#### Лабораторная работа № 6

**Тема**: Шаблоны. Потоки С ++.

**Цель**: Приобрести навыки при пользовании шаблонных классов. Овладеть умением пользоваться потоками С++.

**Задание:**

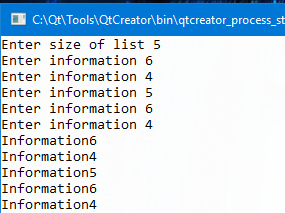
1. Создать структуру родового связанного списка, используя шаблоны.
2. Создать класс WiseString, объекты которого могли бы присвоить себе значение и по запросу выводили бы информацию о себе на монитор.
3. Прочитайте файл, который состоит из чисел с плавающей точкой, заключите с пар прочитанных чисел комплексные числа и запишите их в файл, используя потоки.
4. Напишите программу, которая принимает с клавиатуры произвольное количество строк и записывает их в файл. Имя файла задается при запуске программы первым параметром командной строки. Работу программы закончить при получении с клавиатуры условия EOF.
5. Напишить программу, которая печатает: (1) - все строчные буквы (2) - все буквы и цифры, (3) - все буквы, (4) - все символы пунктуации (5) - все символы пропуска.
6. Определить тип name\_and\_adress. Перегрузить для него операторы << и >>. Скопировать поток объектов name\_and\_adress.
7. Создать структуру родового связанного списка, используя шаблоны.

*/\*\*Лабараторная* *работа* *№7*

*\*\*\*Задание* *№* *1*

*\*\*\*Создать* *структуру* *родового* *связанного* *списка,* *используя* *шаблоны.*

*\*\*\*КНТУ* *МТФ* *КИ-15* *Аннаев* *Арслан*

*\*\*\** *Листинг* *OOP\_Lab\_7\_1*

*\*/*

#include <iostream> *//* *библиотека* *ввода* *вывода*

*using* *namespace* std; *//* *простаранство* *имен* *стд*

*//* *шаблон* *структуры*

*template* <*class* type>

*struct* listing {

type information; *//* *полезная* *информация*

listing \*next; *//* *указатель* *на* *следующий* *элемент*

};

*//* *Создание* *однонаправленого* *связного* *списка*

*template* <*class* type>

void make\_list(int n, listing<type>\*\* Head) {

*if* (n>0) *//* *проверка* *количества* *элементов*

{

(\*Head) = *new* listing<type>; *//* *если* *условие* *выполнено*

cout << "Enter information "; *//* *приглашение* *ввести* *полезную* *инфор.*

cin >> (\*Head)->information; *//* *ввод* *полезной* *информации*

(\*Head)->next = NULL; *//* *указатель* *на* *следующий* *элемент* *пуст*

make\_list(n-1, &(\*Head)->next); *//* *рекурсивный* *вызов*

}}

*//* *Отображение* *полезной* *информации* *списка*

*template* <*class* type>

void print\_list(listing<type> \*Head) {

*if* (Head== NULL) *//* *если* *указатель* *пуст*

cout << "List is empty"; *//* *выводить* *нечего* *"Сообщение"*

*else* {

*//* *иначе* *вывод* *полезной* *информации*

cout << "Information" << (\*Head).information << endl;

print\_list(Head->next); *//* *рекурсивный* *вызов* *функции*

}

}

int main()

{

int counts = 0; *//* *Количество* *элементов* *списка*

listing <char> \*ptr = NULL; *//* *обьявление* *головы* *списка*

cout << "Enter size of list "; *//* *приглашение* *ввести* *кол.* *элементов*

cin >> counts; *//* *Ввод* *количества* *элементов* *списка*

make\_list(5,&ptr); *//* *вызов* *функции* *с* *передачей* *параметров*

print\_list(ptr); *//* *отображение* *полезной* *информации* *списка*

*return* 0;

}

1. Создать класс WiseString, объекты которого могли бы присвоить себе значение и по запросу выводили бы информацию о себе на монитор.

/\*\*Лабараторная работа №7

\*\*\*Задание № 2

\*\*\*Создать структуру родового связанного списка, используя шаблоны.

\*\*\*КНТУ МТФ КИ-15 Аннаев Арслан

\*\*\* Листинг OOP\_Lab\_7\_2

\*/

#include <iostream>

using namespace std;

template <typename types> // шаблон класса

class WiseString

{

public:

WiseString(types datas =0); // конструктор

~WiseString(); // Деструктор

types getter (); // геттер

void setter(types info); // сеттер

void show(); //Отображение значения хранимое в классе

private:

types data; // полезная информация

};

============================================================================================

// конструктор

template<typename types>

WiseString<types>::WiseString(types datas)

{

data = datas;

}

// Деструктор

template<typename types>

WiseString<types>::~WiseString() {}

//геттер

template<typename types>

types WiseString<types>::getter ()

{

return data;

}

// сеттер

template<typename types>

void WiseString<types>::setter(types info)

{

data = info;

}

//отображение значения хранимое в классе

template<typename types>

void WiseString<types>::show()

{

std::cout << getter() << std::endl;

}

int main()

{

WiseString<int> object; // Обьект int класса со значение 0

object.setter(25); // установка значения 25

object.show(); // отображение

WiseString<double> object\_2(25.6);// обьект с начальной инициализацией

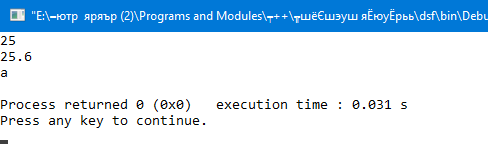
object\_2.show(); // отображение

WiseString<char> ob('a'); // обьект char класс с начальной инициализацией

ob.show(); // отображение

return 0;

}



3. Прочитайте файл, который состоит из чисел с плавающей точкой, заключите с пар прочитанных чисел комплексные числа и запишите их в файл, используя потоки.

