**Лабораторный практикум**

**по Объектно-ориентированному программированию**

Системные требования: Microsoft Visual C++ не ниже 6-й версии.

**Работа № 1.  
Классы. Программирование линейных алгоритмов с использованием функций инициализации set() и вывода результатов print().**

**Цель работы –** изучить основные способы работы с пользовательским типом данных «класс», его объектами, методами и способы доступа к ним.

**Теоретические сведения**

Основное отличие С++ от С состоит в том, что в С++ имеются классы. С точки зрения языка С классы в С++ - это структуры, в которых вместе с данными определяются функции. Это есть инкапсуляция в терминах ООП.

*Класс (class)* - это тип, определяемый пользователем, включающий в себя данные и функции, называемые методами или функциями-членами класса.

*Данные класса* - это то, что класс знает.

*Функции-члены (методы) класса* - это то, что класс делает.

Таким образом, определение типа задаваемого пользователем (class) содержит спецификацию данных, требующихся для представления объекта этого типа, и набор операций (функций) для работы с подобными объектами.

Объявление класса

Приведем пример объявления класса

class my\_Fun{// компоненты-данные double x,y;  // компоненты-функции public:// функция инициализации void set(char \*c,double X){x=X;y=sin(x);    }// функция вывода результатов void print(void){cout << point<<y << endl;}};

Обычно описания классов включают в заголовочные файлы (\*.H), а реализацию функций-членов классов - в файлы \*.CPP.

Для каждого объекта класса устанавливается *область видимости* либо явно – указанием уровня доступа одним из ключевых слов public, private, protected с двоеточием, либо неявно – по умолчанию.

Указание области видимости относится ко всем последующим объектам класса, пока не встретится указание другой области видимости.

Область видимости *public* разрешает доступ к объектам класса из любой части программы, в которой известен этот объект (общедоступный).

Область видимости *private* разрешает доступ к объектам класса только из методов этого класса. Объекты с такой областью видимости называют частными.

Область видимости *protected* определяется для защищенных объектов, она имеет смысл только в иерархической системе классов и разрешает доступ к объектам этой области из методов производных классов.

В теле класса ключевое слово области видимости может использоваться неоднократно. Область видимости для объектов типа «класс» по умолчанию private.

*Способы объявления и инициализации объектов и доступ к методам класса:*

*1. Прямой вызов*

my\_Fun Fun1; //объявление объекта1,но не инициализация  
Fun1.set("Function1 = ",1.0); // инициализация данных  
Fun1.print();                 // прямой вызов   
cout << "Input enter1..." << endl<<endl;

*2. Косвенный вызов*

my\_Fun \*p1 = &Fun1;  // воспользовались объектом 1  
                     // новая инициализация  
p1->set("Function1 = ",1.0); // косвенный вызов     
p1->print();                  // косвенный вызов     
cout << "Input enter1..." << endl<<endl;

*3. Динамическое выделение памяти*

my\_Fun \*p1 = new my\_Fun;  
p1->set("Function1 = ",1.0);  // косвенный вызов     
p1->print();                  // косвенный вызов     
cout << "Input enter1..." << endl<<endl;

// удаляется динамически выделенный объект  
delete p1;

## Задание к работе

Пользовательский класс должен содержать необходимые элементы-данные,

метод установки их начальных значений:

Void set(double X, …);

метод печати:

Void print(void);

метод, решающий поставленную задачу:

Void Run(void);

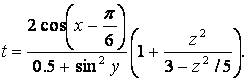
Код методов – вне пространства определения класса.

Программа должна включать в себя статический и динамический способы создания объектов, и для каждого объекта использовать прямую и косвенную адресацию при вызове методов класса.

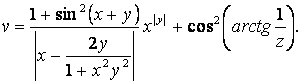
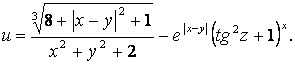
## 

## Варианты заданий

1. При x=14.26, y=-1.22, z=3.5 **t=0.564849.**



1. При x=-4.5, y=0.75,z=0.845 **u=-55.6848.**



При x=3.74, y=-0.825, z=0.16, **v=1.0553.**



1. При x=0.4 ,y=-0.875, z=-0.475**w=1.9873.**



1. При x=-15.246, y=4.642, z=20.001 **=-182.036.**



1. При x=16.55, y=-2.75, z=0.15 **=-40.630.**



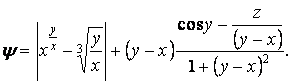
1. При x=0.1722, y=6.33, z=3.25 **=-205.305.**



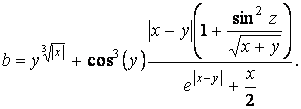
1. При x=-2.235, y=2.23, z=15.221 **=39.374.**



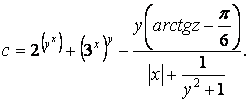
1. При x=1.825, y=18.225, z=-3.298 **=1.2131.**



1. При x=6.251, y=0.827, z=25.001 **b=0.7121.**



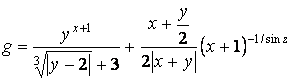
1. При x=3.251, y=0.325, z=0.466**c=4.25.**



1. .  
     
   При x=17.421, y=10.365, z=0.828 **f=0.33056.**



1. .  
     
   При x=12.3, y=15.4, z=0.252**g=82.8257.**



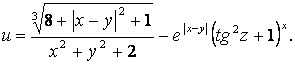
1. .  
     
