Д. А. Ганьшин, С. В. Снисаренко

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ПРОЕКТИРОВАНИЕ СИСТЕМ УПРАВЛЕНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО вузов Республики Беларусь по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов учреждений, обеспечивающих получение высшего образования по специальности «Информационные технологии и управление в технических системах»

Под общей редакцией А. Г. Корбита

Минск 2008

УДК (075.8)
ББК 32.973.202-018.2 я7
Г 19

Р е ц е н з е н т ы :
доцент кафедры №206 УО «Военная академия Республики Беларусь»,
канд. техн. наук В. А. Кондратёнок

заведующий кафедрой информатики УО «Минский государственный высший радиотехнический колледж,
канд. техн. наук Ю. А Скудняков

Ганьшин, Д. А.

Г 19 Информационные технологии и проектирование систем управления. Лабораторный практикум : учебно-метод. пособие / Д. А. Ганьшин, С. В. Снисаренко; под общ. ред. А. Г. Корбита. – Минск : БГУИР, 2008. – 40 с.

ISBN 978-985-488-189-8

Целью настоящего издания является оказание помощи студентам специальности «Информационные технологии и управление в технических системах» при выполнении лабораторных работ по курсу «Информационные технологии и проектирование систем управления».

В практикуме рассматриваются вопросы, относящиеся к использованию технологии объектно-ориентированного программирования на языке С++ при проектировании систем управления. Описание методологии построения и использования основных принципов объектно-ориентированного программирования сопровождается примерами.

УДК (075.8)
ББК 32.973.202-018.2 я7

ISBN 978-985-488-189-8

СОДЕРЖАНИЕ

Введение

C++ является языком объектно-ориентированного программирования (ООП). Объект – абстрактная сущность, наделенная характеристиками объектов реального мира. Как и любой другой язык ООП, C++ использует три основные идеи ООП – инкапсуляцию, наследование и полиморфизм.

Инкапсуляция – сведение кода и данных воедино в одном объекте, получившим название класс.

Наследование – наличие в языке ООП механизма, позволяющего объектам класса наследовать характеристики более простых и общих типов. Наследование обеспечивает как требуемый уровень общности, так и необходимую специализацию.

Полиморфизм – дословный перевод с греческого «много форм». В C++ полиморфизм реализуется с помощью виртуальных функций, которые позволяют в рамках всей иерархии классов иметь несколько версий одной и той же функции. Решение о том, какая именно версия должна выполняться в данный момент, определяется на этапе выполнения программы и носит название позднего связывания.

Существует несколько реализаций системы, поддерживающих стандарт C++, из которых можно выделить реализации Visual C++ (Microsoft) и Builder C++ (Inprise). Отличия относятся в основном к используемым библиотекам классов и средам разработки. В действительности в C++ программах можно использовать библиотеки языка C, библиотеки классов C++, библиотеки визуальных классов VCL (Builder C++), библиотеку MFC (Visual C++ и Builder C++).

Язык C++ является родоначальником множества объектно-ориентированных языков, таких, как Java, C#, PHP и др.

Данное издание предназначено для начинающих изучение технологии ООП для проектирования систем управления на основе C++.

Лабораторная работа №1
Программирование алгоритмов с использованием динамических массивов

Цель работы – научиться использовать операции динамического выделения и освобождения памяти на примере работы с одномерными и двухмерными массивами, а также косвенное обращение к элементам массива.

- ***Теоретические сведения***

Объявление динамического массива

Массивы, создаваемые в динамической памяти, будем называть *динамическими* (размерность становится известна в процессе выполнения программы). При описании массива после имени в квадратных скобках задается количество его элементов (размерность), например, `int a[10]`. Размерность массива может быть задана только константой или константным выражением.

При описании массив можно инициализировать, т.е. присвоить его элементам начальные значения, например:

```
int a[10] = {1, 1, 2, 2, 5, 100};
```

Если инициализирующих значений меньше, чем элементов в массиве, остаток массива обнуляется, если больше – лишние значения не используются. Элементы массивов нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности. Номер элемента указывается после его имени в квадратных скобках: например, `a[0], a[3]`.

