же как и в других контейнерах, в ассоциативных контейнерах можно хранить создаваемые вами типы данных. Например, в представленной ниже программе создается отображение для хранения слов с соответствующими словам антонимами. С этой целью используются два класса: `word` (слово) и `opposite` (антоним). Поскольку для ассоциативных контейнеров поддерживается отсортированный список ключей, в программе для объектов типа `word` определяется оператор `<`. Как правило, оператор `<` необходимо перегружать для всех классов, объекты которых предполагается использовать в качестве ключей. (Помимо оператора `<` в некоторых компиляторах может потребоваться определить дополнительные операторы сравнения).

```
#include <map>
#include <iostream>
using namespace std;

class word {
	char str[20];
public:
	word() { strcpy_s(str, 20, ""); }
	word(char* s) { strcpy_s(str, 20, s); }
	word(const char* s) {strcpy_s(str, 20, s); }
	char* get() { return str; }
	// Для объектов типа word следует определить оператор < (меньше)
	bool operator< (word b) {
		return strcmp(get(), b.get()) < 0;
	}

};
// Для объектов типа word следует определить оператор < (меньше)
	bool operator< (word a, word b) {
```

```
        return strcmp(a.get(), b.get()) < 0;
}

class opposite {
        char str[20];
public:
        opposite() { strcpy_s(str, 20, ""); }
        opposite(const char* s) { strcpy_s(str, 20, s); }
        char* get() { return str; }
};

int main() {
        system("chcp 1251>nul");
        map<word, opposite> m;
        // Размещение в ассоциативном контейнере имен абонентов
        // и их телефонных номеров
        m[word("Василий")] = opposite("541-85-51");
        m[word("Иосиф")] = opposite("550-09-96");
        m[word("Михаил")] = opposite("8-3712-41-16-36");
        m[word("Никодим")] = opposite("8-095-967-85-85");
        // Поиск телефонного номера по заданному имени абонента
        char str[80];
        cout << "Введите имя: ";
        cin >> str;
        map<word, opposite>::iterator p;
        p = m.find(word(str));
        if (p != m.end())
                cout << "Телефонный номер: " << p->second.get();
        else
                cout
                << "Такого имени в ассоциативном контейнере нет\n";
                cout << '\n';
        system("pause");
        return 0;
}
```

В данном примере любой объект, который вводится в ассоциативный контейнер, представляет собой символьный массив для хранения заканчивающейся нулем строки.

Пример 3. Поскольку класс `string` определяет тип данных, появляется возможность создавать контейнеры для хранения объектов типа `string`. Например, ниже представлена усовершенствованная версия программы создания ассоциативного списка для хранения слов и антонимов.

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
 int main()  {
        system("chcp 1251 > null");
        map<string, string> m;
        int i;
        m[string("да")] = string("нет");
        m[string("хорошо")] = string("нет");
        m[string("влево")] = string("вправо");
        m[string("вверх")] = string("вниз");
        string s;
        cout << "Введите слово: ";
        cin >> s;
        map<string, string>:: iterator p;
        p = m.find(s);
        if (p != m.end())
                cout << "Антоним: " << p->second;
        else
                cout << "Такого слова в ассоциативном списке нет\n";
```