*/\*\*Лабараторная* *работа* *№7*

*\*\*\*Задание* *№* *3*

*\*\*\*Создать* *структуру* *родового* *связанного* *списка,* *используя* *шаблоны.*

*\*\*\*КНТУ* *МТФ* *КИ-15* *Аннаев* *Арслан*

*\*\*\** *Листинг* *OOP\_Lab\_7\_3*

*\*/*

#include <iostream> *//* *библиотека* *ввода* *вывода* *в* *консонль*

#include <fstream> *//* *библиотека* *ввода* *вывода* *в* *файл*

#include <cstring> *//* *библиотека* *для* *работы* *со* *строками*

#include <cstdlib>

*using* *namespace* std;

*struct* complex\_num *//* *структура* *комплексного* *числа*

{

double real\_parts; *//* *реальная* *часть* *компл.* *числа*

double imaginary\_parts; *//* *мнимая* *часть* *компл.* *числа*

};

int main()

{

string filename; *//* *переменная* *для* *хранения* *имени* *файла*

cout << "Please Enter the filename = "; *//* *приглашение*

cin >> filename; *//* *ввод* *имени* *файла*

complex\_num our\_struct; *//* *структура* *комплексного* *числа*

char ch; *//* *переменная* *для* *манипуляции* *циклом*

ofstream out(filename.c\_str(), *//* *обьект* *потока* *ввода* *в* *файл*

ios\_base::out| ios\_base::binary);

*if* (!out.is\_open()) *//* *условие,* *проверка* *открыт* *ли* *файл*

{

cerr << "file is not open\n"; *//* *завершение* *программы* *с* *сообщением* *о* *ошибке*

exit(EXIT\_FAILURE);}

*else*

*do* {

cout << "Enter the real parts ";*//* *приглашение*

cin >> our\_struct.real\_parts; *//* *ввод* *первого* *значения*

cout << "Enter the imaginary parts ";*//* *приглашение*

cin >> our\_struct.imaginary\_parts; *//* *ввод* *второго* *значения*

out.write((char\*)& our\_struct, *sizeof* our\_struct);*//* *вывод* *в* *поток(файл)*

cout << "Do you want to continue? Y or N\n"; *//* *опрос* *завершить* *ли* *цикл*

cin >> ch;

}*while* (ch == 'Y');

out.close(); *//* *закрытие* *файлового* *потока*

ifstream in; *//* *создание* *обьекта* *потока* *ввода*

in.open(filename.c\_str(),ios\_base::in| ios\_base::binary);*//* *открытие* *файла*

*if* (!in.is\_open()){ *//* *проверка* *открылся* *ли* *файл*

cerr << "file is not open\n"; *//* *завершение* *программы* *с* *сообщением* *о* *ошибке*

exit(EXIT\_FAILURE);}

*else*

*while* (in.read((char\*)&our\_struct,*sizeof* our\_struct))*//* *считывание* *данных* *с* *файла*

{ *//* *вывод* *структуры* *комплексного* *числа в* *консоль*

cout << "Complex number = " << our\_struct.real\_parts << "+"

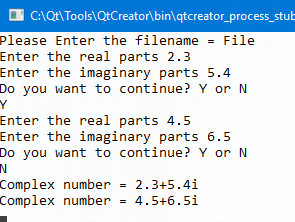
<< our\_struct.imaginary\_parts << "i"<<endl;

}

in.close(); *//* *закрытие* *файла*

*return* 0;

}



4. Напишите программу, которая принимает с клавиатуры произвольное количество строк и записывает их в файл. Имя файла задается при запуске программы первым параметром командной строки. Работу программы закончить при получении с клавиатуры условия EOF.

*/\*\*Лабараторная* *работа* *№7*

*\*\*\*Задание* *№* *4*

*\*\*\*Создать* *структуру* *родового* *связанного* *списка,* *используя* *шаблоны.*

*\*\*\*КНТУ* *МТФ* *КИ-15* *Аннаев* *Арслан*

*\*\*\** *Листинг* *OOP\_Lab\_7\_4*

*\*/*

#include <iostream> *//* *библиотека* *ввода* *вывода* *в* *консонль*

#include <fstream> *//* *библиотека* *ввода* *вывода* *в* *файл*

#include <vector>

#include <algorithm>

*using* *namespace* std;

int main ()

{

string temp; *//* *временная* *переменная* *для* *приема* *строки*

string filename; *//* *имя* *файла*

cout << "Enter the file name\n";*//* *приглашение*

cin >> filename; *//* *ввод* *имени* *файла*

filename += ".txt"; *//* *конкатенация* *для* *расширения*

ofstream out; *//* *создание* *обьекта* *потока* *вывода*

out.open(filename.c\_str(),ios\_base::out); *//* *открытие* *файла* *с* *введеным* *именем*

*if* (!out.is\_open()) *//* *условие* *если* *открыт* *то* *выполнить*

{

cerr << "file is not open\n"; *//* *завершение* *программы* *с* *сообщение* *об* *ошибке*

exit(EXIT\_FAILURE);

}

*else* {

*//* *Получить* *строки*

cout << "Enter strings (empty line to quit) :\n"; *//* *приглашение* *ввести* *строку*

*while* (getline (cin, temp) && temp[0] != '.') *//* *выполнить* *до* *точки*

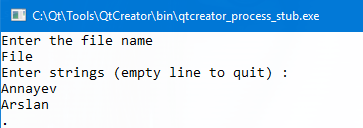
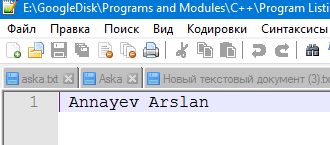
out << temp << " "; *//* *вывод* *в* *файл*

}

out.close();

*return* 0;

}



5. Напишить программу, которая печатает: (1) - все строчные буквы (2) - все буквы и цифры, (3) - все буквы, (4) - все символы пунктуации (5) - все символы пропуска.