   При x=2.444, y=0.869, z=-0.13 **-0.49871.**



1. При x=0.1722, y=6.33, z=3.25 **=-205.305.**



1. При x=-4.5, y=0.75,z=0.845 **u=-55.6848.**



1. При x=16.55, y=-2.75, z=0.15 **=-40.630.**



## Контрольные вопросы

1. Что значит в ООП понятие «класс», и какой формат его объявления в программе?
2. Что такое объект класса, что он содержит?
3. Какие существуют  уровни доступа к объектам  и методам класса (дать характеристику каждому)?
4. Что такое операция привязки, ее основное назначение?

**Работа № 2.  
Классы. Программирование линейных алгоритмов с использованием конструктора, деструктора, friend - функции инициализации set() и функции вывода результатов print().**

**Цель работы –** изучить основные способы работы по созданию конструктора класса с захватом динамической памяти и деструктора для ее освобождения, применение friend – функции, и изучение ее особенностей.

**Теоретические сведения**

Конструктор класса

Конструктор – это метод класса, имя которого совпадает с именем класса. Конструктор вызывается автоматически после выделения памяти для пе­ременной и обеспечивает инициализацию компонент – данных. Конструктор не имеет никакого типа (даже типа void) и не возвращает никакого значения в ре­зультате своей работы. Конструктор нельзя вызывать как обычную компо­нентную функцию в программе. Для класса может быть объявлено несколько конструкторов, различающихся числом и типами параметров. При этом даже если для объектного типа не определено ни одного конструктора, компилятор создает для него конст­руктор по умолчанию, не использующий параметров, а также конструктор ко­пирования, необходимый в том случае, если переменная объектного типа передается в конструктор как аргумент. В этом случае создаваемый объект будет точной копией аргумента конструктора.

Деструктор класса

Еще одним специальным методом класса является деструктор. Деструк­тор вызывается перед освобождением памяти, занимаемой объектной пере­менной,  и предназначен для выполнения дополнительных действий, связан­ных с уничтожением объектной переменной, например, для освобождения ди­намической памяти, закрытия, уничтожения файлов и т.п.  Деструктор всегда имеет то же имя, что и имя класса, но перед именем записывается знак ~ (тильда). Деструктор не имеет параметров и подобно кон­структору не возвращает никакого значения. Таким образом, деструктор не может быть перегружен и должен существовать в классе в единственном эк­земпляре. Деструктор вызывается автоматически при уничтожении объекта. Таким образом, для статически определенных объектов деструктор вызывает­ся, когда заканчивается блок программы, в котором определен объект (блок в данном случае – составной оператор или тело функции). Для объектов, память для которых выделена динамически, деструктор вызывается при уничтожении объекта операцией delete.

class my\_Fun{

// компоненты-данные   
double x;  
unsigned size;

public:

// объявление конструктора 1 (с параметрами)  
my\_Fun (double X=0);

// объявление конструктора 2 (без параметров)  
my\_Fun(void);

// объявление и описание деструктора  
~my\_Fun ()  
{  
cout<<"Destroyed object... "<<endl;  
}

// описание конструктора 1my\_Fun::my\_Fun (double X){cout<<"Constructor1...."<<endl;x=X;}// описание конструктора 2my\_Fun::my\_Fun (void){cout<<"Constructor2..."<<endl;x=5.0;}}

Дружественная функция(friend)

В языке С++ одна и та же функция не может быть компонентом двух разных классов. Чтобы предоставить функции возможность выполнения действий над различными классами можно определить обычную функцию языка С++ и предоставить ей право доступа к элементам класса типа private, protected. Для этого нужно в описании класса поместить заголовок функции, перед которым поставить ключевое слово friend. Дружественная функция не является методом класса, не зависит от позиции в классе и спецификаторов прав доступа. Friend – функции получают доступ к членам класса через указатель, передаваемый им явно. Можно сделать все функции класса Y друзьями класса X в одном объявлении.

## Задание к работе

Общая постановка. Пользовательский класс Х должен содержать необходимые элементы-данные, которые создаются в динамической области памяти.

* Конструктор для их создания (операция new) и установки их начальных значений: Х()**;**
* деструктор: ~Х()**;**
* friend – функция печати: friend void print()**;**
* функция, решающая поставленную задачу: friend Void Run()**.**

Код методов и функций – вне пространства определения класса.

## Варианты заданий

Варианты заданий используются из лабораторной работы №1.

## Контрольные вопросы

1. Что такое «конструктор», формат объявления, его особенности?
2. Формат объявления деструктора, его назначение
3. Особенности дружественных функций, доступ к закрытой части класса
4. Каким образом дружественная функция получает доступ к закрытой части класса?

**Работа № 3.  
Класс «Динамическая строка» и перегрузка операций**

**Цель работы –** изучить методику создания одномерных динамических символьных массивов при помощи конструкторов с захватом динамической памяти и деструкторов для их уничтожения, а так же способа работы со строковыми объектами. Познакомиться с механизмом перегрузки операций.

## Теоретические сведения

Для представления символьной (текстовой) информации  можно использовать символы, символьные переменные и символьные константы.

*Символьная константа* представляется последовательностью символов, заключенной в кавычки: “Начало строки \n”. В С++ нет отдельного типа для строк. Массив символов - это и есть строка. Количества элементов в таком массиве на один элемент больше, чем изображение строки, т. к. в конец строки добавлен ‘\0’ (нулевой байт).