```
        cout << '\n';
        system("pause");
        return 0;
}
```

Методика и порядок выполнения работы. Перед выполнением практической работы каждый студент получает индивидуальные задания. Демонстрация выполнения практической работы происходит только после выполнения всех индивидуальных заданий. Порядок выполнения работы:

1. Проработать примеры, приведенные в лабораторной работе.
2. При выполнении заданий 1, 2, 3 написать программу с использованием строковых переменных типа `string`. Номер варианта - номер студента по списку в журнале.
3. При выполнении заданий 4, 5, 6, 7, 8 написать программу с использованием контейнера `vector`. Номер варианта – остаток от деления на 10 номера студента по списку в журнале плюс 1.

Индивидуальное задание №1.

Варианты заданий:

1. Пользователь вводит два текста (две переменные). Вычислить количество предложений в каждом из них.
2. Пользователь вводит два слова по 8 символов (две переменные). Сколько раз во втором слове встречается последняя буква первого слова.
3. Пользователь вводит текст. Определить в каких позициях в тексте встречаются символы «;» (другими словами: вывести на экран номера позиций, в которых встречается символ «;»)
4. Пользователь вводит текст. Переставить в нем первую букву первого слова и первую букву последнего слова (сначала найти номер последнего пробела).
5. Пользователь вводит текст. Определить в каких позициях в нем начинается каждое новое предложение (сначала найти позиции точек).
6. Пользователь вводит текст. Переставить в нем первую букву первого предложения и первую букву последнего предложения (сначала найти номер последней точки без учета точки в конце всего текста).
7. Пользователь вводит два слова (две переменные). Сколько раз в первом слове встречается третья буква второго слова.
8. Пользователь вводит текст. Сколько раз в нем встречается символ «=».
9. Пользователь вводит текст. Определить в каких позициях в тексте начинается каждое новое слово (сначала найти позиции пробелов).
10. Пользователь вводит два текста (две переменные). В каком из них больше слов? При условии, что слова разделяются только одним пробелом (сначала найти количество пробелов в каждом тексте).
11. Пользователь вводит текст. Переставить в нем первую букву первого слова и первую букву второго слова (сначала найти номер первого пробела).
12. Пользователь вводит два слова (две переменные). Сколько раз в первом слове встречается последняя буква второго слова.
13. Пользователь вводит текст. Сколько раз в нем встречается символ «@».
14. Пользователь вводит текст. Вывести на экран номера позиций в которых встречается символ «@».
15. Пользователь вводит 3 слова - Имя, Отчество, Фамилия в 3-х разных переменных. Образовать новую символьную переменную, хранящую только ваши инициалы (через точку и пробел).

16. Пользователь вводит два слова (две переменные). Сколько раз во втором слове встречается третья буква первого слова.
17. Пользователь вводит 3 слова в 3-х разных переменных. Образовать новую последовательность символов, состоящую из последних букв каждого слова (слитно без пробелов).
18. Пользователь вводит 3 слова в 3-х разных переменных. Образовать новую последовательность символов, состоящую из первых букв каждого слова (слитно без пробелов).
19. Пользователь вводит 3 слова - ваши Имя, Отчество, Фамилия в 3-х разных переменных. Образовать новую символьную переменную, хранящую полностью «имя отчество фамилию».
20. Пользователь вводит 2 слова. Образовать новую символьную переменную, в которой должны чередоваться буквы первого и второго слова.
21. Пользователь вводит 3 строки - фамилия, имя и отчество учащегося. Образовать новую последовательность, оставить только фамилию и инициалы через пробел и точку.
22. Пользователь вводит 4 слова в 4-х разных переменных. Образовать новую последовательность символов, состоящую из вторых букв каждого слова (слитно).
23. Пользователь вводит 3 слова в 3-х разных переменных. Образовать новую последовательность символов, состоящую из последних букв каждого слова (через пробел).
24. Пользователь вводит 3 слова в 3-х разных переменных. Образовать новую последовательность символов, состоящую из первых букв каждого слова (через пробел).
25. Пользователь вводит текст. Переставить в нем первую букву первого предложения и первую букву второго предложения (сначала найти номер первой точки).
26. Пользователь вводит 3 слова в 3-х разных переменных. Образовать новую символьную переменную, хранящую все три слова через пробел.
27. Пользователь вводит 3 слова в 3-х разных переменных. Образовать новую символьную переменную, хранящую все три слова через запятую.
28. Даны 3 слова в 3-х разных переменных. Образовать новую символьную переменную, хранящую все три слова через запятую и пробел.
29. Пользователь вводит 3 слова - ваши Имя, Отчество, Фамилия в 3-х разных переменных. Образовать новую символьную переменную, хранящую только ваши инициалы (через точку и пробел).
30. Пользователь вводит два слова (две переменные). Сколько раз во втором слове встречается первая буква первого слова.

Индивидуальное задание №2.

Варианты заданий:

1. Пользователь вводит некоторую последовательность символов. Образовать новую последовательность, включив в нее символы исходной, кроме символов пробелов, точек и запятых.
2. Пользователь вводит некоторую последовательность символов. Образовать новую последовательность, включив в нее символы исходной, кроме символов пробелов и скобок.
3. Пользователь вводит некоторую последовательность символов. Образовать последовательность символов, включив в нее символы данной последовательности, расположенные на четных позициях.
4. Пользователь вводит некоторый текст. Переписать в другую переменную только цифры.

5. Пользователь вводит некоторый текст. Образовать последовательность символов, включив в нее символы данной последовательности, расположенные на нечетных позициях.
6. Пользователь вводит некоторую последовательность символов. Образовать новую последовательность, удвоив каждый символ «=» и пропустив пробелы.
7. Пользователь вводит некоторую последовательность символов. Образовать новую последовательность, пропустив пробелы.
8. Пользователь вводит некоторый текст. Переписать в другую переменную все символы за исключением цифр и символов арифметических операций.
9. Пользователь вводит некоторый текст. Переписать в другую переменную только все символы за исключением цифр.
10. Пользователь вводит некоторый текст. Образовать новую последовательность, включив в нее символы исходной, кроме точек.
11. Пользователь вводит некоторый текст. Образовать новую последовательность, включив в нее символы исходной, кроме запятых.
12. Пользователь вводит некоторый текст. Образовать из него новый, в который включить информацию, заключенную между пробелом и запятой.
13. Пользователь вводит строка - фамилия, имя и отчество учащегося. Вывести на экран преобразованную строку: оставить только фамилию и инициалы
14. Пользователь вводит некоторый текст. Образовать новую последовательность, включив в нее символы исходной в обратном порядке.
15. Пользователь вводит некоторый текст из символов латинского алфавита, содержащий букву «а». Напечатать все символы, расположенные за первой буквой «а» до ее второго вхождения или до конца текста.
16. В предложении, вводимом с клавиатуры в одну переменную, поменять местами первое и последнее слова.
17. С клавиатуры вводится слово. Определить, является ли оно «перевертышем», т.е. читается одинаково слева направо и справа налево.
18. Пользователь вводит некоторый текст. Образовать из него новый, в который включить информацию, заключенную между первым пробелом и первой точкой.
19. Пользователь вводит некоторый текст, содержащий пару квадратных скобок. Создать новый текст, включив в него текст, заключенный в квадратные скобки.
20. Вводится строка - фамилия, имя и отчество учащегося. Вывести на экран преобразованную строку: оставить только фамилию и имя.
21. Пользователь вводит 2 слова в 2-х разных переменных одинаковой длины. Образовать новую последовательность, в которой должны чередоваться буквы первого и второго слова.
22. Пользователь вводит некоторый текст, содержащий символ «=». Напечатать все символы, расположенные за первым вхождением «=» до его второго вхождения или до конца текста.