Если до начала работы программы неизвестно, сколько в массиве элементов, то в программе следует использовать динамические массивы. Память под них выделяется с помощью операции `new` или функции `malloc` в динамической области памяти во время выполнения программы. Адрес начала массива хранится в переменной, называемой указателем. Например:

```
int n = 10;
int *mass1 = new int[n];
```

Во второй строке описан указатель на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью операции `new`. Выделяется столько памяти, сколько необходимо для хранения `n` величин типа `int`. Величина `n` может быть переменной. Инициализировать динамический массив нельзя.

Обращение к элементу динамического массива осуществляется так же, как и к элементу обычного. Если динамический массив в какой-то момент работы программы перестает быть нужным и мы собираемся впоследствии использовать эту память повторно, то необходимо освободить ее с помощью операции `delete[]`, например: `delete [] a`; (размерность массива при этом не указывается).

```
delete[] mass1;
```

При необходимости создания многомерных динамических массивов сначала следует с помощью операции new выделить память под n указателей (вектор, элемент которого указатель), при этом все указатели располагаются в памяти последовательно друг за другом. После этого необходимо в цикле каждому указателю присвоить адрес выделенной области памяти размером, равным второй границе массива:

```
mass2=new int*[row];      // mass2 - указатель на
                                  // массив
                                  //указателей на
                                // одномерные массивы
    for(i=0;i<row;i++)
    mass2[i]=new int[col];// каждый элемент массива
                          // указывает на
                          //одномерный
    for (i=0; i<row;i++)
    for (j=0;j<col;j++)
```

Освобождение памяти от двухмерного динамического массива:

```
for(i=0;i<row;i++)        //удаление всех одномерных
        delete[] mass2[i]; // массивов
delete[] mass2;            // удаление массива
                     //указателей
                     // на одномерные массивы
```

- ***Задание к лабораторной работе***

Общая постановка. Составить программы – одномерные массивы : задания 1 – 25, двухмерные массивы: задания 26 – 50. Массивы создаются в динамической области памяти с использованием операций NEW и DELETE. Ввод исходных данных: реальный размер массивов и их значения. Обращение к элементам массива – через косвенную адресацию.

- ***Варианты заданий***

1. Заданы два массива – A(5) и B(4). Первым на печать вывести массив, сумма значений которого окажется наименьшей.

2. Заданы два массива – A(5) и B(4). Первым на печать вывести массив, произведение значений которого окажется наименьшим.

3. Заданы два массива – A(5) и B(5). В каждом из массивов найти наименьшее значение и прибавить его ко всем элементам массивов. На печать вывести исходные и преобразованные массивы.

4. Заданы два массива – A(5) и B(5). В каждом из массивов найти наибольшее значение и вычесть его из всех элементов массивов. На печать вывести исходные и преобразованные массивы.

5. Заданы два массива – A(5) и B(5). В каждом из массивов найти среднее арифметическое всех элементов массивов. На печать вывести исходные массивы и найденные значения.

6. Заданы два массива – A(5) и B(4). Первым на печать вывести массив, содержащий наибольшее значение. Напечатать также это значение и его порядковый номер.

7. Заданы два массива – A(5) и B(5). Подсчитать в них количество отрицательных элементов и первым на печать вывести массив, имеющий наименьшее их количество.

8. Заданы два массива – A(5) и B(5). Подсчитать в них количество положительных элементов и первым на печать вывести массив, имеющий наименьшее их количество.

9. Заданы два массива – A(5) и B(5). Подсчитать в них количество отрицательных элементов и первым на печать вывести массив, имеющий наибольшее их количество.

10. Заданы два массива – A(5) и B(5). Подсчитать в них количество положительных элементов и первым на печать вывести массив, имеющий наибольшее их количество.

11. Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, больших значения t, и первым на печать вывести массив, имеющий наименьшее их количество.

12. Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, меньших значения t, и первым на печать вывести массив, имеющий наименьшее их количество.

13. Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, больших значения t, и первым на печать вывести массив, имеющий наибольшее их количество.

14. Заданы два массива – A(5) и B(5). В каждом из массивов найти наименьшее значение и умножить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.