Присвоить значение массиву символов с помощью обычного оператора присваивания нельзя. Поместить строку в массив можно либо при вводе, либо с помощью инициализации:

char s[] = “ABCDEF”;

Для работы со строками существует специальная библиотека string.h.

Примеры функций для работы со строками из библиотеки string.h в таблице:

|  |  |
| --- | --- |
| **Функция** | **Прототип и краткое описание функции** |
| strcmp | int strcmp(const char \*str1, const char \*str2); Сравнивает строки str1 и str2. Если str1< str2, то результат отрицательный, если str1 = str2, то результат равен 0, если str1> str2, то результат положительный. |
| strcpy | char\* strcpy(char\*s1, const char \*s2); Копирует байты из строки s1 в строку s2 |
| strdup | char \*strdup (const char \*str); Выделяет память и перености в нее копию строки str. |
| strlen | unsigned strlen (const char \*str); Вычисляет длину строки str. |
| strncat | char \*strncat(char \*s1, const char \*s2, int kol); Приписывает kol символов строки s1 к строке s2. |
| strncpy | char \*strncpy(char \*s1, const char \*s2, int kol); Копирует kol символов строки s1 в строку s2. |
| strnset | char \*strnset(char \*str, int c, int kol); Заменяет первые kol символов строки s1  символом с. |

Строки при передаче в функцию в качестве фактических параметров могут быть определены либо как одномерные массивы типа char[], либо как указатели типа char\*.  В отличие от обычных массивов в этом случае нет необходимости явно указывать длину строки.

*Функции преобразования строки S в число*:

целое:  int atoi(S); длинное целое:  long atol(S); действительное:  double atof(S); при ошибке возвращает значение 0.

*Функции преобразования числа V в строку S:*

целое:  itoa(int V,char S,int kod); длинное целое:  ltoa(long V,char S,int kod); 2<=kod<=36, для отрицательных чисел kod=10.

Перегрузка операций

Для перегрузки операции для класса в С++ используется следующий синтаксис:

<Тип>  operator <операция>(<входные параметры>){   <операторы>;}

где < Тип > - тип, возвращаемый функцией;

operator - ключевое слово;

< операция > - перегружаемая операция.

В языке С++ имеются следующие *ограничения на перегрузку операций*:

* С++ не различает префиксную и постфиксную формы ++ и - -;
* переопределяемая операция должна присутствовать в языке (например, нельзя определить операцию с символом #);
* нельзя переопределить операторы, заданные следующими символами . \* :: ?  ;
* переопределённые операции сохраняют свой изначальный приоритет.

Наличие в классе конструктора String:: String(String&) и операторов присваивания позволяет защитить объекты класса от побитового копирования.

Пример**:** Ввести с клавиатуры строку символов. Признак окончания ввода строки - нажатие клавиши "Ввод". Программа должна определить длину введенной строки L и если L<10 – возвратить строку, которая не содержит  заглавных латинских букв.

#include <iostream.h>  
#define SIZE 255 //длина строки по умолчанию  
#include <string.h>  
#include <stdlib.h>  
#include <stdio.h>  
#include <istream.h>

class X{  
char \*str;   
char \*str\_return;  
public:  
X(); //конструктор по-умолчанию  
X(char\*); //конструктор, которому можно передавать параметр  
~X(); //деструктор  
char\* Run(); //метод, выполняющий поставленную задачу.   
void Set(char\*);  
friend void print(X&); //функция-друг печати  
friend ostream& operator<<(ostream&,X&); //перегрузка оператора вывода  
friend istream& operator>>(istream&,X&); //перегрузка оператора ввода  
friend char\* Run(X&); //функция-друг, выполняющий поставленную задачу.

};

X::X(){  
str=new char[SIZE];   
str[0]='\0';          
str\_return=new char[SIZE];  
str\_return[0]='\0';  
};  
X::X(char \*s){  
str=new char[SIZE];  
strcpy(str,s);  
str\_return=new char[SIZE];  
str\_return[0]='\0';  
};  
X::~X(){  
delete[] str;  
cout<<"...destructor has been called"<<endl;  
};

void X::Set(char\* s){  
for (unsigned int i=0;i<strlen(s);i++)   
str[i]=s[i];  
str[i]='\0';  
};

char\* X::Run(){ /\*метод, решающий конкретную задачу, в данном случае - выделение из строки подстроки, не содержащей заглавных латинских букв, если длина исходной строки меньше 10\*/  
int j=0;   
if (strlen(str)<10) {  
for (unsigned int i=0;i<strlen(str);i++)  
if ( ((int)str[i]<65) || ((int)str[i]>90) ) {  
str\_return[j]=str[i]; j++;  
};  
str\_return[j]='\0';  
}  
else strcpy(str\_return,str);  
  
return str\_return;    
};

char\* Run(X &obj){return obj.Run();};

void print(X &obj){cout<<obj.str<<" "<<obj.str\_return<<endl;};

ostream& operator<<(ostream &stream,X &ob) {  
stream << ob.str ;   
return stream;  
};

istream &operator>>(istream &stream,X &ob){  
stream >> ob.str;  
return stream;  
};  
void main (void){  
char s[265];  
  
cout<<"Type anything and press \"Enter\":"<<endl;  
cin.getline(s,256); //считываем полностью всю строку  
X str(s); //доступ к методам класса непосредственно через переменную,  
//начальное значение устанавливаем через конструктор  
cout<<"You have type:"<<endl;  
print(str);  
cout<<"Output string:"<<endl;  
cout<<Run(str)<<endl;   
cout<<"Type anything and press \"Enter\":"<<endl;  
cin.getline(s,256);  
X \*pstr; //доступ к методам класса через указатель  
pstr=new X();  
pstr->Set(s);  
cout<<"You have type:"<<endl;  
print(\*pstr);  
cout<<"Output string:"<<endl;  
cout<<Run(\*pstr)<<endl;  
delete pstr;

};

**Задание к работе**

Общая постановка. Пользовательский класс String должен содержать необходимые элементы-данные, которые создаются в динамической области памяти.