23. Пользователь вводит некоторый текст. Переписать в другую переменную только цифры и символы арифметических операций (использовать склейку «+») (не использовать find, erase, substr…)
24. В предложении, вводимом с клавиатуры в одну переменную, поменять местами первое и последнее слова (не использовать find) (использование функции substr( ))
25. Пользователь вводит некоторый текст. Создать новый текст, включив в него только латинские буквы и цифры (не использовать find)
26. Пользователь вводит некоторый текст, содержащий пару фигурных скобок. Создать новый текст, включив в него текст заключенный в фигурные скобки (не использовать find)

27. Пользователь вводит некоторый текст, содержащий скобки. Поменять местами первое и последнее слово заключенное в скобки (не использовать find) (использование функции substr())
28. Пользователь вводит некоторый текст. Образовать новую последовательность, включив в нее символы исходной, кроме символов «g» и «v» (использовать склейку «+») (не использовать find, erase, substr…)
29. Пользователь вводит некоторый текст. Переписать в другую переменную только буквы латинского алфавита и пробелы (использовать склейку «+») (не использовать find, erase, substr…)

Индивидуальное задание №3.

Для тестирования программы создать текстовый файл в любом редакторе (например, в блокноте) для тестирования программы и поместить его в каталог проекта. Написать программу, согласно варианту (вариант определяется по списку преподавателя). Оформить обработку строки в виде функции. Произвести обработку каждой строки исходного файла, используя полученную функцию. Полученный результат вывести на экран.

Варианты заданий:

1. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, заменяя все маленькие латинские буквы на большие.
2. Дан текстовый файл. Записать в один новый файл только буквы из исходного файла и пробелы, в другой новый файл только цифры и пробелы.
3. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, заменяя "старый" символ на "новый" символ ("старый" символ и "новый" символ запрашиваются у пользователя).
4. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, удалив лишние пробелы.
5. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, заменяя все большие латинские буквы на маленькие.
6. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, заменяя все цифры на '*'.
7. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, заменяя все нецифры на '*'.
8. Дан текстовый файл. Запишите в другой файл содержимое исходного файла кроме цифр.
9. Дан текстовый файл. Запишите в один файл из исходного файла только большие латинские буквы, в другой новый файл - только малые латинские буквы и посчитайте количество цифр в исходном файле.
10. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, кроме латинских букв.
11. Из заданного входного файла считать символы и записать в новый файл все символы за исключением символов пунктуации: точки, запятые, двоеточия и т.д.
12. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, нумеруя каждую строку.
13. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, в начале каждой строки добавив “ – “.
14. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, чтобы каждое слово исходного файла было в отдельной строке.
15. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, утроив точку.

16. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, каждую строку заключив в скобки.
17. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, вставив после каждой строки пустую строку.
18. Дан текстовый файл. Запишите в другой файл только такие строки исходного файла, если они начинаются с цифры.
19. Дан текстовый файл. Запишите в другой файл только такие строки исходного файла, если они заканчиваются восклицательным знаком.
20. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, удвоив пробелы.
21. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, утроив восклицательный знак.
22. Дан текстовый файл. Запишите в другой файл только такие строки исходного файла, если они начинаются с “PS:”
23. Дан текстовый файл. Запишите в другой файл только такие строки исходного файла, которые содержат введенное с клавиатуры слово.
24. Дан текстовый файл. Запишите в другой файл содержимое исходного файла, в начале каждой строки добавив “e-mail: “.
25. Дан текстовый файл. Запишите в другой файл только такие строки исходного файла, если они содержат кавычки.
26. Дан текстовый файл. Найдите в исходном файле строки, в которых есть три звездочки и текст после них. Запишите в другой файл текст после звездочек.
27. Дан текстовый файл. Запишите в другой файл только такие строки исходного файла, если они содержат символ @.
28. Дан текстовый файл. Запишите в другой файл только такие строки исходного файла, которые не содержат чисел.

Индивидуальное задание №4.

Варианты заданий:

1. Создайте вектор `vector<float>` и занесите в него 25 вещественных чисел в цикле `for`. Выведите содержимое вектора. Удалить из вектора элемент с индексом 1 и снова распечатать вектор. Переделать программу: печатать вектор с помощью функции.
2. Создайте три объекта `vector<float>` и заполните первые два объекта так, как в предыдущем примере. Напишите цикл `for`, который суммирует соответствующие элементы первых двух векторов и заносит результат в соответствующий элемент третьего вектора. Выведите содержимое всех трех векторов. Переделать программу: печатать векторы с помощью функции.
3. Создайте вектор `vector<float>` и занесите в него 25 вещественных чисел. Возведите каждое число в квадрат и сохраните результат в исходном элементе вектора. Выведите содержимое вектора до и после возведения в квадрат. Удалить из вектора элемент с индексом 5 и снова распечатать вектор. Переделать программу: печатать вектор с помощью функции.
4. Создать вектор из 5-ти целых случайных чисел (0 - 100). Распечатать. Удвоить каждое число в векторе. Распечатать. Удалить из вектора элемент с индексом 2 и снова распечатать вектор. Переделать программу: удвоение сделать с помощью функции и печатать вектор с помощью функции.
5. Создать вектор из 20-ти логических случайных чисел (0 или 1). Распечатать. Посчитать количество истинных и ложных значений в векторе. Распечатать. Удалить из вектора первые десять элементов и распечатать полученный вектор. Переделать программу: печатать вектор с помощью функции.
6. Написать программе для ввода с клавиатуры массива строк (окончание ввода строк – пустая строка), которые записать в вектор. Распечатать введенный массив строк в

столбик с указанием номера каждой строки. Удалить из вектора элемент с индексом 3 и снова распечатать вектор. Переделать программу: печатать вектор с помощью функции.

7. Создать вектор из 6-ти вещественных случайных чисел от (-100 до 100). Распечатать. Посчитать сумму всех элементов массива. Удалить из вектора элемент с индексом 2 и снова распечатать вектор. Переделать программу: печатать вектор с помощью функции.
8. Создать вектор из 6-ти вещественных случайных чисел от (-100 до 100). Распечатать. Из первого вектора создать второй вектор, который содержит только отрицательные элементы первого вектора и распечатать его. Удалить из первого вектора элемент с индексом 4 и снова распечатать вектор. Переделать программу: печатать векторы с помощью функции.
9. Создать вектор из 10-ти вещественных случайных чисел (-50 до 50). Распечатать. Посчитать среднее число вектора. Из первого вектора создать второй вектор, который содержит только те элементы первого вектора, которые больше среднего первого массива. Удалить из первого вектора элемент с индексом 8 и снова распечатать вектор. Переделать программу: печатать векторы с помощью функции.
10. Создайте два вектора для хранения имен абонентов и их телефонных номеров. Имена и номера телефонов должны вводиться пользователем. После окончания ввода распечатать имена и телефонные номера абонентов в виде строк: имя абонента – его номер. Выполнить поиск номера по имени абонента. Удалите найденный номер и имя абонента из векторов. Снова распечатайте векторы. Переделать программу: печатать векторы с помощью функции.

Индивидуальное задание №5.

Варианты заданий:

1. Ниже представлен пример класса `Point`. Напишите программу для хранения объектов типа `Point` в векторе. (Подсказка: не забудьте для класса `Point` определить операторы `<`, `<<` и `==`)