15. Заданы два массива – A(5) и B(5). В каждом из массивов найти наибольшее значение и умножить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.

16. Заданы два массива – A(5) и B(5). В каждом из массивов найти наименьшее значение и разделить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.

17. Заданы два массива – A(5) и B(5). В каждом из массивов найти наибольшее значение и разделить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.

18. Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, кратных двум, и первым на печать вывести массив, имеющий наибольшее их количество.

19. Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, кратных трем, и первым на печать вывести массив, имеющий наибольшее их количество.

20. Заданы два массива – A(5) и B(5). Подсчитать в них количество элементов, меньших значения t, и первым на печать вывести массив, имеющий наибольшее их количество.

21. Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые больше 0.

22. Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые меньше 0.

23. Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые кратны двум.

24. Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые больше значения T.

25. Задан массив – A(10). Получить из него массив B, состоящий из элементов массива A, которые кратны трем.

26. Дан массив – A(n,n). Найти число элементов массива a(i,j) > t и просуммировать все эти элементы.

27. Дан одномерный массив – A(n). Сформировать массив B(k), состоящий из a(i) > t. На печать вывести исходный массив, сформированный массив и его размерность.

28. Дан массив – A(n,n). Вычислить сумму всех неотрицательных элементов, а также их количество.

29. Дан массив – A(n,n). Вычислить сумму всех отрицательных его элементов и их количество.

30. Дан массив – A(n,n). Сформировать вектор B(k) из a(i,j) < 0. На печать вывести исходный массив, полученный вектор и его размерность.

31. Дан массив – A(n,n). Написать программу его поворота на 90° относительно его центра. На печать вывести исходный и повернутый массивы.

32. Дан массив – A(n,n). Написать программу его поворота на 180° относительно его центра. На печать вывести исходный и повернутый массивы.

33. Дан массив – A(n,n). Написать программу его поворота на 270° относительно его центра. На печать вывести исходный и повернутый массивы.

34. Дан массив – A(n,n). Найти сумму всех его элементов, расположенных выше главной диагонали.

35. Дан массив – A(n,n). Найти сумму всех его элементов, расположенных ниже главной диагонали.

36. Дан массив – A(n,n). Найти сумму всех его элементов, расположенных выше диагонали, противоположной главной.

37. Дан массив – A(n,n). Найти сумму всех его элементов, расположенных ниже диагонали, противоположной главной.

38. Задана матрица – A(n,n). Найти суммы и произведения элементов, стоящих на главной и противоположной (побочной) диагоналях.

39. Задана матрица – A(n,n), состоящая из нулей и единиц. Подсчитать количество нулей и единиц в этой матрице.

40. Задана матрица – A(n,n). Переставить местами k-ю и i-ю строки, а затем l-й и j-й столбцы.

41. Задан массив действительных чисел A(n). Необходимо каждый элемент массива разделить на среднее арифметическое этих элементов. На печать вывести исходный и преобразованный массивы.

42. Задан массив – A(n). Получить массив B(k), состоящий из элементов массива A, которые делятся на 3. Подсчитать количество элементов массива B.

43. Задана матрица A(n,n). Получить матрицу B=A^2. Элемент b[I][j] определяется как сумма от поэлементного произведения i-й строки на j-й столбец матрицы A.

44. Вычислить первую норму матрицы A(n,n), определяемую как $\|A\| = \max_i \sum |a[i][j]|$, т.е. максимальная сумма из сумм элементов по столбцам.

45. Вычислить вторую норму матрицы A(n,n), определяемую как максимальная сумма из сумм элементов по строкам $\|A\| = \max_j \sum |a[i][j]|$.

46. Задан двухмерный массив целых чисел A размером N на M. Найти сумму элементов, расположенных на главной диагонали.

47. Задан двухмерный массив целых чисел A размером N на M. Найти произведение элементов, расположенных на главной диагонали.

48. Задан двухмерный массив целых чисел A размером N на M. Найти максимальный элемент и поменять его с элементом A[1,1].

49. Задан двухмерный массив целых чисел A размером N на M. Найти минимальный элемент и поменять его с элементом A[1,1].

50. Задан двухмерный массив целых чисел A размером N на M. Найти максимальный элемент и поменять его с последним.