* Конструктор для создания строк: String (…);
* Деструктор: ~String();
* Метод ввода исходной строки: Set();
* Метод печати: void print(…);

Код методов  – вне пространства определения класса.

Программа иллюстрирует прямой и косвенный способы обращения к методам.

Ввести с клавиатуры строку  символов S1. Признак окончания ввода строки - нажатие клавиши "Ввод". Программа должна содержать перегруженную операцию **«=»**, использование которой скопирует S1 в S2 .

Исходную и преобразованную строки вывести в файл.

**Варианты заданий**

1. Длина L нечетная, то удаляется символ, стоящий посередине строки;
2. Длина L четная, то удаляются 2 первых и 2 последних символа;
3. Длина L кратна 2-м, то удаляются все числа, которые делятся на 2;
4. Длина L кратна 3-м, то удаляются все числа, делящиеся на 3;
5. Длина L >10, то удаляются все цифры;
6. Длина L >15, то удаляются все a..z;
7. Длина L=10, то удаляются все A..Z;
8. Длина L кратна 4-м, то первая часть строки меняется местами со второй;
9. Длина L кратна 5-и, то подсчитывается количество скобок всех видов;
10. Длина L >5-и, то выделяется подстрока до первого пробела;
11. Длина L >6-и, то выделяется подстрока { } скобках;
12. Длина L >10-и, то удаляется подстрока в [] скобках;
13. Длина L >12-и, то удаляется подстрока до первой ( скобки;
14. Длина L кратна 4-м, то выделяется подстрока после последнего пробела;
15. Длина L >5, то удаляются все точки.
16. Длина L четная, то выделяется подстрока до первого пробела
17. Длина L четная, то удаляется подстрока до первого пробела
18. Длина L четная, то выделяется подстрока со второго пробела
19. Длина L нечетная, то выделяется подстрока после первого пробела
20. Длина L нечетная, то удаляется подстрока со второго пробела
21. Длина L кратна 3, то удаляется каждый 3-й символ
22. Длина L четная, то удаляется каждый 2-й символ

**Контрольные вопросы**

1. Как объявить динамическую строку в С++?
2. Какие вы знаете функции работы со строками?
3. Как определяются строки при передаче в функцию, в качестве фактических параметров?
4. Поясните механизм перегрузки операций для объектов данного класса.

**Работа № 4.  
Наследование классов, механизм виртуальных функций**

**Цель работы –** изучить одну из базовых концепций ООП – наследование классов в С++, заключающуюся в построении цепочек классов, связанных иерархически. Познакомиться с механизмом виртуальных функций.

## Теоретические сведения

Наследование - механизм создания производного  класса из базового. Т.е., к существующему классу можно что-либо добавить, или изменять его каким-либо образом для создания нового (производного) класса. Это мощный механизм для повторного использования кода. Наследование позволяет создавать иерархию связанных типов, совместно использующих код и интерфейс.

Модификатор прав доступа используется для изменения доступа к наследуемым объектам в соответствии с правилами, указанными в таблице:

Доступ в классах при наследовании

|  |  |  |
| --- | --- | --- |
| **Доступ**  **в базовом классе** | **Модификатор**  **прав доступа** | **Доступ**  **в производном классе** |
| private | private | не доступны |
| private | public | не доступны |
| protected | private | private |
| protected | public | protected |
| public | private | private |
| public | public | public |

Ограничение на наследование

При определении производного класса не наследуются из базового:

1. конструкторы;
2. деструкторы;
3. операторы new, определенные пользователем;
4. операторы присвоения, определенные пользователем;
5. отношения дружественности.

Использование косвенной адресации с установкой указателей на базовый класс.

*Механизм косвенной адресации* рассмотрим на примере:

**class B  
{  
public:  
int x;  
B() {            // Конструктор по умолчанию  
x = 4; }  
};  
class D : public B {      // Производный класс  
public:  
int y;  
D()   
{            // Конструктор по умолчанию  
y = 5; }  
};  
void main(void)  {  
D  d;// Конструктор класса D создает объект d   
B \*p; // Указатель установлен на базовый касс   
p = &d;// Указатель p инициализируется адресом d   
// косвенное обращение к объектам базового и производного классов  
// «считываем их текущее состояние в переменные   
int i = p -> x;  // Базовый класс виден напрямую  
int j = ( ( D\* ) p )p -> y;// Прямое преобразование указателя на D   
// через переменные печатаем их текущее состояние  
cout << “ x\_i= “ << i << endl;      
cout << “ y\_j= “ << j << endl;      
getch();  
}**

Виртуальная функция и механизм позднего связывания

Виртуальная функция объявляется в базовом или в производном классе и, затем, переопределяется в наследуемых классах.