```
class Point {
public:
  double x, y;
  Point() { x = y = 0; }
  Point(double a, double b) { x = a; y = b; }
};
```

Создать массив точек с помощью контейнера `vector` в `main()` и сразу занести в него информацию о 6 точках: (1.2, 6.3), (4.0, 0.7), (7.2, 0.8), (5.3, 3.0), (4.9, 6.6), (9.3, 0.2). Вывести массив на экран таким образом, чтобы координаты каждой точки выводились в отдельной строке. Выведите на экран координаты точки, которая наиболее удалена от центра координат. Сдвиньте все точки влево по оси абцисс на расстояние, которое введет пользователь и снова выведите массив на экран. Напишите функцию вывода массива на экран.

2. Создать класс с именем `Planet` для хранения следующей информации:
название (`string`), количество спутников.
Создать массив планет с помощью контейнера `vector` в `main()`.
Занести в вектор информацию о 9 планетах. Данные о планетах:
`"Меркурий", 0`
`"Венера", 0`
`"Земля", 1`
`"Марс", 2`
`"Юпитер", 69`

```
"Сатурн", 62
"Уран", 27
"Нептун", 14
"Плутон", 10
```

Вывести массив планет на экран, используя функцию.
В цикле определите элемент, у которого максимальное количество спутников.

3. Создать класс:

```
class Card {
   string title;        // заглавие книги
   string author;       // автор
   int number;          // количество имеющихся экземпляров
public:
   Card() : title(""), author(""), number(0) {}
   Card(string t, string a, int n) : title(t), author(a), number(0) {}
       };
```

В программе создать вектор из объектов класса `Card` (5 элементов).
Распечатать вектор. Пользователь вводит автора, программа выводит книги автора или сообщение об их отсутствии.

4. Составить описание класса `Complex` для представления комплексных чисел с возможностью задания вещественной и мнимой частей числами типа `double`. В программе создать вектор из объектов класса `Complex` (6 элементов): (-1.2, 6.3), (4.0, 0.7), (7.2, -0.8), (5.3, 3.0), (-4.9, 6.6), (-9.3, 0.2).
Распечатать вектор в виде:

```
-1.2 + i * 6.3
 4.1 + i * 0.7
 7.2 - i * 0.8
 5.3 + i * 3
-4.9 + i * 6.6
-9.3 + i * 0.2
```

Сложите все числа (у комплексных чисел отдельно складываются действительные и мнимые части) и результирующее число выведите на экран.

5. Ниже представлен пример класса `Rectangle` (прямоугольник). Напишите программу для хранения объектов типа `Rectangle` в векторе. (Подсказка: не забудьте для класса `Rectangle` определить операторы `<` и `==`)

```
class Rectangle {
       double a;     // ширина
       double b;     // длина
public:
       Rectangle() : a(0), b(0) {}
       Rectangle(double a, double bi) : a(a), b(b){}
};
```

Создать массив объектов типа `Rectangle` с помощью контейнера `vector` в `main()` и сразу занести в него информацию о 6 объектах: (1.2, 6.3), (4.0, 0.7), (7.2, 0.8), (5.3, 3.0), (4.9, 6.6), (9.3, 0.2). Вывести массив на экран таким образом, чтобы размеры каждого прямоугольника выводились в отдельной строке. Выведите на экран размеры прямоугольника, у которого наибольшая площадь. Напишите функцию вывода массива на экран.

6. Создайте класс с именем `Date`, содержащий три закрытых поля типа `int`, предназначенные для хранения дня, месяца и года. (Подсказка: не забудьте для класса `Date` определить операторы `<` и `==`). Один из конструкторов класса должен инициализировать поля датой 01.01.1970, а другой конструктор — заданным набором

значений. Создайте метод класса, который будет выводить значения полей на экран в формате 11.05.2019.
Создать массив объектов типа `Date` с помощью контейнера `vector` в `main()` и сразу занести в него информацию о шести датах: (1, 2, 1963), (14, 7, 1995), (7, 12, 2088), (5, 3, 2030), (24, 9, 2013), (19, 9, 2020). Вывести массив на экран таким образом, чтобы каждая дата выводились в отдельной строке. Создайте новый вектор, в который запишите даты первого вектора, которые относятся к будущему. Выведите на экран второй вектор. Напишите функцию вывода массива на экран.

7. Создайте класс с именем `Time`, содержащий три закрытых поля типа `int`, предназначенные для хранения часов, минут и секунд. (Подсказка: не забудьте для класса `Time` определить операторы `<` и `==`). Один из конструкторов класса должен инициализировать поля нулевыми значениями, а другой конструктор — заданным набором значений. Создайте метод класса, который будет выводить значения полей на экран в формате 11:59:59.
Создать массив объектов типа `Time` с помощью контейнера `vector` в `main()` и сразу занести в него информацию о шести датах: (1, 2, 63), (14, 57, 19), (7, 32, 20), (5, 13, 23), (10, 19, 45), (19, 9, 59). Вывести массив на экран таким образом, чтобы каждая дата выводились в отдельной строке. Создайте новый вектор, в который запишите даты первого вектора, которые относятся к ночи (с 0 до 5 часов). Выведите на экран второй вектор. Напишите функцию вывода массива на экран.

8. Создать класс `Inventory` для учета товаров на складе. Класс содержит следующие закрытые компоненты: `string item` – наименование товара, `double cost` – стоимость, `int on_hand` – количество. (Подсказка: не забудьте для класса `Inventory` определить операторы `<` и `==`). Один из конструкторов класса должен инициализировать поля нулевыми значениями, а другой конструктор — заданным набором значений.
Создать массив объектов типа `Inventory` с помощью контейнера `vector` в `main()` и сразу занести в него информацию о шести объектах: `("Отверка",99, 0), ("Молоток",430, 10), ("Гайки",70, 100), ("Профиль",540, 0), ("Уголок",230, 9), ("Доска",350, 17)`. Вывести массив на экран таким образом, чтобы каждый товар выводился в отдельной строке:

```
Отверка - 99 - 0
Молоток - 430 - 10
Гайки - 70 - 100
Профиль - 540 - 0
Уголок - 230 - 9
Доска - 350 - 17
```

Создайте новый вектор, в который объекты первого вектора, у которых количество равно нулю. Выведите на экран второй вектор. Напишите функцию вывода массива на экран.

9. Ниже представлен пример класса `Box` (коробка). Напишите программу для хранения объектов типа `Box` в векторе. (Подсказка: не забудьте для класса `Box` определить операторы `<` и `==`)

```
class Box {
        double a;     // ширина
        double b;     // высота
        double c;     // длина
public:
        Box () : a(0), b(0), c(0) {}
        Box (double a, double b, double c) : a(a), b(b), c(c){}
};
```

Создать массив объектов типа `Box` с помощью контейнера `vector` в `main()` и сразу занести в него информацию о 6 объектах: (1, 2, 63), (14, 57, 19), (7, 32, 20), (5, 13, 23), (10, 19, 45), (19, 9, 59). Вывести массив на экран таким образом, чтобы размеры каждой коробки выводились в отдельной строке. Выведите на экран размеры коробки, у которой наибольший объем. Напишите функцию вывода массива на экран.

10. Ниже представлен пример класса `Graduate` (выпускник). Напишите программу для хранения объектов типа `Graduate` в векторе. (Подсказка: не забудьте для класса `Graduate` определить операторы `<` и `==`)