- ***Контрольные вопросы***

1. В чем заключается особенность динамических массивов?
2. Какие вы знаете операции динамического выделения и освобождения памяти в C++?
3. Как объявить динамический многомерный массив, используя указатели?
4. Что содержит указатель на массив?

Лабораторная работа №2
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 C++ от C состоит в том, что в C++ имеются классы. С точки зрения языка C классы в C++ – это структуры, в которых вместе с данными определяются функции. Это и 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 лабораторной 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. $$t = \frac{2\cos\left(x - \frac{p}{6}\right)}{0{,}5 + \sin^2 y}\left(1 + \frac{z^2}{3 - z^2/5}\right).$$

При x=14,26, y= –1,22, z=3,5×10^{-2}, t=0,564849**.**

2. $$u = \frac{\sqrt[3]{8 + |x - y|^2 + 1}}{x^2 + y^2 + 2} - e^{|x-y|}\left(tg^2 z + 1\right)^x.$$

При x= –4,5, y=0,75×10^{-4}, z=0,845×10^2, u=-55,6848.

3. $$v = \frac{1 + \sin^2(x + y)}{\left|x - \frac{2y}{1 + x^2 y^2}\right|} x^{|y|} + \cos^2\left(arctg\frac{1}{z}\right)$$

При x=3,74×10^{-2}, y= –0,825, z=0,16×10^2, v=1,0553**.**

4. $$w = \left|\cos x - \cos y\right|^{\left(1 + 2\sin^2 y\right)}\left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right)$$

При x=0,4×10^4, y= –0.875, z= –0,475×10^{-3}, w=1,9873.

5. $a = \ln\left(y^{-\sqrt{|x|}}\right)\left(x - \frac{y}{2}\right) + \sin^2 arctg(z).$

При x= –15,246, y=4,642×10^{-2}, z=20.001×10^{2}, a = –182.036.

6. $b = \sqrt{10\left(\sqrt[3]{x} + x^{y+2}\right)}\left(\arcsin^2 z - |x - y|\right).$

При x=16,55×10^{-3}, y= –2,75, z=0,15, b = – 40,630.

7. $g = 5arctg(x) - \frac{1}{4}\arccos(x)\frac{x + 3|x - y| + x^2}{|x - y|z + x^2}.$

При x=0,1722, y=6,33, z=3,25×10^{-4}, g = –205,305.

8. $j = \frac{e^{|x-y|}|x - y|^{x+y}}{arctg(x) + arctg(z)} + \sqrt[3]{x^6 + \ln^2 y}.$

При x= –2,235×10^{-2}, y=2,23, z=15,221, j =39,374.

9. $y = \left|x^{\frac{y}{x}} - \sqrt[3]{\frac{y}{x}}\right| + (y - x)\frac{\cos y - \frac{z}{(y - x)}}{1 + (y - x)^2}.$

При x=1,825×10^{2}, y=18,225, z= –3,298×10^{-2}, y =1,2131.

10. $b = y^{\sqrt[3]{|x|}} + \cos^3(y)\frac{|x - y|\left(1 + \frac{\sin^2 z}{\sqrt{x + y}}\right)}{e^{|x-y|} + \frac{x}{2}}.$

При x=6,251, y=0,827, z=25,001, b=0,7121.

11. $c = 2^{\left(y^x\right)} + \left(3^x\right)^y - \dfrac{y\left(arctgz - \dfrac{p}{6}\right)}{|x| + \dfrac{1}{y^2+1}}$.

При x=3,251, y=0,325, z=0,466×10^{-4}, c = 4,25.

12. $f = \dfrac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y|\left(\sin^2 z + tgz\right)}$.

При x=17,421, y=10,365×10^{-3}, z=0,828×10^{5}, f=0,33056.

13. $g = \dfrac{y^{x+1}}{\sqrt[3]{|y-2|+3}} + \dfrac{x + \dfrac{y}{2}}{2|x+y|}(x+1)^{-1/\sin z}$.

При x=12,3×10^{-1}, y=15,4, z=0,252×10^{3}, g=82,8257.