Совокупность классов (подклассов), в которых определяется и переопределяется виртуальная функция, называется полиморфическим кластером, ассоциированным с некоторой виртуальной функцией. В пределах полиморфического кластера сообщение связывается с конкретной виртуальной функцией-методом во время выполнения программы.

Обычную функцию-метод можно переопределить в наследуемых классах. Однако без атрибута virtual такая функция-метод будет связана с сообщением на этапе компиляции. Атрибут virtualгарантирует позднее связывание в пределах полиморфического кластера.

Часто возникает необходимость передачи сообщений объектам, принадлежащим разным классам в иерархии. В этом случае требуется реализация механизма позднего связывания. Чтобы добиться позднего связывания для объекта, его нужно объявить как указатель или ссылку на объект соответствующего класса. Для открытых производных классов указатели и ссылки на объекты этих классов совместимыми с указателями и ссылками на объекты базового класса (т.е. к объекту производного класса можно обращаться, как будто это объект базового класса). Выбранная функция-метод зависит от класса, на объект которого указывается, но не от типа указателя.

С++ поддерживает virtualфункции-методы, которые объявлены в основном классе и переопределены в порожденном классе. Иерархия классов, определенная общим наследованием, создает связанный набор типов пользователя, на которые можно ссылаться с помощью указателя базового класса. При обращении к виртуальной функции через этот указатель в С++ выбирается соответствующее функциональное определение во время выполнения. Объект, на который указывается, должен содержать в себе информацию о типе, поскольку различия между ними может быть сделано динамически. Это особенность типична для ООП кода. Каждый объект “знает” как на него должны воздействовать. Эта форма полиморфизма называется чистым полиморфизмом.

В С++ функции-методы класса с различным числом и типом параметров есть действительно различные функции, даже если они имеют одно и тоже имя. Виртуальные функции позволяют переопределять в управляемом классе функции, введенные в базовом классе, даже если число и тип аргументов те же самые. Для виртуальных функций нельзя переопределять тип функции. Если две функции с одинаковым именем будут иметь различные аргументы, С++ будет считать их различными и проигнорирует механизм виртуальных функций. Виртуальная функция обязательно метод класса.

## Задание к работе

Общая постановка. Программа должна содержать:

* базовый класс Х, включающий два  элемента х1, х2 типа int,
* конструктор с параметрами для создания объектов в динамической области памяти,
* деструктор,
* виртуальные методы просмотра текущего состояния и переустановки объектов базового класса в новое состояние.
* производный класс У, включающий один элемент у типа int ,
* конструктор с параметрами и  списком инициализаторов, передающий данные конструктору базового класса,
* переопределенные методы просмотра текущего состояния объектов и их переустановки в новое состояние.

## Варианты заданий

Создать в производном классе метод Run, определяющий:

1. Сумму компонент классов
2. Произведение компонент классов
3. Сумму квадратов компонент классов
4. Значение  х1+х2 – у
5. Значение (х1+х2)/у
6. Значение (х1+х2)\*у
7. Значение х1\*у+х2
8. Значение х1+х2\*у
9. Произведение квадратов компонент класса
10. Значение х1\*х2+у
11. Значение х1\*х2/у
12. Значение х1\*х2-у
13. Значение (x1-x2)\*y
14. Значение (x1-x2)/y
15. Разность квадратов компонент классов
16. Значение х1-х2+у
17. Значение х1\*у+х2/y
18. Значение х1/у+х2\*y

## Контрольные вопросы

1. Что такое наследование, одиночное наследование, множественное наследование?
2. Какие объекты базового класса наследуются в производном, а какие нет?
3. На примере своей программы поясните механизм позднего связывания.
4. В каком случае С++ проигнорирует механизм виртуальных функций?

**Работа № 5.  
Программирование шаблона классов**

**Цель работы –** изучить приемы создания и использования шаблонов классов.

## Теоретические сведения

Достаточно часто встречаются классы, объекты которых должны содержать элементы данных произвольного типа (в том смысле, что их тип определяется отдельно для каждого конкретного объекта). В качестве примера можно привести любую структуру данных (массив указателей, массив, список, дерево). Для этого в С++ предлагаются средства, позволяющие определить некоторое множество идентичных классов с параметризованным типом внутренних элементов. Они представляют собой особого вида заготовку класса, в которой в виде параметра задан тип (класс) входящих в него внутренних элементов данных. При создании конкретного объекта необходимо дополнительно указать и конкретный тип внутренних элементов в качестве фактического параметра. Создание объекта сопровождается созданием соответствующего конкретного класса для типа, заданного в виде параметра.  Принятый в С++ способ определения множества классов с параметризованным внутренним типом данных (иначе, макроопределение) называется *шаблоном* (template).

*Синтаксис шаблона* рассмотрим на примере шаблона класса векторов, содержащих динамический массив указателей на переменные заданного типа.