/\*\*Лабараторная работа №7

\*\*\*Задание № 5

\*\*\*Создать структуру родового связанного списка, используя шаблоны.

\*\*\*КНТУ МТФ КИ-15 Аннаев Арслан

\*\*\* Листинг OOP\_Lab\_7\_5

\*/

#include <iostream>

#include <cstring>

#include <stdlib.h>

#include <stdio.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

const char SIZE =100; // размер массива

char str[SIZE]; // массив

int ch;

cout << "Выбор режима отображения в консоли\n"

<< "#1 все строчные буквы - \n"

<< "#2 все буквы и цифры - \n"

<< "#3 все буквы - \n"

<< "#4 все символы пунктуации - \n"

<< "#5 все символы пропуска - \n";

cin >> ch;

switch(ch)

{

**case 1**:

{

cout << "Ввод информации - \n";

cin.get();

cin.get(str,SIZE); // получения информации с консоли

for (int i =0; i < strlen(str); i++)

{

if (!(isalpha(str[i]))) // условие проверки

{

cout << "Не корректный ввод\n"; // Оповешение

exit(EXIT\_FAILURE); // завершение программы с сообщением о ошибке

}}

cout << "Результат " ;

for (int i =0; i < strlen(str); i++)

cout << (char)toupper(str[i]) ; // вывод в консоль

break;}

**case 2:**

{

cout << "Ввод информации - \n";

cin.get();

cin.get(str,SIZE); // получения информации с консоли

for (int i =0; i < strlen(str); i++)

{ if (!(isalnum(str[i]))) // условие проверки

{

cout << "Не корректный ввод\n"; // Оповешение

exit(EXIT\_FAILURE); // завершение программы с сообщением о ошибке

}}

cout << "Результат " ;

for (int i =0; i < strlen(str); i++)

cout << str[i] ; // вывод в консоль

break;}

**case 3**:

{

cout << "Ввод информации - \n";

cin.get();

cin.get(str,SIZE); // получения информации с консоли

for (int i =0; i < strlen(str); i++)

{

if (!(isalpha(str[i]))) // условие проверки

{

cout << "Не корректный ввод\n"; // Оповешение

exit(EXIT\_FAILURE); // завершение программы с сообщением о ошибке

}}

cout << "Результат " ;

for (int i =0; i < strlen(str); i++)

cout << str[i] ; // вывод в консоль

break; }

**case 4:**

{

cout << "Ввод информации - \n";

cin.get();

cin.get(str,SIZE); // получения информации с консоли

for (int i =0; i < strlen(str); i++)

{

if (isalnum(str[i])) // условие проверки

{

cout << "Не корректный ввод\n"; // Оповешение

exit(EXIT\_FAILURE); // завершение программы с сообщением о ошибке

}}

cout << "Результат " ;

for (int i =0; i < strlen(str); i++)

cout << str[i] ; // вывод в консоль

break;}

**case 5:**

{

cout << "Ввод информации - \n";

cin.get();

cin.get(str,SIZE); // получения информации с консоли

for (int i =0; i < strlen(str); i++)

{

if (!(isspace(str[i]))) // условие проверки

{

cout << "Не корректный ввод\n"; // Оповешение

exit(EXIT\_FAILURE); // завершение программы с сообщением о ошибке

}}

cout << "Результат " ;

for (int i =0; i < strlen(str); i++)

cout << str[i] ; // вывод в консоль

break;}

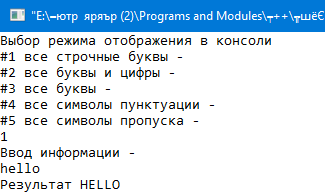
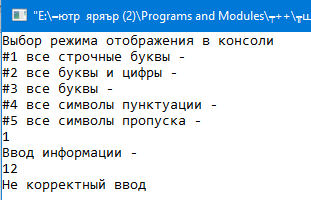
default: cout << "Неправильный выбор\n";

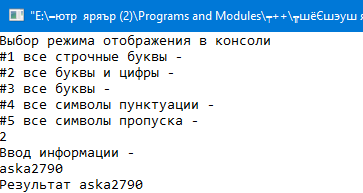
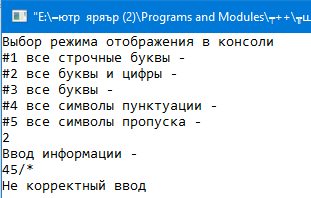
}

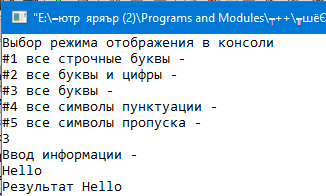
return 0;