14. $h = \dfrac{x^{y+1} + e^{y-1}}{1 + x|y - tgz|}\left(1 + |y-x|\right) + \dfrac{|y-x|^2}{2} - \dfrac{|y-x|^3}{3}$.

При x=2,444, y=0,869×10^{-2}, z= –0,13×10^{3}, h = –0,49871.

- ***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?

Лабораторная работа №3
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 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.

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

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`.

Дружественная функция (`friend`)

В языке C++ 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 C++ 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 лабораторной работе***

Общая постановка. Пользовательский класс `X` должен содержать необходимые элементы - данные, которые создаются в динамической области памяти, конструктор для их создания (операция `new`) и установки их начальных значений: `X()`, деструктор: `~ X ()`, `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 № 2.

- ***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?

Лабораторная работа №4
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 также 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 C++ нет отдельного типа для строк. Массив символов – это и есть строка. Количество 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` приведены в табл. 4.1.

Таблица 4.1

Функции работы со строками

Функция	Прототип и краткое описание функции
1	2
`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 переносит 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`.

Окончание табл. 4.1

1	2
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 c.

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 C++ используется следующий синтаксис:

<Тип> operator <операция>(<входные параметры>)
{
  <операторы>;
}
где < Тип > – тип, возвращаемый функцией;
operator – ключевое слово;
< операция > – перегружаемая операция.

В языке C++ имеются следующие ограничения на перегрузку операций:

- C++ 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: . * :: ? ;

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

19. Если длина L нечетная, то выделяется подстрока после первого пробела.

20. Если длина L нечетная, то удаляется подстрока со второго пробела.

21. Если длина L кратна 3, то удаляется каждый 3-й символ.

22. Если длина L четная, то удаляется каждый 2-й символ.

23. Если длина L нечетная, то удалить средние 3 символа.

24. Если длина L четная, то выделяется подстрока до последнего пробела.

25. Если длина L нечетная, то выделяется подстрока от последней цифры.

26. Если длина L=15, то удаляются все символы, кроме A – Z.

27. Если длина L делится на 5, то удаляются все символы кроме a – z.

28. Если длина L четная и ≥ 10, то удаляются все пробелы.

29. Если длина L нечетная и <12, произвести инверсию (abcdef в fedcba).

30. Если длина L >5 и <30, изменить регистр символов (aBcDeF в AbCdEf).

- ***Контрольные вопросы***

1. Как объявить динамическую строку в C++?
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.

Лабораторная работа №5
Наследование классов, механизм виртуальных функций

Цель работы – изучить одну из базовых концепций ООП, наследование классов в C++, 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 табл. 5.1.

Таблица 5.1

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 класс
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 `virtual` гарантирует 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 совместимы 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.

C++ поддерживает `virtual` функции-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 C++ выбирается соответствующее функциональное определение во время выполнения. Объект, на который указывается, должен содержать в себе информацию о типе, поскольку различие между ними может быть создано динамически. Эта особенность типична для ООП-кода. Каждый объект «знает» как на него должны воздействовать. Эта форма полиморфизма называется чистым полиморфизмом.

В C++ функции-методы класса с различным числом и типом параметров есть действительно различные функции, даже если они имеют одно и то же 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, C++ будет считать их различными и проигнорирует механизм виртуальных функций. Виртуальная функция – обязательно метод класса.

- ***Задание к лабораторной работе***

Общая постановка. Программа должна содержать:

- базовый класс **X**, включающий два элемента x1, x2 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 переменных классов.
2. Произведение переменных классов.
3. Сумму квадратов переменных классов.
4. Значение x1+x2 – y.
5. Значение (x1+x2)/y.
6. Значение (x1+x2)·y.
7. Значение x1·y+x2.
8. Значение x1+x2·y.
9. Произведение квадратов переменных класса.
10. Значение x1·x2+y.
11. Значение x1·x2/y.
12. Значение x1·x2-y.
13. Значение (x1-x2) ·y.
14. Значение (x1-x2)/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 C++ проигнорирует механизм виртуальных функций?