```
class Graduate {
	string name;   // фамилия
	double rating; // рейтинг
public:
	Graduate() {
		name = "";
		rating = 0;
	}
	Graduate(string Name, double Rating) {
		name = Name;
		rating = Rating;
	}
};
```

Создать массив объектов с помощью контейнера `vector` в `main()` и сразу занести в него информацию о 6 объектах: ("Иванов",99), ("Петров",430), ("Семенов",70), ("Котов",540), ("Белых",230), ("Черных",350). Вывести массив на экран таким образом, чтобы характеристики каждого объекта выводились в отдельной строке. Выведите на экран характеристики объекта, который имеет максимальный рейтинг. Увеличьте рейтинг каждого выпускника на число, которое введет пользователь и снова выведите массив на экран. Напишите функцию вывода массива на экран.

Индивидуальное задание №6.

Варианты заданий:

1. Описать функцию, которая добавляет после каждого элемента заданного контейнера-списка `list <int>` еще один такой же элемент, но с обратным знаком, а затем исключает из списка все отрицательные элементы и распечатывает результат. Переделать программу: печатать список до изменения и после с помощью функции.
2. Описать функцию, которая считает количество положительных элементов заданного контейнера-списка `list <int>`, а затем распечатывает это значение (выдает в стандартный поток `cout`). Продемонстрировать её работу.
3. Описать функцию, которая печатает «Yes» или «No» в зависимости от того, содержится ли заданное целое число x в заданном контейнере-списке `list <int>`.
4. Создать вектор из 5-ти целых случайных чисел (0 - 100). Распечатать. Удвоить каждое число в векторе. Распечатать. Удалить из вектора элемент с индексом 2 и снова распечатать вектор. Переделать программу: удвоение сделать с помощью функции и печатать вектор с помощью функции.
5. Создать вектор из 20-ти логических случайных чисел (0 или 1). Распечатать. Посчитать количество истинных и ложных значений в векторе. Распечатать. Удалить из вектора первые десять элементов и распечатать полученный вектор. Переделать программу: печатать вектор с помощью функции.
6. Написать программе для ввода с клавиатуры массива строк (окончание ввода строк – пустая строка), которые записать в вектор. Распечатать введенный массив строк в

столбик с указанием номера каждой строки. Удалить из вектора элемент с индексом 3 и снова распечатать вектор. Переделать программу: печатать вектор с помощью функции.

7. Создать вектор из 6-ти вещественных случайных чисел от (-100 до 100). Распечатать. Посчитать сумму всех элементов массива. Удалить из вектора элемент с индексом 2 и снова распечатать вектор. Переделать программу: печатать вектор с помощью функции.
8. Создать вектор из 6-ти вещественных случайных чисел от (-100 до 100). Распечатать. Из первого вектора создать второй вектор, который содержит только отрицательные элементы первого вектора и распечатать его. Удалить из первого вектора элемент с индексом 4 и снова распечатать вектор. Переделать программу: печатать векторы с помощью функции.
9. Создать вектор из 10-ти вещественных случайных чисел (-50 до 50). Распечатать. Посчитать среднее число вектора. Из первого вектора создать второй вектор, который содержит только те элементы первого вектора, которые больше среднего первого массива. Удалить из первого вектора элемент с индексом 8 и снова распечатать вектор. Переделать программу: печатать векторы с помощью функции.
10. Создайте два вектора для хранения имен абонентов и их телефонных номеров. Имена и номера телефонов должны вводиться пользователем. После окончания ввода распечатать имена и телефонные номера абонентов в виде строк: имя абонента – его номер. Выполнить поиск номера по имени абонента. Удалите найденный номер и имя абонента из векторов. Снова распечатайте векторы. Переделать программу: печатать векторы с помощью функции.

Индивидуальное задание №7. Выполнить задания, используя контейнер `list`.

Варианты заданий:

1. Создать класс с именем `Planet` для хранения следующей информации: название (`string`), количество спутников.
   Создать массив планет с помощью контейнера `list` в `main()`.
   Занести в вектор информацию о 9 планетах. Данные о планетах:
   ```
   "Меркурий", 0
   "Венера", 0
   "Земля", 1
   "Марс", 2
   "Юпитер", 69
   "Сатурн", 62
   "Уран", 27
   "Нептун", 14
   "Плутон", 10
   ```
   Вывести массив планет на экран, используя функцию.
   В цикле определите элемент, у которого максимальное количество спутников.

2. Создать класс:
   ```
   class Card {
          string title;        // заглавие книги
          string author;       // автор
          int number;          // количество имеющихся экземпляров
   public:
          Card() : title(""), author(""), number(0) {}
          Card(string t, string a, int n) : title(t), author(a), number(0) {}
   };
   ```
   В программе создать `list` из объектов класса `Card` (5 элементов).

Распечатать список. Пользователь вводит автора, программа выводит книги автора или сообщение об их отсутствии.

3. Составить описание класса `Complex` для представления комплексных чисел с возможностью задания вещественной и мнимой частей числами типа `double`. В программе создать список из объектов класса `Complex` (6 элементов): (-1.2, 6.3), (4.0, 0.7), (7.2, -0.8), (5.3, 3.0), (-4.9, 6.6), (-9.3, 0.2).
   Распечатать список в виде:
   ```
   -1.2 + i * 6.3
    4.1 + i * 0.7
    7.2 - i * 0.8
    5.3 + i * 3
   -4.9 + i * 6.6
   -9.3 + i * 0.2
   ```
   Сложите все числа (у комплексных чисел отдельно складываются действительные и мнимые части) и результирующее число выведите на экран.

4. Ниже представлен пример класса `Rectangle` (прямоугольник). Напишите программу для хранения объектов типа `Rectangle` в `list`. (Подсказка: не забудьте для класса `Rectangle` определить операторы `<` и `==`)
   ```
   class Rectangle {
         double a;     // ширина
         double b;     // длина
   public:
         Rectangle() : a(0), b(0) {}
         Rectangle(double a, double bi) : a(a), b(b){}
   };
   ```
   Создать список объектов типа `Rectangle` с помощью контейнера `list` в `main()` и сразу занести в него информацию о 6 объектах: (1.2, 6.3), (4.0, 0.7), (7.2, 0.8), (5.3, 3.0), (4.9, 6.6), (9.3, 0.2). Вывести список на экран таким образом, чтобы размеры каждого прямоугольника выводились в отдельной строке. Выведите на экран размеры прямоугольника, у которого наибольшая площадь. Напишите функцию вывода списка на экран.

5. Создайте класс с именем `Date`, содержащий три закрытых поля типа `int`, предназначенные для хранения дня, месяца и года. (Подсказка: не забудьте для класса `Date` определить операторы `<` и `==`). Один из конструкторов класса должен инициализировать поля датой 01.01.1970, а другой конструктор — заданным набором значений. Создайте метод класса, который будет выводить значения полей на экран в формате 11.05.2019.
   Создать список объектов типа `Date` с помощью контейнера `list` в `main()` и сразу занести в него информацию о шести датах: (1, 2, 1963), (14, 7, 1995), (7, 12, 2088), (5, 3, 2030), (24, 9, 2013), (19, 9, 2020). Вывести список на экран таким образом, чтобы каждая дата выводились в отдельной строке. Создайте новый список, в который запишите даты первого списка, которые относятся к будущему. Выведите на экран второй список. Напишите функцию вывода списка на экран.

6. Создайте класс с именем `Time`, содержащий три закрытых поля типа `int`, предназначенные для хранения часов, минут и секунд. (Подсказка: не забудьте для класса `Time` определить операторы `<` и `==`). Один из конструкторов класса должен инициализировать поля нулевыми значениями, а другой конструктор — заданным набором значений. Создайте метод класса, который будет выводить значения полей на экран в формате 11:59:59.

Создать список объектов типа `Time` с помощью контейнера `list` в `main()` и сразу занести в него информацию о шести датах: (1, 2, 63), (14, 57, 19), (7, 32, 20), (5, 13, 23), (10, 19, 45), (19, 9, 59). Вывести список на экран таким образом, чтобы каждая дата выводились в отдельной строке. Создайте новый список, в который запишите даты первого списка, которые относятся к ночи (с 0 до 5 часов). Выведите на экран второй список. Напишите функцию вывода списка на экран.

7. Создать класс `Inventory` для учета товаров на складе. Класс содержит следующие закрытые компоненты: `string item` – наименование товара, `double cost` – стоимость, `int on_hand` – количество. (Подсказка: не забудьте для класса `Inventory` определить операторы `<` и `==`). Один из конструкторов класса должен инициализировать поля нулевыми значениями, а другой конструктор — заданным набором значений.
   Создать список объектов типа `Inventory` с помощью контейнера `list` в `main()` и сразу занести в него информацию о шести объектах: `("Отверка",99, 0), ("Молоток",430, 10), ("Гайки",70, 100), ("Профиль",540, 0), ("Уголок",230, 9), ("Доска",350, 17)`. Вывести список на экран таким образом, чтобы каждый товар выводился в отдельной строке:

```
Отверка - 99 - 0
Молоток - 430 - 10
Гайки - 70 - 100
Профиль - 540 - 0
Уголок - 230 - 9
Доска - 350 - 17
```

   Создайте новый список, в который объекты первого списка, у которых количество равно нулю. Выведите на экран второй список. Напишите функцию вывода списка на экран.

8. Ниже представлен пример класса `Box` (коробка). Напишите программу для хранения объектов типа `Box` в списке. (Подсказка: не забудьте для класса `Box` определить операторы `<` и `==`)

```
class Box {
        double a;     // ширина
        double b;     // высота
        double c;     // длина
public:
        Box () : a(0), b(0), c(0) {}
        Box (double a, double b, double c) : a(a), b(b), c(c){}
};
```

   Создать массив объектов типа `Box` с помощью контейнера `list` в `main()` и сразу занести в него информацию о 6 объектах: (1, 2, 63), (14, 57, 19), (7, 32, 20), (5, 13, 23), (10, 19, 45), (19, 9, 59). Вывести список на экран таким образом, чтобы размеры каждой коробки выводились в отдельной строке. Выведите на экран размеры коробки, у которой наибольший объем. Напишите функцию вывода списка на экран.

9. Ниже представлен пример класса `Graduate` (выпускник). Напишите программу для хранения объектов типа `Graduate` в списке. (Подсказка: не забудьте для класса `Graduate` определить операторы `<` и `==`)

```
class Graduate {
        string name;    // фамилия
        double rating; // рейтинг
public:
        Graduate() {
                name = "";
```

```
            rating = 0;
        }
        Graduate(string Name, double Rating) {
            name = Name;
            rating = Rating;
        }
};
```

Создать список объектов с помощью контейнера `list` в `main()` и сразу занести в него информацию о 6 объектах: ("Иванов",99), ("Петров",430), ("Семенов",70), ("Котов",540), ("Белых",230), ("Черных",350). Вывести список на экран таким образом, чтобы характеристики каждого объекта выводились в отдельной строке. Выведите на экран характеристики объекта, который имеет максимальный рейтинг. Увеличьте рейтинг каждого выпускника на число, которое введет пользователь и снова выведите список на экран. Напишите функцию вывода списка на экран.

10. Ниже представлен пример класса `Point`. Напишите программу для хранения объектов типа `Point` в списке. (Подсказка: не забудьте для класса `Point` определить операторы `<` и `==`)

```
class Point {
public:
    double x, y;
    Point() { x = y = 0; }
    Point(double a, double b) { x = a; y = b; }
};
```

Создать массив точек с помощью контейнера `list` в `main()` и сразу занести в него информацию о 6 точках: (1.2, 6.3), (4.0, 0.7), (7.2, 0.8), (5.3, 3.0), (4.9, 6.6), (9.3, 0.2). Вывести список на экран таким образом, чтобы координаты каждой точки выводились в отдельной строке. Выведите на экран координаты точки, которая наиболее удалена от центра координат. Сдвиньте все точки влево по оси абцисс на расстояние, которое введет пользователь и снова выведите список на экран. Напишите функцию вывода списка на экран.

Индивидуальное задание №8.

Варианты заданий:

1. Создайте отображение `map<char,int>` и занесите в него пары A-1, B-2, C-3, D-4, E-5. Выведите содержимое отображения на экран. Удалить пару с ключом C и снова распечатать отображение.
2. Создайте отображение `map<int,double>` и занесите в него пары 1 - 1.1, 2 - 2.2, 3 - 3.3, 4 - 4.4, 5 - 5.5 . . . 9 - 9.9. Выведите содержимое отображения на экран. Удалить пары с четными ключами и снова распечатайте отображение.
3. Создайте отображение `map<int, int >` и занесите в него пары 1 - 10, 2 - 20, 3 - 30, . . . 20 - 200. Выведите содержимое отображения на экран. Удалить пары с нечетными ключами и снова распечатайте отображение.
4. Создайте отображение `map<string, int >` и занесите в него пары `"one"-100, "two"-200, "three"-300, . . . "six"-600`. Выведите содержимое отображения на экран. Удалить пары с ключами `"five" "six"` и снова распечатайте отображение.
5. Занести в отображение информацию о планетах: <название, количество спутников> . Данные о планетах:

```
"Меркурий", 0
"Венера", 0
```

```
"Земля", 1
"Марс", 2
"Юпитер", 69
"Сатурн", 62
"Уран", 27
"Нептун", 14
"Плутон", 10
```

Вывести отображение на экран. В цикле определите элемент, у которого максимальное количество спутников.

6. Создать отображение `Inventory`, которое содержит данные о названии товара и количестве его на складе. Занести в него информацию: `("Отверка", 0), ("Молоток", 10), ("Гайки", 100), ("Профиль", 0), ("Уголок", 9), ("Доска", 17)`. Вывести отображение на экран таким образом, чтобы каждый товар выводился в отдельной строке, и только такой товар, у которого количество не равно нулю:

```
Гайки - 100
Доска – 17
Молоток - 10
Уголок - 9
```

7. Создайте отображение `map<string,int>` и занесите в него пары занести в него информацию о 6 объектах: ("Иванов",99), ("Петров",430), ("Семенов",70), ("Котов",540), ("Белых",230), ("Черных",350). Выведите содержимое отображения на экран. Увеличьте значение каждой пары на единицу и снова распечатайте отображение.
8. Создайте отображение `map<char,int>` , в котором ключ – буква, значение – её код. Заполните отображение пятью парами. Выведите содержимое отображения на экран. Удалить вторую пару и снова распечатайте отображение.
9. Создайте отображение `map<string, string>` , в котором ключ – страна, значение – её столица. Заполните отображение пятью парами. Пользователь вводит страну, если такой ключ есть в контейнере, то на экран выводится название столицы, если – нет, то пользователю предлагается ввести столицу и если он её вводит, то в отображение добавляется новая пара. По окончании ввода выведите содержимое отображения на экран.
10. Создайте отображение `map<int, double >` и занесите в него пары 1 – 100.0, 2 – 200.0, 3 – 300.0, . . . 9 – 900.0. Выведите содержимое отображения на экран. Вычислите сумму значений в контейнере. Каждое значение разделите на полученную сумму и вновь выведите отображение.

Индивидуальное задание №9.

Варианты заданий:

1. Даны два массива:

```
string states[] = { "Wyoming", "Colorado", "Nevada", "Montana", "Arizona", "Idaho"};
int pops[] = { 470, 2890, 800, 787, 2718, 944 };
```

На основе этих массивов создайте отображение, в котором ключ – это название американского штата, а значение – количество жителей этого штата в тысячах из массива `pops`. Напишите программу: пользователь вводит название штата, а в консоль выводится количество жителей этого штата. Выведите всё содержимое отображения в консоль с помощью функции.

2. В отображении находятся записи о соответствии номеров телефона (7 цифр) и фамилии. Например:

```
5671234 Ivanov
```

```
3214567 Petrov
9871234 Sidorov
```

В массиве находятся номера телефонов. Например:

```
string mas[4] = { "1112233", "9871234", "5671234", "5556688" };
```

Осуществить поиск фамилий из заданного отображения, соответствующих телефонам из массива `mas`. Найденные фамилии вывести на консоль. В данном примере:

```
Sidorov
Ivanov
```

3. В первом отображении находится информация о зарплате сотрудников. Например:

   Ivanov 45500
   Petrov 37000
   Sidorov 245000
   Petrenko 65000
   Tovalds 91200
   Popov 54600
   Andrienko 35000

   Во втором отображении принадлежность сотрудников к отделам предприятия. Например:

   Ivanov Research
   Petrov Research
   Sidorov Management
   Petrenko Management
   Tovalds Development
   Popov Research
   Andrienko Sales

   Необходимо вычислить общий фонд зарплаты и отдельно по отделам. Результат вывести на экран. В данном случае (возможны опечатки):

   Research 137100
   Management 310000
   Development 91200
   Sales 35000
   Overall: 573300

   Примечание: Для решения этой задачи необходимо завести ассоциативный массив (std::map), где ключом будет название отдела (строка), а значением – текущая сумма зарплат. В цикле перебирать пары первого отображения, необходимо находить по ключу отдел во втором отображении и прибавлять к имеющейся там (в ассоциативном массиве) сумме только что считанную.

4. Создайте ассоциативный контейнер для хранения имен абонентов и их телефонных номеров. Имена и номера телефонов должны вводиться пользователем. После окончания ввода распечатать имена и телефонные номера абонентов в виде строк: имя абонента – его номер. Выполнить поиск номера по имени абонента, которое введет пользователь. Удалите найденный номер и имя абонента из отображения. Снова распечатайте отображение. Переделать программу: печатать отображение с помощью функции.

5. Разработайте программу, в которой используется отображение, которое содержит перечень городов и расстояние каждого города от Москвы:

```
'Минск' : 713,
'Киев' : 856,
'Санкт-Петербург' : 786,
```

```
'Астана' : 2748,
'Нижний Новгород' : 421,
'Владивосток' : 9141
```

Найти в отображении ближайший город к Москве и самый дальний и выведите в консоль с пояснением. Распечатать содержимое отображения в консоли, используя функцию.

6. Дано отображение с данными о количестве учащихся в разных классах школы:
   ```
   {'1а': 21, '1б':20, '2':25, '3а': 20, '3б': 23, '4а':25, '4б':25, '5а':
   26, '5б':25, '6а':19, '6б': 18, '7а':26, '7б':25, '8':25, '9':17, '10':
   15, '11’: 14}
   ```
   В программе определить количества учащихся в школе. Распечатать содержимое отображения в консоли, используя функцию.

7. Необходимо объединить два прайс-листа (задаются в виде отображений) с тем условием, что если наименование товара присутствует в обоих прайсах, то в итоговый прайс помещается тот, чья цена выше.
   ```
   m1 = {'яблоки':100, 'груши':13, 'арбузы':20, 'картофель':15, 'алыча':22}
   m2 = {'яблоки':150,'груши':18,'ананасы':45, 'апельсины':30, 'киви':35}
   ```
   Распечатать содержимое отображений в консоли, используя функцию.

8. У вас есть выгрузка оставшегося количества фруктов из базы данных магазинов. Каждое отображение - магазин. Какие-то фрукты есть в разных магазинах, какие-то только в одном. Нужно сделать отображение, в котором ключами будут фрукты, а их количество - значениями. Если фрукт есть в наличии в нескольких магазинах - сложите эти значения.

   ```
   Shop1 = {  'банан' :  400,  'яблоко' :  300, … }
   Shop2 = {  'банан' : 750,  'груша' : 20, 'яблоко' : 150 … }
   ```

9. Дано отображение, содержащее сведения о количестве изделий, собранных сборщиками цеха за неделю:
   ```
   'Иванов': 123, 'Петров': 89, 'Сидоров': 108, 'Краснов': 132,
   'Демин': 99, 'Черепанов': 111
   ```
   Каждая пара содержит поля: фамилия сборщика - количество изделий, собранных им ежедневно в течение недели. Количество записей - произвольное. Сколько всего собрали сборщики цеха за неделю. Распечатать содержимое отображений в консоли, используя функцию.

10. Дано отображение с данными о количестве учащихся в разных классах школы:
    ```
    {'1а': 21, '1б':20, '2':25, '3а': 20, '3б': 23, '4а':25, '4б':25, '5а':
    26, '5б':25, '6а':19, '6б': 18, '7а':26, '7б':25, '8':25, '9':17, '10':
    15, '11’: 14}
    ```
    В программе пользователь вводит год обучения школьников, а консоль выводится количество учащихся заданного года обучения, например:

    ```
    Год обучения: 1
    Количество: 41
    ```

    Распечатать содержимое отображения в консоли, используя функцию.

Приложение1. Коды основных символов в кодировке ASCII

Таблица 6.1. *Коды основных символов в кодировке ASCII*

dec	oct	hex	символ
0	0	0	\0
7	7	7	\a
8	10	8	\b
9	11	9	\t
10	12	a	\n
11	13	b	\v
12	14	c	\f
13	15	d	\r
32	40	20	пробел
33	41	21	!
34	42	22	"
35	43	23	#
36	44	24	$
37	45	25	%
38	46	26	&
39	47	27	'
40	50	28	(
41	51	29	)
42	52	2a	*
43	53	2b	+
44	54	2c	,
45	55	2d	-
46	56	2e	.

dec	oct	hex	символ
76	114	4c	L
77	115	4d	M
78	116	4e	N
79	117	4f	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5a	Z
91	133	5b	[
92	134	5c	\
93	135	5d	]
94	136	5e	^
95	137	5f	_
96	140	60	`
97	141	61	a
98	142	62	b

47	57	2f	/
48	60	30	0
49	61	31	1
50	62	32	2
51	63	33	3
52	64	34	4
53	65	35	5
54	66	36	6
55	67	37	7
56	70	38	8
57	71	39	9
58	72	3a	:
59	73	3b	;
60	74	3c	<
61	75	3d	=
62	76	3e	>
63	77	3f	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G

99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6a	j
107	153	6b	k
108	154	6c	l
109	155	6d	m
110	156	6e	n
111	157	6f	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7a	z
123	173	7b	{

72	110	48	H
73	111	49	I
74	112	4a	J
75	113	4b	K

124	174	7c	|
125	175	7d	}
126	176	7e	~
127	177	7f	DEL

Справочные материалы по спискам

Конструкторы класса list:

list	Constructs a list of a specific size or with elements of a specific value or with a specific **allocator** or as a copy of some other list.

Определения typedefs list:

allocator_type	A type that represents the **allocator** class for a list object.
const_iterator	A type that provides a bidirectional iterator that can read a **const** element in a list.
const_pointer	A type that provides a pointer to a **const** element in a list.
const_reference	A type that provides a reference to a **const** element stored in a list for reading and performing **const** operations.
const_reverse_iterator	A type that provides a bidirectional iterator that can read any **const** element in a list.
difference_type	A type that provides the difference between two iterators that refer to elements within the same list.
iterator	A type that provides a bidirectional iterator that can read or modify any element in a list.
pointer	A type that provides a pointer to an element in a list.
reference	A type that provides a reference to a **const** element stored in a list for reading and performing **const** operations.
reverse_iterator	A type that provides a bidirectional iterator that can read or modify an element in a reversed list.
size_type	A type that counts the number of elements in a list.
value_type	A type that represents the data type stored in a list.

Методы класса list:

assign	Erases elements from a list and copies a new set of elements to the target list.
back	Returns a reference to the last element of a list.
begin	Returns an iterator addressing the first element in a list.
clear	Erases all the elements of a list.

empty	Tests if a list is empty.
end	Returns an iterator that addresses the location succeeding the last element in a list.
erase	Removes an element or a range of elements in a list from specified positions.
front	Returns a reference to the first element in a list.
get_allocator	Returns a copy of the **allocator** object used to construct a list.
insert	Inserts an element or a number of elements or a range of elements into a list at a specified position.
max_size	Returns the maximum length of a list.
merge	Removes the elements from the argument list, inserts them into the target list, and orders the new, combined set of elements in ascending order or in some other specified order.
pop_back	Deletes the element at the end of a list.
pop_front	Deletes the element at the beginning of a list.
push_back	Adds an element to the end of a list.
push_front	Adds an element to the beginning of a list.
rbegin	Returns an iterator addressing the first element in a reversed list.
remove	Erases elements in a list that match a specified value.
remove_if	Erases elements from the list for which a specified predicate is satisfied.
rend	Returns an iterator that addresses the location succeeding the last element in a reversed list.
resize	Specifies a new size for a list.
reverse	Reverses the order in which the elements occur in a list.
size	Returns the number of elements in a list.
sort	Arranges the elements of a list in ascending order or with respect to some other order relation.
splice	Removes elements from the argument list and inserts them into the target list.

swap	Exchanges the elements of two lists.
unique	Removes adjacent duplicate elements or adjacent elements that satisfy some other binary predicate from the list.