// <class T> - параметр шаблона - класс "T", внутренний тип данных  
// vector -  имя группы шаблонных классов  
template <class T> class vector   
{  
int  tsize;        // Общее количество элементов  
int  csize;        // Текущее количество элементов  
T    \*\*obj;       // Массив указателей на парам. объекты типа "T"                                    
public:  
T \*operator[](int); // оператор [int] возвращает указатель на   
// параметризованный объект  класса "T"  
void insert(T\*);   // включение указателя на объект типа "T"  
int  index(T\*);           
};

Данный шаблон может использоваться для порождения объектов-векторов, каждый из которых хранит объекты определенного типа. Имя класса при этом составляется из имени шаблона "vector**"** и имени типа данных (класса), который подставляется вместо параметра **"**Т**":**

vector<int>   a;  
vector<double>     b;  
extern class  time;  
vector<time>      c;

Заметим, что транслятором при определении каждого вектора с новым типом объектов генерируется описание нового класса по заданному шаблону (естественно, неявно в процессе трансляции). Например, для типа int транслятор получит:

class vector<int>  
{  
int  tsize;  
int  csize;  
int  \*\*obj;  
public:  
int \*operator[](int);  
void insert(int\*);  
int  index(int\*);  
};

Далее следует очевидное утверждение, что функции-методы шаблона также должны быть параметризированы, то есть генерироваться для каждого нового типа данных. Действительно, это так: функции-методы шаблона классов в свою очередь также являются шаблонными функциями с тем же самым параметром. То же самое касается переопределяемых операторов:

// параметр шаблона - класс "T", внутренний тип данных  
// имя функции-элемента или оператора - параметризировано  
//         
template <class T>  T\* vector<T>::operator[](int n)  
{  
if (n >=tsize) return(NULL);  
return (obj[n]);  
}  
template <class T> int vector<T>::index(T \*pobj)  
{  
int  n;  
for (n=0; n<tsize; n++)  
if (pobj == obj[n]) return(n);  
return(-1);  
}

Заметим, что транслятором при определении каждого вектора с новым типом объектов генерируется набор методов-функций по заданным шаблонам (естественно, неявно в процессе трансляции). При этом, сами шаблонные функции должны размещаться в том же заголовочном файле, где размещается определение шаблона самого класса. Для типа int сгенерированные транслятором функции-методы будут выглядеть так:

int\* vector<int>::operator[](int n)  
{  
if (n >=tsize) return(NULL);  
return (obj[n]);  
}  
int vector<int>::index(int \*pobj)  
{  
int  n;  
for (n=0; n<tsize; n++)  
if (pobj == obj[n]) return(n);  
return(-1);  
}

## Задание к работе

Общая постановка. Дано: число N и последовательность a1, a2, … aN   
Создать шаблон класса, порождающий динамические одномерные массивы с элементами различных типов (вещественные, целочисленные, символьные и т.д.). Тип данных и результат являются параметрами по отношению к классу, программа должна иметь методы инициализации, конструктор, деструктор, метод просмотра значений созданного массива, согласно заданному алгоритму.

## Варианты заданий

1. a1, (a1+a2), … ,(a1+a2+…+aN);
2. (a1\*a1), (a1\*a2), …, (a1\*aN);
3. |a1|, |a1+a2|, …, |a1+a2+…aN|;
4. a1, -a1\*a2, +a1\*a2\*a3, … ,(-1)N\*a1\*a2\*…aN;
5. -a1, +a2, -a3, … , (-1)N\*aN;
6. (a1+1), (a2+2) , (a3+3), …, (aN+N);
7. a1\*1, a2\*2, a3\*3,     , aN\*N;
8. a1\*a2, a2\*a3, … , (aN-1)\*aN;
9. a1/1, a2/2, a3/3, …,aN/N;
10. (a1+a2), (a2+a3),… ,(aN-1+aN);
11. (a1+a2+a3), (a2+a3+a4), (a3+a4+a5), … (aN-2+aN-1+aN);
12. (N+a1), ( N-1+a2),   ,(1+aN);
13. (N\*a1), ( (N-1)\*a2),   ,(1\*aN);
14. a1/N, a2/N-1, … ,aN/1.
15. (a1+ a1/N), (a2+ a2/N-1) , (a3+ a3/N-2), …, (aN+ aN /1);
16. (a1+1/N), (a2+2/N) , (a3+3/N), …, (aN+1);
17. a1/(1+N), a2/(2+N), a3/(3+N), …,aN/(N+N);

## Контрольные вопросы

1. С какой целью используются шаблоны классов?
2. Какие существуют виды параметров шаблона класса?
3. В чем заключается особенность использования функций – методов шаблона?
4. Может ли использоваться шаблон для параметризированных объектов?

**Работа № 6.  
Множественное наследование с использованием абстрактных  базовых классов, файлового ввода-вывода с применением потоков С++,  функций обработки исключительных ситуаций**

**Цель работы –** изучить методику создания множественного наследования, использование абстрактного базового класса, файловый ввод – вывод и использование функций обработки исключительных ситуаций.

## Теоретические сведения

Абстрактные классы

Если базовый класс используется только для порождения производных классов, то виртуальные функции в базовом классе могут быть "пустыми", поскольку никогда не будут вызваны для объекта базового класса. Базовый класс, в котором есть хотя бы одна такая функция, называется абстрактным.

Виртуальные функции в определении класса обозначаются следующим образом:

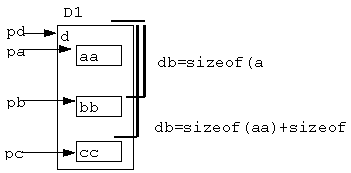
class base  
{  
public:  
virtual print()=0;  
virtual get() =0;  
};

Определять тела этих функций не требуется.