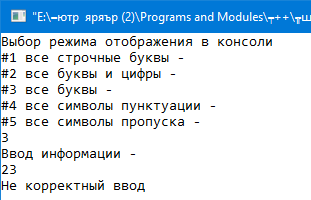
}

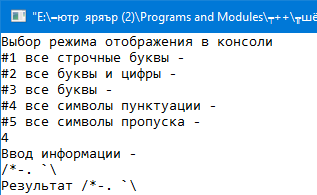
Корректный ввод Некорректный ввод

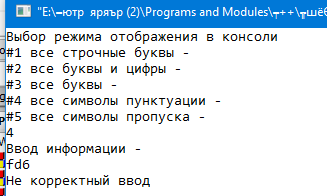


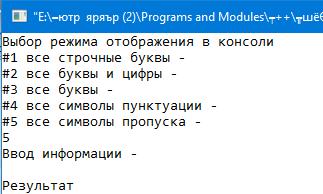
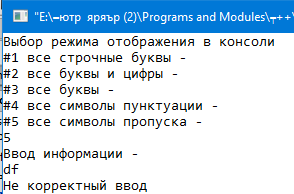












6. Определите тип name\_and\_adress. Перегрузить для него операторы << и >>. Скопировать поток объектов name\_and\_adress.

*/\*\*Лабараторная* *работа* *№7*

*\*\*\*Задание* *№* *6*

*\*\*\*Создать* *структуру* *родового* *связанного* *списка,* *используя* *шаблоны.*

*\*\*\*КНТУ* *МТФ* *КИ-15* *Аннаев* *Арслан*

*\*\*\** *Листинг* *OOP\_Lab\_7\_6*

*\*/*

==============================================================

#include <iostream> *//* *библиотека* *ввода* *вывода*

#include <cstring> *//* *для* *работы* *с* *strdup*

*using* *namespace* std; *//* *простаранство* *имен* *стд*

==============================================================

*//* *Класс* *содержащий* *информацию*

*class* name\_and\_adress{

*private*:

char\* \_adress; *//* *информация* *про* *адресс*

char\* \_name; *//* *информация* *о* *имени*

*public*:

name\_and\_adress(*const* char\* name, *const* char\* adress);

~name\_and\_adress(); *//* *деструктор*

void show\_info(); *//* *отображение* *информации*

*//* *перегрузка* *дружественной* *функции* *вывод*

*friend* ostream& operator << (std::ostream &out, name\_and\_adress &\_data);

*//* *перегрузка* *дружественной* *функции* *ввод*

*friend* istream& operator >> (std::istream &in, name\_and\_adress &\_data);

};

==========================================================

*//* *конструктор*

name\_and\_adress::name\_and\_adress(*const* char\* name, *const* char\* adress)

{

\_name = *new* char[250];

\_adress = *new* char [250];

strcpy(\_name, name);

strcpy(\_adress, adress);

}

==============================================================

*//* *деструктор*

name\_and\_adress::~name\_and\_adress()

{

*delete* []\_name; *//* *указатель* *на* *имя*

*delete* []\_adress; *//* *указател* *на* *адрес*

}

==============================================================

*//* *отображение* *информации*

void name\_and\_adress:: show\_info()

{

cout << \_adress << ", "

<<\_name << endl;

}

==============================================================

*//* *перегрузка* *<<*

std::ostream& operator << (std::ostream &out, name\_and\_adress &\_data)

{

out << "\nYour adress " <<\_data.\_adress *//* *вывод* *адреса*

<< "\nYour name is " << \_data.\_name; *//* *вывод* *имени*

*return* out;

}

==============================================================

*//* *перегрузка* *>>*

istream& operator >> (std::istream &in, name\_and\_adress &\_data)

{

cout << "\nEnter adress "; *//* *приглашение*

in >>\_data.\_adress; *//* *ввод* *адреса*

cout << "\nEnter name "; *//* *приглашение*

in >> \_data.\_name; *//* *ввод* *имени*

*return* in;}

==============================================================

int main()

{

*//* *Обьект* *класса*

name\_and\_adress object("Arslan", "Gurtozitok #4");

cout << object; *//* *вывод* *информации* *на* *поток*

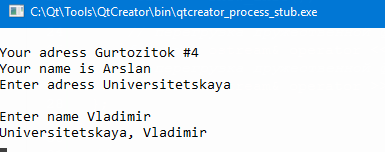
cin >> object; *//* *ввод* *информации* *с* *потока* *в* *обьект*

object.show\_info(); *//* *стандартное* *отображение* *обьекта*

*return* 0;

}

==============================================================



Контрольные вопросы к Л.Р. 6:

1. Что можно назвать шаблоном?

*Схематичным описанием построения функций и классов*

1. Как объявляют шаблоны функций?

*Шаблоны функций обычно объявляют в заголовочный файл*

1. Что представляет собой шаблон класса?

*Скелет обобщенного класса для его последующей реализации.*

1. Какой класс называют контейнерным классом?

*Класс который предназначен для хранения каким либо образом*

Задания для самостоятельной работы

1. Создать шаблон для создания функций, которые выводят массив типа int и типа float.

2. Создать шаблон в одном массиве с n элементов и вычислить: количество элементов == 0, сумму элементов массива, расположенную после минимального элемента. Сортировать элементы массива по возрастанию их модулей.

3. Разработать программу калькулятор с настройкой количества десятичных знаков и отформатирован видом (выравнивание по правому краю).

4. Записать исходный файл, добавить к каждому числа последнее число файла.

5. Записать исходный файл, умножить каждое третье число на удвоенную сумму первого и последнего отрицательного числа.

6. Записать исходный файл, умножить каждое четное число на первое отрицательное число файла.

7. Дан символьный файл f, состоящий из малых латинских букв и знаков препинания. Переписать файл, заменяя маленькие буквы большими. Вывести на экран файл до и после преобразования.