Лабораторная работа №6
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, список, дерево). Для этого в C++ 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 C++ 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 параметризованные
                                              //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 "`T`"**:**

```
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 параметризованы, т. е. генерироваться для каждого нового типа данных. Действительно, это так: функции-методы шаблона классов, в свою очередь, также являются шаблонными функциями с тем же параметром. То же самое касается переопределяемых операторов:

```
// параметр шаблона - класс "T", внутренний тип данных
// имя функции-элемента или оператора - параметризовано
//
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 лабораторной работе***

Общая постановка. Даны: число N и последовательность a_1, a_2, … a_N. Создать шаблон класса, порождающий динамические одномерные массивы с элементами различных типов (вещественные, целочисленные, символьные). 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, … ,aN/1.

- ***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 параметризованных объектов?

Лабораторная работа № 7
Множественное наследование с использованием абстрактных базовых классов, файлового ввода-вывода с применением потоков C++, 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:

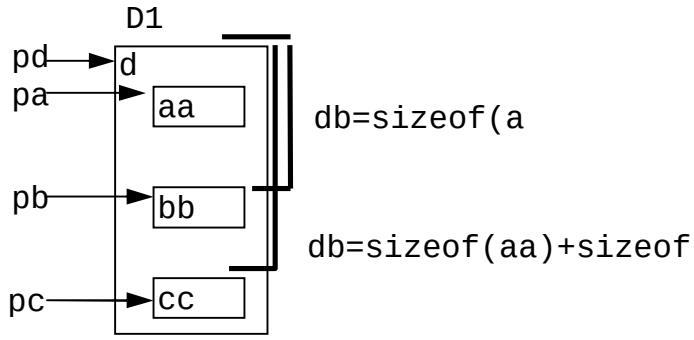
```
class base
     {
public:
virtual print()=0;
virtual get() =0;
     };
```

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 присоединением файла с дескриптором 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, будет выполнен последний – универсальный.

В случае невозникновения 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 расположен 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 уровням вложенности с продолжением вызова деструкторов.

- ***Задание к лабораторной 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, а также при вводе – выводе данных использо-\вать функции обработки исключительных ситуаций. Используя стандартные файловые потоки, информацию об объектах вывести в файл.

Варианты заданий

1. Книги (название, автор, жанр, год, количество страниц, тираж, отпечатано листов – функция Run ( )).

2. Транспорт (наименование, тип, год выпуска, максимальная скорость, объем двигателя, расход, объем бензобака, расстояние без подзаправки – функция Run ( )).

3. Продовольственные товары (наименование, отдел магазина, дата выпуска, срок хранения, последний срок реализации – функция Run ( ), вес).

4. Студенты (ФИО, год поступления, курс, дисциплины, оценки, средний балл – функция Run ( )).

5. Объекты недвижимости (адрес, тип, этажность, квартир на этаже, 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. Аудиостудия (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/с)).

- ***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).

Литература

1. Павловская, Т. А. С++ Объектно-ориентированное программирование / Т. А. Павловская, Ю. А. Щупак. – СПб. : Питер, 2004.

2. Карпов, Б. В. С++ – специальный справочник / Б. В. Карпов, Т. С. Баранова. – СПб. : Питер, 2001.

3. Эккель, Б. Философия С++. Практическое программирование / Б. Эккель, Ч. Эллисон. – СПб. : Питер, 2004.

4. Страуструп, Б. Язык программирования С++ / Б. Страуструп. – СПб.: Питер, 2005.

5. Бусько, В. Л. Основы ООП. С++ / В. Л. Бусько, А. Г. Корбит, Т. М. Кривоносова: лаб. практикум для студ. всех спец. и форм обуч. БГУИР. – Минск : БГУИР, 2005.

Учебное издание

Ганьшин Дмитрий Алексеевич
Снисаренко Светлана Валерьевна

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ПРОЕКТИРОВАНИЕ СИСТЕМ УПРАВЛЕНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебно-методическое пособие

Редактор С. Б. Саченко
Корректор Е. Н. Батурчик

Подписано в печать 28.04.2008.	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Печать ризографическая.	Усл. печ. л. 2,44.
Уч. изд. л. 2,0.	Тираж 150 экз.	Заказ 181.

Издатель и полиграфическое оформление: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ № 02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки,6