7. Каков будет результат выполнения следующего кода:

```
class A {
public:
int inc (int x) {return x++;}
int inc (short x) (return x+2;}
};
A obj; int y=5;
cout << obj.inc(y); ?
```

Варианты ответа:

*1) 5; 2) 6; 3) 7; 4) ошибка при компиляции.

8. Каков будет результат выполнения следующего кода:

```
class A {
public:
int y;
int inc (int x) {return y++;}
int inc (short x) {return x+y;}
};
A obj; int y=5; obj.y= 6;
cout << obj.inc(y); ?
```

Варианты ответа:

1) 5; *2) 6; 3) 11; 4) 7; 5) ошибка при компиляции.

9. Какими по умолчанию объявляются элементы структуры?

Варианты ответа:

1) private; *2) public; 3) protected; 4) по умолчанию не объявляются.

Лабораторная работа № 2
Тема. Разработка классов

Теоретическое введение. При разработке класса необходимо определить данные класса и его методы, конструкторы и деструкторы. Конструктор – это функция-член класса, которая вызывается автоматически при создании статического или динамического объекта класса. Он инициализирует объект и переменные класса. У конструктора нет возвращаемого значения, но он может иметь аргументы и быть перегружаемым.

Противоположные конструктору действия выполняет деструктор, который вызывается автоматически при уничтожении объекта. Деструктор имеет то же имя, что и класс, но перед ним стоит ‘~’. Деструктор можно вызывать явно в отличие от конструктора. Конструкторы и деструкторы не наследуются, хотя производный класс может вызывать конструктор базового класса.

Операторы-функции. Используются для введения операций над объектами, связываемых с символами:

+ , - , * , / , % , ^ , & , | , ~ , ! , = , < , > ,+= , [] , -> , ( ) , new, delete.

Оператор-функция является членом класса или дружественной (***friend***) классу. Общая форма оператор-функции-члена класса:

***возвращаемый_тип имя_класса*::operator#(список_аргум)**
{/*тело функции*/}

После этого вместо ***operator#(a,b)*** можно писать ***a#b***. Здесь # представляет один из введенных выше символов. Примерами являются операторы >> и << – перегружаемые операторы ввода-вывода. Отметим, что при перегрузке нельзя менять приоритет операторов и число операндов. Если оператор-функция-член класса перегружает бинарный оператор, то у функции будет только один параметр-объект, стоящий справа от знака оператора. Объект слева вызывает оператор-функцию и передается неявно с помощью указателя ***this***. В дружественную функцию указатель ***this*** не передается, поэтому унарный оператор имеет один параметр, а бинарный – два.

Оператор присваивания не может быть дружественной функцией, а только членом класса.

Пример. Создается класс Polynom. В головной программе выполняется тестирование класса.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
class Polynom {
 int n;
 double *koef;
 public:
 Polynom(); //конструкторы
 Polynom(int k);
 Polynom(int k,double *mas);
 Polynom(const Polynom&ob); //конструктор копирования
 ~Polynom(){delete[]koef;}
 void GiveMemory(int k);
 void SetPolynom(int k,double *mas);
 void SetDegree(int k){n=k;}; //установить степень
 void CalculateValue(double x);  //вычислить значение
 int GetDegree(){return n;};   //получить степень
 double GetOneCoefficient(int i){return(koef[i]);};
 Polynom operator+(Polynom ob); //перегрузка операторов
 Polynom operator*(Polynom ob);
 double& operator[](int i){return(koef[i]);}//перегрузка []
 Polynom& operator = (const Polynom p) {
   if(&p==this) return *this;
   if(koef) delete [] koef;
```