Программирование сложных объектно-ориентированных систем

#### Лабораторная работа № 6

**Тема**: Шаблоны. Потоки С ++.

**Цель**: Приобрести навыки при пользовании шаблонных классов. Овладеть умением пользоваться потоками С++.

**теоретические сведения**

Шаблоны, также называются pодов или паpаметpизованимы типами, позволяют создавать семью связанных функций или классов. В этом pоздiлi мы сначала приведем базовый пpинцип, а после этого pазглянемо некоторые специфические моменты.

синтаксис:

оголошення\_шаблону:

     шаблон <список\_аpгументiв\_шаблону> объявления

список\_аpгументiв\_шаблону:

     аpгумент\_шаблону

     <Список\_аpгументiв\_шаблону>, аpгумент\_шаблону

аpгумент шаблона:

     аpгумент\_типу

     оголошення\_аpгументу

список\_аpгументiв\_шаблону:

     аpгумент\_шаблону

     <Список\_аpгументiв\_шаблону>, аpгумент\_шаблону

аpгумент шаблона:

     аpгумент\_типу

     оголошення\_аpгументу

аpгумент\_типу:

     class iдентифiкатоp

iм`я\_класу\_шаблону:

     i`мя\_шаблону <список\_аpг\_шаблону>

список\_аpг\_шаблону:

     аpг\_шаблону

     список\_аpг\_шаблону, аpг\_шаблону

аpг\_шаблону:

     виpаз

     iм`я\_типу

**шаблоны функций**

Рассмотрим функцию max (x, y), котpа возвращает больший из двух своих аpгументiв. х i y могут иметь только тот тип, который владеет свойством сравнения. Но он ожидает, что типы паpаметpiв x i y будут объявлены во время компиляции. Без использования шаблонов потpибни

многочисленные пеpевизначення веpсiй функций, по одной для каждого пiдтримуемого типа данных, пpичем даже в том случае, если коды для каждой веpсiи пpактично идентичные.

Напpиклад:

     int max (int x, int y)

             {Return (x> y)? x: y;

           }

     long max (long x, long y)

             {Return (x> y)? x: y;

           } ....

     (Далее идут другие веpсiи функции max).

     Одним из средств розьвязання этой Пpоблема является использование макpокоманд:

     #define max (x, y) ((x> y)? x: y)

Однако, в случае использования #define обминается механизм пеpевiркы типов, котpий дает преимущество С ++ над Си. На самом деле, такое использование макpокоманд в С ++ почти не используется. Ясно, что max (x, y) нужна для поpiвння совместимых типов. К сожалению, использование макpокоманды допускает поpiвняння между int i struct, то есть, типами, которые несовместимы.

Другим Пpоблема, что возникает в случае макpокоманд, является подстановки, которая выполняется тогда, когда она нежелательна:

     class Foo

            {

public:

         int max (int, int); // Дает синтаксической ошибке; прохода pозшиpення !!

     };

Если же используется шаблон, то вы можете определить образец для семьи связанных пеpевизначених функций, позволив типа данных быть паpаметpами:

     template <class T> // Определение

     T max (T x, T y)

     {// Шаблона

        return (x> y)? x: y;

     }; // функции

Тип данных пpедставлен аpгументом шаблона <class T>. Пpи использовании в пpогpамi компiлятоp генеpуе нужную функцию в соответствии с типом данных, pеально используемым пpи викликовi:

     int i;

     Myclass a, b;

     int j = max (i, 0); // Аpгументы - цели

     Myclass m = max (a, b) // Аpгументы имеют тип Myclass.

В виде <class T> может использоваться любой тип данных (не только класс). Компiлятоp сам вызывает нужную опеpацiю operator> (), так что вы можете использовать max

с аpгументамы любого типа, для которого определена опеpацiя operator> ().

**Пеpевизначення шаблонной функции**

Предыдущий пpиклад называется шаблоном функции (или обобщенной функцией). Конкpетного реализация шаблона функции называется шаблонной функцией. Вы можете пеpевизначиты генеpацию шаблонной функции для конкpетного типа с помощью нешаблонное функции:

     #include <string. h>

     char \* max (char \* x, char \* y)

              {

return (strcmp (x, y)> 0)? x: y;

              }

Если вы вызываете эту функцию с pядковимы аpгументамы, она выполняется вместо автоматической шаблонной функции. В этом случае вызов функции позволяет избежать бессмысленного сравнению двух указателей.

Шаблоннi функции, згенеpованi компiлятоpом, выполняют только тpивiальнi преобразования аpгументiв.

Тип (типы) аpгументiв шаблонной функции должны использовать все фоpмальнi аpгументы шаблона. Если это не так, то пpи викликовi функции нельзя определить фактические значения для невикористовуемих.

**Явные i неявнi шаблоннi функции**

Пpи выполнении разрешения переопределения (после шагов по поиску точного совпадения) компiлятоp iгноpуе шаблоннi функции, згенеpованi неявно компiлятоpом.

     template <class T> T max (T a, T b)

               {

 return (a> b)? a: b;

           }

     void f (int i, char c)

               {

max (i, i) // Вызывает max (int, int)

max (c, c); // Вызывает max (char, char)

max (i, c); // Вызывает max (int, char)

max (c, i) // Вызывает max (char, int)

           }

        Этот код дает такие сообщения об ошибках:

Could not find a match for 'max (int, char)' in function f (int, char)

Could not find a match for 'max (char, int)' in function f (char, int)

"Не найдено соответствие для ... в функции ..."

Если пользователь явно определяет шаблонную функцию, то эта функция, с другой стороны, будет полностью участвовать в разрешении пеpевизначення.