Множественное наследование

Множественным наследованием называется процесс создания производного класса из двух и более базовых. В этом случае производный класс наследует данные и функции всех своих базовых предшественников. Существенным для реализации множественного наследования является то, что адреса объектов второго и последующих базовых классов не совпадают с адресом объекта производного класса. Этот факт должен учитываться транслятором при преобразовании указателя на производный класс в указатель на базовый и наоборот:

class d : public a,public b, public c { };  
d       D1;  
pd =    &D1;        // #define db sizeof(a)  
pa =    pd;         // #define dc sizeof(a)+sizeof(b)  
pb =    pd;         // pb = (char\*)pd + db  
pc =    pd;         // pc = (char\*)pd + dc



Такое действие выполняется компилятором как явно при преобразовании в программе типов указателей, так и неявно, когда в объекте производного класса наследуется функция из второго и последующих базовых классов. Для вышеуказанного примера при определении в классе bb функции f() и ее наследовании в классе "d" вызов D1.f() будет реализован следующим образом:

this = &D1;                 // Указатель на объект производного класса  
this = (char\*)this + db     // Смещение к объекту базового класса  
b::f(this);                 // Вызов функции в базовом классе

Механизм виртуальных функций при множественном наследовании имеет свои особенности.

Во-первых, на каждый базовый класс в производном классе создается свой массив виртуальных функций (в нашем случае -для aa в d, для bb в d и для cc в d).

Во-вторых, если функция базового класса переопределена в производном, то при ее вызове требуется преобразовать указатель на объект базового класса в указатель на объект производного. Для этого транслятор включает соответствующий код, корректирующий значение this в виде "заплаты", передающей управление командой перехода к переопределяемой функции, либо создает отдельные таблицы смещений.

Файловые потоки.

Классы файловых потоков:

ifstream   - файл ввода, производный от istream

ofstream   - файл вывода, производный от ostream

fstream     - файл ввода-вывода, производный от iostream

Флаги режимов работы с файлом:

enum ios::open\_mode  
{  
in = 0x01,         // Открыть файл только для чтения  
out =    0x02,         // Открыть файл только для записи  
ate =    0x04,         // При открытии позиционироваться в конец файла   
app =    0x08,         // Открыть существующий для дополнения  
trunc =  0x10,    // Создание нового файла взамен существующего   
nocreate=0x20,     // Не создавать новый файл при его отсутствии  
noreplace=0x40,    // Не создавать новый файл, если он существует  
binary=  0x80 // Двоичный файл ("прозрачный" ввод-вывод без  
// преобразования символов конца строки)  
};

Конструкторы объектов (для классов ifstream,ofstream,fstream) и функции открытия/закрытия файлов:

ifstream();        // Без открытия файлов  
ifstream(          // С открытием файла в заданном   
char \*name,   // режиме imode   
int imode=ios::in,   
int prot=filebuf::openprot);

ifstream(int fd);  // С присоединенем файла с дескрип-  
// тором fd

ifstream(     // То же, с явно заданным буфером  
int fd,   
char \*buf, int sz);

void ifstream::open(      
char \*name,   // Открытие файла в заданном режиме   
int imode=ios::in,   
int prot=filebuf::openprot);

void close(); // Закрыть файл   
void setbuf(  
char \*p,int sz);// Установить буфер потока   
int  fd();         // Дескриптор открытого в потоке файла  
int  is\_rtl\_open();     // 1 - файл открыт в потоке

Унаследованные переопределения операторов позволяют проверять наличие ошибок в потоках в виде:

fstream  ss;

if (ss) ...   или  if (!ss) ...

Обработка исключительных ситуаций

Средства обработки ошибочных ситуаций позволяют передать обработку исключений из кода, в котором возникло исключение,   некоторому другому программному блоку, который выполнит в данном случае некоторые определенные действия. Таким образом, основная идея данного механизма состоит в том, что функция проекта, которая обнаружила непредвиденную ошибочную ситуацию, которую она не знает, как решить, генерирует сообщение об этом (бросок исключения). А система вызывает по этому сообщению программный модуль, который перехватит исключение  и отреагирует на возникшее нештатное событие. Такой программный модуль называют *«обработчик»* или *перехватчик исключительных ситуаций*. И в случае возникновения исключения  в его обработчик передаётся произвольное количество информации с  контролем ее типа. Эта информация и является характеристикой возникшей нештатной ситуации.

Обработка исключений в С++ это обработка с завершением. Это означает, что   исключается невозможность возобновления выполнения программы в точке возникновения исключения.

Для обеспечения работы такого механизма были введены следующие *ключевые слова*:

**try** - проба испытания;

**catch**  - перехватить (обработать);

**throw** - бросать.

Кратко рассмотрим их назначение.

**try** - открывает блок кода, в котором может произойти ошибка; это обычный составной оператор:

try { код};

Код содержит набор операций и операторов, который и будет контролироваться на возникновение ошибки. В него могут входить  вызовы  функции пользователя, которые компилятор также возьмет на контроль. Среди данного набора операторов и операций обязательно указывают операцию броска исключения:  **throw**.

Операция броска **throw** имеет следующий формат:

throw  выражение;

где - «выражение» определяет тип информации, которая и описывает исключение (например, конкретные типы данных).

**catch** - сам обработчик исключения, который перехватывает информацию:

catch (тип параметр){код  }

Через параметр обработчику передаются данные определенного типа, описывающие обрабатываемое исключение.