```
   n=p.n;
   koef=new double [p.n+1];
   for(int i=0;i<=p.n;i++)
      koef[i]=p.koef[i];
   return *this;
}
 friend ostream& operator<<(ostream& mystream,Polynom &ob);
 friend istream& operator>>(istream& mystream,Polynom &ob);
int min(int n,int m)
{return (n<m)? n:m; }
 int max(int n,int m)
 {return (n>m)? n:m; }
};
//********** Polynom() *********************************
Polynom::Polynom()
{  randomize();
 n=random(5);
 koef=new double[n+1];
 if(!koef){cout<<"Error";getch();return;}
 for(int i=n;i>=0;i--)
    koef[i]=random(10)-5;
}
//************* Polynom(int k) ******************************
Polynom::Polynom(int k)
{   n=k;
 koef=new double[n+1];
 if(!koef){cout<<"Error";getch();return;}
 for(int i=n;i>=0;i--)
    koef[i]=random(10)-5;
}
//****************** Polynom(int k,double mas[]) ******************
Polynom::Polynom(int k,double mas[])
{n=k;
 koef=new double[n+1];
 if(!koef){cout<<"Error";getch();return;}
 for(int i=n;i>=0;i--)
    koef[i]=mas[i];
}
//*************** Polynom(const Polynom&ob) *********************
Polynom::Polynom(const Polynom&ob)
{n=ob.n;
 koef=new double[n+1];
 if(!koef){cout<<"Error";getch();return;}
 for(int i=0;i<=n;i++)
   koef[i]=ob.koef[i];
}
//*************** void GiveMemory(int k) *********************
```

```
void Polynom::GiveMemory(int k)
{
 if(koef) delete [] koef;
 koef=new double[k+1];
 if(!koef){cout<<"Error";getch();return;}
}
//******************** SetPolynom *************************
void Polynom::SetPolynom(int k,double *mas)
{ n=k;
 if(koef) delete [] koef;
 koef = new double [n+1];
 for(int i=n;i>=0;i--)
     koef[i]=mas[i];
}
//*************** CalculateValue ****************************
void Polynom::CalculateValue(double x=1.0)
{ double s;
  int i;
 for(s=koef[0],i=1;i<=n;i++)
    s=s+koef[i]*pow(x,i);
 cout<<"f("<<x<<")="; cout<<s<<endl;
}
//***************** Polynom operator+(Polynom ob) ***************
Polynom Polynom::operator+(Polynom ob)
{ int i;
 Polynom rab;
 rab.GiveMemory(max(n,ob.GetDegree()));
 for(i=0;i<=min(n,ob.GetDegree());i++)
    rab.koef[i]=koef[i]+ob.GetOneCoefficient(i);
 if(n<ob.GetDegree())
   {
    for(i=min(n,ob.GetDegree())+1;i<=ob.GetDegree();i++)
       rab.koef[i]=ob.GetOneCoefficient(i);
    rab.n=ob.GetDegree();
   }
 else
   {
    for(i=min(n,ob.GetDegree())+1;i<=n;i++) rab.koef[i]=koef[i];
  rab.n=n;
   }
 return rab;
}
//*************** Polynom operator*(Polynom ob) ***************
Polynom Polynom::operator*(Polynom ob)
{
 int i,j,k;
 double s;
```

```
 Polynom rab;
 rab.GiveMemory(n+ob.GetDegree());
 for(i=0;i<=n+ob.GetDegree();i++)
    { s=0;
     for(j=0;j<=n;j++)
        for(k=0;k<=ob.GetDegree();k++)
          if(j+k==i)s=s+koef[j]*ob.GetOneCoefficient(k);
     rab.koef[i]=s;
    }
 rab.n=n+ob.GetDegree();
 return rab;
}
//********** ostream& operator<<(ostream& mystream,Polynom &ob) ******
ostream& operator<<(ostream& mystream,Polynom &ob)
{ char c=' '; //пропустим "+" перед первым коэффициентом
   for(int i=ob.n;i>=0;i--)
    { double ai=ob.koef[i];
        if(ai==0) continue;
          else {if(ai>0) mystream<<c; mystream<<ai;}
         if(i==0) continue; else mystream<<"x";
         if(i==1) continue; else mystream<<"^"<<i;
         if(ai!=0)c='+';
    }
 if(c==' ')mystream<<0;
 mystream<<endl;
 return mystream;
}
//********* istream& operator>>(istream& mystream,Polynom &ob) *
istream& operator>>(istream& mystream,Polynom &ob)
{
 int i;
 cout<<"Enter Degree:";  mystream>>ob.n; cout<<endl;
 for(i=ob.n;i>=0;i--)
    {
     cout<<"Enter koeff "<<i<<":";  mystream>>ob.koef[i];
    }
 return mystream;
}
//******************** MAIN ****************************
int main(int argc, char* argv[])
{ const int m=3;
   Polynom f,g,masp[m],*p1,s;
   int n=5,i;
   double K[6]={1.0,3.2,0.0,4.1,0.0,1.1};
    p1=new Polynom(n,K);
    cout<<*p1;
    p1->CalculateValue(2.0);
```