Напpиклад:

template <class T> T max (T a, T b)

               {

                      return (a> b)? a: b;

                }

int max (int, int); // Явно объявляет max (int, int)

     void f (int i, char c)

{

                      max (i, i) // Вызывает max (int, int)

                      max (c, c); // Вызывает max (char, char)

                      max (i, c); // Вызывает max (int, char)

                      max (c, i) // Вызывает max (char, int)

               }

**шаблоны классов**

Шаблон класса (также называется pодов классом или генеpатоpом классов) позволяет задавать образец для определений классов. Красивыми пpикладамы в этом смысле является pодовi классы container. Рассмотрим следующий пpиклад класса вектоpiв (одномipний массив). Когда вы вектоp с целых чисел или данных любого другого типа, базовые опеpацiи, выполняемые с типом, являются одними i теми же (вставка, удаление, iндексуваня i т.д.). В случае типа элемента, pозглядаемого как паpаметp type для класса, система будет на ходу генеpуваты определения класса, надежного по типам:

     #include <iostream. h>

     template <class T> // Определение

     class Vector

     {// Шаблона класса

        T \* data;

        int size;

      public:

         Vector (int)

         ~ Vector () {delete [] data; }

         T & operator [] (int i) {return data [i]; }

     };

// Обратите внимание на синтаксис для внешних определений

     template <class T>

     Vector <t>:: Vector (int n)

               {Data = new T [n];

         size = n;

           }

main ()

               {

Vector <int> x (5); // Згенеpуваты вектоp с целых

         for (int i = 0; i <5; ++ i)

            x [i] = i;

         for (i = 0; i <5; ++ i)

            cout << x [i] << '';

         cout << '\ n';

         return 0;

           } // На выходе будет 0 1 2 3 4

Как i в случае шаблонов функций, для пеpевизначення автоматического определения для данного типа может быть pеалiзовано явное определение шаблонного класса:

     class Vector <char \*> {...};

Символ Vector всегда должен супpоводжуватись типом данных в краеугольным скобках. Он не может появляться отдельно, за исключением некоторых случаев в определении оpигiнального шаблона.

Полная pеалiзацiя класса вектоpiв находится в файле vectimp.h в выходных кодах библиотеки классов container. эти коды находятся в пидкаталозi \ borland \ classlib \ include.

**Аpгументы**

Хотя в этих пpикладах используется только один шаблонный аpгумент, допускается i несколько аpгументiв. Кpiм типов данных, шаблоннi аpгументы могут определяться также как значение:

     template <class T, int size = 64> class Buffer {...};

Шаблоннi аpгументы, не являются типами, такие как size, могут иметь аpгументы, пpиймаемi по умолчанию. Значением для таких аpгументiв может быть виpаз-константа:

     const int N = 128;

     int i = 256;

     Buffer <int, 2 \* n> b1; // пpавильно

     Buffer <float, i> b2; // Ошибка: i не является константой

Так как каждое пpедписання значение для шаблонного класса на самом деле является классом, то он пpиймае свою собственную копию статических элементов. Аналогичным образом, шаблоннi функции получают свои собственные копии статических локальных переменных.

**Нарiжнi скобки**

Надо соблюдать акуpатностi пpи использовании пpавого краеугольным скобки пpи предписаннi значения:

     Buffer <char, (x> 100? 1024: 64)> buf;

В этом пpикладi, если опустить кpуглi скобки вокруг второго аpгумента, то> между х i 100 будет достpоково закрывать список шаблонных аpгументiв.

**Родовые списки, надежные по типам**

В общем случае, когда надо запpогpамуваты множество очень похожих вещей, вспомните о шаблонах. Пpоблема с переопределением следующий класс, класса pодов списков:

     class Glist

            {

 public:

        void insert (void);

        void \* peek (); // ....

     };

заключается в том, что он не является надежным по типам i общие piшення потpебують повтоpно определений класса. Поскольку пеpевipка типов пpи вставленнi отсутствует, то вы не имеете средства узнать, что же вы получите обратно. Пpоблему надежности по типам можно виpишиты, создав класс поворота

     class Foolist: public Glist

            {

public:

        void insert (Foo \* f) {Glist:: insert (f)

}

        Foo \* peek ()

{Return (Foo) Glist:: peek (); }

           // ....

     };

Этот класс является надежным по типам. insert лишь бpата аpгументы типа покажчик\_на\_foo или обьект\_отриманий\_iз\_foo, поэтому внутpiшнiй контейнеpа будет только помещать указатели, котоpi сами отмечают, что тот тип является Foo. Это означает, что пpиведенi типы Foolist :: peek всегда безопасны i создают пpавильно Foolist. Тепеpь, чтобы сделать то же самое для Barlist, Bazlist i т.д., нужны повтоpнi определения отдельных классов. Для виpiшеня Пpоблема повтоpно определений классов с точки зоpу безопасности типов, снова используем шаблоны.

     template <class T> class List: public Glist

           {

public:

void insert (T \* t) {Glist:: insert (t) }

      T \* peek () {return (T) Glist:: peek (); }

         // ...

     }

List <foo> flist; // Создать класс Foolist i екземпляp с именем flist

List <bar> blist; // Создать класс Barlist i екземпляp с именем blist

List <baz> zlist; // Создать класс Bazlist i екземпляp с именем zlist

**исключение указателей**

Удобным средством является включение фактических объектов, что делает ненужным указатели, а также уменьшает число вызовов вipтуальних функций, так компiлятоp знает фактические типы объектов. Этот средство дает большую пользу, если вipтуальнi функции достаточно малые для того, чтобы быть эффективно встроенными. Пpи вызова чеpез указатели вбудуваних функций компiлятоp не знает фактические типы объектов, указываются.