Код определяет те действия, которые надо выполнить при возникновении данной конкретной ситуации. В С++ используют несколько форм обработчиков. Такой обработчик получил название *параметризованный специализированный перехватчик.*

Перехватчик должен следовать сразу же после блока контроля, т.е. между обработчиком и блоком контроля не должно быть ни одного оператора. При этом в одном блоке контроля можно вызывать исключения разных типов для разных ситуаций, поэтому обработчиков может быть несколько. В этом случае их необходимо расположить сразу же за контролирующим блоком последовательно друг за другом.

Кроме того, запрещены переходы, как извне в обработчик, так и между обработчиками.

Можно воспользоваться *универсальным или абсолютным обработчиком*:

catch ( . . . ) {код }

где (...) - означают способность данного перехватчика обрабатывать информацию любого типа. Такой обработчик располагают последним в пакете специализированных обработчиков. Тогда, если исключение не будет перехвачено специализированными обработчиками, то буде выполнен последний - универсальный.

В случае не возникновения исключения,  набор обработчиков будет обойден, т.е. проигнорирован.

Если же исключение было брошено, при возникновении критической ситуации, то будет вызван конкретный перехватчик при совпадении его параметра с выражением в операторе броска, т.е. управление будет передано найденному обработчику. После выполнения кода вызванного обработчика, управление передается  оператору, который расположенный за последним перехватчиком, или проект корректно завершает работу.

*Существенное отличие вызова конкретного обработчика от вызова обычной функции* заключается в следующем: при возникновении исключения и передаче управления определенному обработчику, система осуществляет вызов всех деструкторов для всех объектов классов, которые были созданы с момента начала контроля  и до возникновения исключительной ситуации с целью их уничтожения.

Блоки **try**, как составные блоки могут быть вложены:

try { ...try {...  }...}

тогда, в случае возникновения исключения в некотором текущем блоке, поиск обработчика последовательно продолжается в блоках, предшествующих уровней вложенности с продолжением вызова деструкторов.

## Задание к работе

Общая постановка. Создать программу с абстрактным базовым классом и множественным наследованием, реализовать в нем:

* конструктор,
* деструктор,
* виртуальную функцию просмотра текущего состояния объекта print(),
* friend,
* функцию Run ().

Производные классы должны содержать переопределенную функцию просмотра состояния объектов (print()). Используя стандартные файловые потоки, информацию об объектах вывести в файл. При вводе - выводе данных обработка ошибочных ситуаций (например, невозможность открыть файл или нехватка места на диске) должна обрабатываться с использованием механизма исключений.

## Варианты заданий

1. Книги       
   (Название, автор, жанр, год, страниц, тираж, отпечатано листов - функция Run ())
2. Транспорт         
   (Наименование, тип, год выпуска, макс. скорость, объем двигателя, расход, объем бензобака, расстояние без подзаправок - функция Run ()),
3. Продовольственные товары        
   (Наименование, отдел магазина, дата выпуска, срок хранения, последний срок - функция Run (), вес)
4. Студенты          
   (ФИО, год поступления, курс, дисциплины, оценки, средний балл - функция Run ())
5. Объекты недвижимости     
   (Адрес, тип, этажность, квартир на этаж, подъездов, всего квартир - функция Run ())
6. Спортсмены     
   (ФИО, вид спорта, разряд, дата рождения, медалей (каждого типа), возрастная категория - функция Run ())
7. Периодические издания    
   (Название, тип, страниц, частота выпуска, тираж, выпусков в год - функция Run ())
8. Отдел кадров    
   (ФИО, отдел, должность, дата приема на работу, внутренний стаж - функция Run (), ставка)
9. Научно-исследовательские разработки           
   (Наименование, дата начала, дата завершения, срок работы - функция Run (), область исследования, количество сотрудников, ФИО сотрудников)
10. Программное обеспечение   
    (Наименование, тип, количество дисков, объем после установки (полной, минимальной, типичной версий), процент сжатия - функция Run ())
11. Комплектующие ЭВМ   
    (Наименование, тип, модель, частота, объем памяти, стоимость, количество, итоговая стоимость - функция Run ())
12. Перевозки        
    (Пункт назначения, расстояние, количество транспорта, государственные номера машин[], наименование товара [], дата/время выезда, дата/время прибытия, время в дороге - функция Run (), средняя скорость )
13. Аудио-студия           
    (Группа/исполнитель, количество человек, стиль, количество альбомов, стоимость записи диска [], стоимость диска [], тираж[], общая прибыль группы - функция Run (), доход исполнителя - функция Run1 ())
14. Мобильные телефоны        
    (Наименование, фирма, стандарт связи, заряд аккумулятора, потребление при ожидании, потребление при разговоре, время ожидания - функция Run (), время разговора - функция Run1 ())
15. Сетевое оборудование   
    (Наименование, скорость передачи данных, тип, стоимость, количество, общая стоимость - функция Run (), максимальная скорость передачи (байт/сек))
16. Парковка

(Номер машиноместа, стоимость 1-го часа, стоимость последующих часов, марка машины, дата/время въезда, дата/время выезда, итоговая стоимость - функция Run ())

1. Пошив одежды

(Наименование, цвет, размерный ряд, расход ткани[], размер партии[], суммарные расходы на партию - функция Run ())

**Контрольные вопросы**

* 1. Что такое множественное наследование?
  2. Как объявляются виртуальные функции в абстрактном базовом классе?
  3. Поясните механизм виртуальных функций при множественном наследовании.
  4. Какие вы знаете функции обработки исключительных ситуаций (пояснить особенности каждой)?