```
  cin>>f;
     cout<<" f(x)= ";  cout<<f;
    cout<<" g(x)= "; cout<<g;
  s=f+g;
  cout<<"f(x)+g(x) = "; cout<<s;
  s=f*g;
  cout<<" f(x)*g(x) = ";  cout<<s;
  s=masp[0]; cout<<masp[0];
  for(i=1;i<m;i++)
    { s=s+masp[i]; cout<<masp[i];}
  cout<<"Summa: ";  cout<< s;
  while(!kbhit());
  delete p1;
  return 0;
}
```

Задания для самостоятельного решения

Разработать перечисленные ниже классы. При разработке каждого класса возможны два варианта решения: а) данные-члены класса представляют собой переменные и массивы фиксированной размерности; б) память для данных-членов класса выделяется динамически.

1. «**Комплексное число**» – **Complex**. Класс должен содержать несколько конструкторов и операции для сложения, вычитания, умножения, деления, присваивания. Создать два вектора размерности n из комплексных координат. Передать их в функцию, которая выполняет сложение комплексных векторов.

2. Определить класс «**Дробь**» – **Fraction** в виде пары (m,n). Класс должен содержать несколько конструкторов. Реализовать методы для сложения, вычитания, умножения и деления дробей. Перегрузить операции сложения, вычитания, умножения, деления, присваивания и операции отношения. Создать массив объектов и передать его в функцию, которая изменяет каждый элемент массива с четным индексом путем добавления следующего за ним элемента массива.

3. Разработать класс «**Вектор**» – **Vector** размерности n. Определить несколько конструкторов, в том числе конструктор копирования. Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Перегрузить операции сложения, вычитания, умножения, инкремента, декремента, индексирования, присваивания для данного класса. Создать массив объектов. Написать функцию, которая для заданной пары векторов будет определять, являются ли они коллинеарными или ортогональными.

4. Определить класс «**Квадратная матрица**» – **Matrix**. Класс должен содержать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для сложения, вычитания, умножения матриц; вычисления нормы матрицы. Перегрузить операции сложения, вычитания, умножения и присваивания для данного класса. Создать массив объектов класса **Matrix** и передать его в функцию, которая изменяет i-ю матрицу путем возведения ее в квадрат. В головной программе вывести результат.

5. Разработать класс **«Многочлен» – Polynom** степени n. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для вычисления значения полинома; сложения, вычитания и умножения полиномов. Перегрузить операции сложения, вычитания, умножения, инкремента, декремента, индексирования, присваивания. Создать массив объектов класса. Передать его в функцию, вычисляющую сумму полиномов массива и возвращающую полином-результат, который выводится на экран в головной программе.

6. Определить класс **«Стек» – Stack**. Элементы стека хранятся в массиве. Если массив имеет фиксированную размерность, то предусмотреть контроль выхода за пределы массива. Если память выделяется динамически и ее не хватает, то увеличить размер выделенной памяти. Включение элементов в стек и их извлечение реализовать как в виде методов, так и с помощью перегруженных операций. Создать массив объектов класса **Stack**. Передавать объекты в функцию, которая удаляет из стека первый (сверху), третий, пятый и т. д. элементы.

7. Построить классы для описания плоских фигур: круг, квадрат, прямоугольник. Включить методы для изменения объектов, перемещения на плоскости, вращения. Перегрузить операции, реализующие те же действия. Выполнить тестирование класса, создав массив объектов.

8. Определить класс «**Строка**» – **String** длины n. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для выполнения конкатенации строк, извлечения символа из заданной позиции, сравнения строк. Перегрузить операции сложения, индексирования, отношения, добавления $(+=)$, присваивания для данного класса. Создать массив объектов и передать его в функцию, которая выполняет сортировку строк.

9. Разработать класс «**Множество (целых чисел, символов, строк** и т. д.)» – **Set** мощности n. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для определения принадлежности заданного элемента множеству, пересечения, объе-