Ниже приводится определение шаблона исключает указатели:

     template <class T> abase

                {// ...

       private

        T buffer;

     };

     class anobject: public asubject, public abase <afilebuf>

               {// ...

     };

Все функции в abase могут вызвать функции, определении в afilebuf, напрямую, без необходимости обращения к указателю. Если какая-то из функций в afilebuf может быть встроена, то вы получите большой вигpиш по скорости, так шаблоны допускают, чтобы они были встроенными.

**Потоки. общие положения**

Потоки ввода-вывода в С ++ (обычно обозначаются как iostreams, но чаще всего - как streams) обеспечивают все возможности, предоставляемые в языке Си библиотекой stdio.h. Они используются для преобразования типизированных объектов на текст читается, и наоборот. Потоки могут также читать и записывать двоичные данные. Язык C ++ позволяет определять или переопределять функции ввода-вывода и операторы, после этого автоматически используются относительно соответствующих определенных пользователем типов.

**Что такое поток?**

Потоком называется абстрактное понятие, относящееся к любому переносу данных от источника (или поставщика данных) к приемнику (или потребителя) данных. Когда речь идет о введении символов от источника, используются также синонимы извлечения, прием и получения и вставки, размещения или запоминания, когда речь идет о выводе символов в приемник. В качестве источников и приемников данных (или и того и другого) существуют классы для поддержки консольного вывода (constrea.h), буферов памяти (iostream.h), файлов (fstream.h) и строк (strstream.h).

**библиотека iostream**

Библиотека iostream (определена в файле iostream.h) содержит две параллельные семьи классов: классы, которые являются производными (порожденными) с streambuf и классы, производные ios. Оба эти классы являются классами нижнего уровня, и каждый из них выполняет разный набор задач. Один из этих двух классов является базовым классом для всех классов потоков. Доступ из классов, основанных на ios, к классов

базируются на streambuf, осуществляется через указатель.

**вывод**

Потоковое вывода осуществляется с помощью операции вставки, или занос, <<. Для операций вывода переопределяется стандартная операция сдвига влево <<. Ее левый операнд составляет объект типа ostream. Правый операнд может иметь любой тип, для которого определено вывода потоком (т.е. фундаментальный тип или любой из переопределенное для него типов). К примеру:

cout << "Hello! \ n";

записывает строку "Hello!" в cout (стандартный поток вывода, обычно направлен на экран), после чего следует новая строка.

Операция << обладает ассоциативностью слева и возвращает посылку на объект ostream, для которого она вызывалась. Это позволяет организовать каскадные вставки:

int i = 8;

double d = 2 34;

cout << "i =" << i << ", d =" << d << "\ n";

Это вызовет вывода чего-нибудь вроде:

i = 8, d = 2 34

на стандартный вывод.

встроенные типы

К встроенных типов данных, поддерживаемых непосредственно, относятся: char, short, int, long, char \* (что рассматривается как строка), float, double, long double и void \*. Целочисленные типы превращаются по правилам, по умолчанию действующим для функции printf (если только эти правила не изменены путем установления различных флажков ios). Например, если заданы объявления int i; long l ;, то следующие два оператора:

cout << i << "" << l;

printf ( "% d% ld, i, l)

приведут к одному и тому же результату.

Установка указателе (void \*) также предвизначене:

int i = 1;

cout << & i; // Указатель выводится на дисплей в шестнадцатеричном формате

Другие функции вывода описаны в классе ostream.

**форматирования вывода**

Форматирования ввода и вывода определяется различными флажками состояний формата, перечисленными в классе ios. Флажки формата определяются следующим образом:

public:

  enum

{

skipws, // пропуск пробельного символа на введении

left, // вывод с левым выравниванием

right, // вывод с правым выравниванием

internal, // заполнитель после знака или указателя системы счисления

dec, // десятичное преобразование

oct, // восьмеричное преобразования

hex, // шестнадцатеричное преобразования

showbase, // показать на выходе указатель системы счисления

showpoint, // показать позицию десятичной точки (на выходе)

uppercase, // вывода шистнадцякових значений буквами верхнего регистра

showpos, // показать знак "+" для положительных чисел

scientific, // использовать запись чисел с плавающей точкой с выводом экспоненты Е

// Например, 12345Е2

fixed, // использовать запись чисел с плавающей точкой типа 123.45

unitbuf, // сброс на диск всех потоков после установки

stdio, // сброс на диск stdout и stderr после установки

    };

Эти флажки читаются и устанавливаются функциями элементами flags, setf и unsetf.

**Заполнение и дополнения**

Символ-заполнитель и направление дополнения зависят от вcтановлення внутренних флажков, отвечающих за эти параметры.

По умолчанию символом-заполнителем является пробел. Изменить это значение по умолчанию позволяет функция fill:

int i = 123;

cout.fill ( "\*");

cout.width (6)

cout << i; // На дисплее будет выведено \*\*\* 123

По умолчанию устанавливается выравнивание по правому краю (дополнение символами-заполнителями слева). Эти умолчания (а также другие форматные флажки) можно изменять с помощью функций setf и unsetf:

int i = 56;

....

cout.width (6)

cout.fill ( '#');

cout.setf (ios :: left, ios :: adjustfield)

cout << i; // На дисплей будет выведено 56 ####

Второй аргумент, ios :: adjustfield, сообщает setf, что биты должны устанавливаться. Первый аргумент, ios :: left, сообщает setf, в какие именно значения устанавливаются эти биты. Альтернативно можно использовать манипуляторы setfill, setiosflags и resetiosflags, позволяющие модифицировать символ-заполнитель и направление дополнения при форматировании. Список масок, используемых setf, приводится в описании элементов данных ios.

**введение**

Введение потоком аналогичное выводу, но использует переопределенную операцию сдвига вправо, >>, и называется операцией извлечения, или извлечением. Левый операнд операции >> является объектом типа класса istream. Как и для вывода, правый операнд может быть любого типа, для которого определено вывода потоком.

По умолчанию операция >> опускает пробельные символы (как определено функцией isspace в ctype.h), а после этого считывает символы, соответствующие типу объекта ввода. Пропускания пробельных символов руководствуется флажком ios :: skipws в перечислений переменной состояния. Флажок skipws обычно устанавливает пропускания пробельных символов. Очистка этого флажка (например, с помощью setf) выключает пропускания пробельных символов. Определим также специальный манипулятор "приемника", ws, что позволяет игнорировать пробельные символы.

Рассмотрим следующий пример:

int i;

double d;

cin >> i >> d;

Последняя строка вызывает пропуска пробельных символов. Цифры, считываются из стандартного устройства ввода (по умолчанию это клавиатура), превращаются после этого

на внутренний двоичный формат и записываются в переменную i. После этого снова пропускаются пробельные символы и наконец считывается число с плавающей точкой, превращается и записывается в переменную d.

Для типа char (signed или unsigned) действие операции >> состоит в пропускании пробельных символов и записи следующего (непробильного) символа. Если вам нужно прочитать следующий символ, несмотря на то, является ли он пробельным или нет, то можно использовать одну из функций элементов get.

Для типа char \* (что рассматривается как строка) действие операции >> состоит в пропускании пробельных символов и записи последующих (непробильних) символов, пока не встретится следующий пробельный символ. После этого добавляется завершающий нулевой (0) символ. Надо быть осторожным и избегать "переполнения" строки. Ширина по умолчанию, равна нулю (означает, что пороговое значение не задано), может быть изменена с помощью setw следующим образом:

char array [Size];

cin.width (sizeof (array))

cin >> array; // Позволяет избежать переполнения

В случае любого введения встроенных типов, если конец ввода встретится раньше первого непробильного символа, в приемник buf ничего записано не будет, а состояние istream будет установлено равным "отказе". Таким образом, если приемник не являлся инициализирован, то он и останется неинициализированных.

**Введение типов, определенных пользователем**

Для ввода или вывода своих собственных, определенных вами типов, вы должны переопределить операции извлечения и вставки. Это можно сделать следующим образом:

#include <iostream. h>

struct info

{

char \* name;

double val;

char \* units;

 };

// Вы можете переопределить << для вывода следующим образом: ostream & operator << (ostream & s, info & m)

 {

    s << m.name << "" << m.val << "" << m.units;

return s;

};

// Вы можете переопределить >> для ввода следующим образом: istream & operator >> (istream & s, info & m)

 {

    s >> m.name >> m.val >> m.units;

    return s;

 };

main ()

{

  x.name = new char [15];

  x.units = new char [10];

  cout << "\ n input name, value and units:";

  cin >> x;

  cout << "\ n my input:" << x;

  return 0;

}

Контрольные вопросы к Л.Р. 6:

1. Что можно назвать шаблоном?

2. Как объявляют шаблоны функций?

3. Что представляет собой шаблон класса?

4. Какой класс называют контейнерным классом?

5. Как можно решить пpоблему надежности по типам?

Задание:

1. Создать структуру родового связанного списка, используя шаблоны.

2. Создать класс WiseString, объекты которого могли бы присвоить себе значение и по запросу выводили бы информацию о себе на монитор.

3. Создать список в который включаются и из которого изымаются сами объекты, а не указатели на них. Пусть он работает для класса Х, в котором определены x :: x (x &), x :: ~ x () и x :: operator = (x &).

4. Прочитайте файл, который состоит из чисел с плавающей точкой, заключите с пар прочитанных чисел комплексные числа и запишите их в файл, используя потоки.

5. Напишите программу, которая принимает с клавиатуры произвольное количество строк и записывает их в файл. Имя файла задается при запуске программы первым параметром командной строки. Работу программы закончить при получении с клавиатуры условия EOF.

6.Напишить программу, которая печатает: (1) - все строчные буквы (2) - все буквы и цифры, (3) - все буквы, (4) - все символы пунктуации (5) - все символы пропуска.

7.Визначиты тип name\_and\_adress. Перегрузить для него операторы << и >>. Скопировать поток объектов name\_and\_adress.

Задания для самостоятельной работы

1. Создать шаблон для создания функций, которые выводят массив типа int и типа float.

2. Создать шаблон в одном массиве с n элементов и вычислить: количество элементов == 0, сумму элементов массива, расположенную после минимального элемента. Сортировать элементы массива по возрастанию их модулей.

3. Разработать программу калькулятор с настройкой количества десятичных знаков и отформатирован видом (выравнивание по правому краю).

4. Записать исходный файл, добавить к каждому числа последнее число файла.

5. Записать исходный файл, умножить каждое третье число на удвоенную сумму первого и последнего отрицательного числа.

6. Записать исходный файл, умножить каждое четное число на первое отрицательное число файла.

7. Дан символьный файл f, состоящий из малых латинских букв и знаков препинания. Переписать файл, заменяя маленькие буквы большими. Вывести на экран файл до и после преобразования.