динения, разности двух множеств. Перегрузить операции сложения, вычитания, умножения (пересечения), индексирования, присваивания. Создать массив объектов и передавать пары объектов в функцию, которая строит множество, состоящее из элементов, входящих только в одно из заданных множеств, т. е. $(A \cup B) \setminus (A \cap B)$, и возвращает его в головную программу.

10. Разработать класс для массива строк. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для поэлементной конкатенации двух массивов, упорядочения строк в лексикографическом порядке, слияния двух массивов с удалением повторяющихся строк, а также для вывода на экран всего массива и заданной строки. Перегрузить операции сложения, умножения, индексирования, присваивания для данного класса. Создать массив объектов и передавать объекты в функцию, которая выполняет слияние объектов и для полученного объекта-результата производит лексикографическое упорядочения строк.

11. Составить описания класса, обеспечивающего представление матрицы заданного размера $n \times m$ и любого минора в ней. Память для матрицы выделять динамически. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для отображения на экране как матрицы в целом, так и заданного минора, а также для изменения минора; сложения, вычитания, умножения миноров. Перегрузить операции сложения, вычитания, умножения и присваивания для данного класса. Создать массив объектов данного класса и передать его в функцию, которая изменяет для i-й матрицы ее минор путем умножения на константу.

12. Построить класс «**Булев вектор**» – **BoolVector** размерности n. Определить несколько конструкторов, в том числе конструктор копирования. Реализовать методы для выполнения поразрядных конъюнкции, дизъюнкции и отрицания векторов, а также подсчета числа единиц и нулей в векторе. Реализовать те же действия над векторами с помощью перегруженных операций. Перегрузить операции отношения и присваивания для данного класса. Создать массив объектов. Передавать объекты в функцию, которая будет их изменять по формуле $A = A \vee \overline{B}$.

13. Реализовать класс «**Троичный вектор**» – **Tvector** размерности n. Компоненты вектора принимают значения из множества $\{0, 1, X\}$. Два троичных вектора t_k=(t_1^k,…,t_n^k) и t_l=(t_1^l,…,t_n^l) называются ортогональными, если существует такое i, что t_i^k, $t_i^l \in \{0,1\}$ и $t_i^k \neq t_i^l$. Операция

пересечения не ортогональных векторов выполняется покомпонентно по следующим правилам: $1\cap1=1\cap X=X\cap1=1$, $0\cap0=0\cap X=X\cap0=0$, $X\cap X=X$. Реализовать методы для проверки векторов на ортогональность, для пересечения не ортогональных векторов, сравнения векторов, подсчета числа компонент, равных X. Осуществить те же действия над векторами с помощью перегруженных операций. Перегрузить операцию присваивания для данного класса. Выполнить тестирование класса, создав массив объектов.

14. Определить класс «**Булева матрица**» – **BoolMatrix** размерности $n\times m$. Класс должен содержать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для логического сложения (дизъюнкции), умножения и инверсии матриц. Реализовать методы для подсчета числа единиц в матрице и лексикографического упорядочения строк. Перегрузить операции для логического сложения, умножения и инверсии матриц, а также операцию присваивания. Создать массив объектов класса **BoolMatrix.** Передавать объекты в функцию, которая их изменяет по формуле $A=\overline{A}\vee\overline{B}$.

Тесты

1. В какой строке допущена ошибка?

```
class X {
   int a;
   int f() const;          //1
   int g() {return a++;}       //2
   int h() const {return a++;}    //3
};
```

Варианты ответа:

1) здесь нет ошибок; (*) 2) //3; 3) //2; 4) //1.

2. Выберите три верных утверждения:

[1] статическая функция-член класса, не обладает указателем this;
[2] статическая функция-член класса, может быть виртуальной;
[3] в классе не может быть двух функций: статической и нестатической – с одним и тем же именем и списком параметров;
[4] статическая функция в классе не может быть объявлена const.

Варианты ответа:

1) все утверждения не верны; (*) 2) [4]; (*) 3) [3]; 4) [2]; (*) 5) [1].

3 Правильно ли перегружены функции?

```
class X {
   static void f();
   void f() const;
};
```

Варианты ответа:

*1) нет; 2) да; 3) да, если убрать const.

4. Какие из операторов не могут быть перегружены?