ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ
Задания для практических занятий студентов 146, 147 групп

Занятие 1. Обзор ОО-возможностей языка С++

1.На основе примера, разобранного на лекции реализовать класс Cat. Листинг снабдить необходимыми комментариями. Класс Cat должен содержать:

- открытые и закрытые члены класса (its_age, its_weight);
- реализацию методов класса (meow, set_age, get_age);
- стандартный и пользовательский конструктор и деструктор (Cat(int initial_age));
- постоянные функции-члены(int get_age () const);
- раздельное объявление класса (файл cat.hpp) и реализацию (файл cat.cpp).

Листинг программы.

```
#include "class_cat.hpp"
int main(int argc, char* argv[])
{
  cat frisky(7);
  std::cout<<"frisky is "<<frisky.get_age()<<" years old.\n";
  frisky.set_age(5);
  frisky.meow();
  std::cout<<"frisky is "<<frisky.get_age()<<" years old.\n";
  frisky.meow();
  return 0;
}
============
#include <iostream>
class cat
{
public:
  cat(int initial_age);
  ~cat();
  int get_age() const;
  void set_age(int age);
  void meow() const;
private:
  int its_age;
};

int cat::get_age() const
{
  return (its_age);
}

void cat::meow() const
{
  std::cout<<"Meow!\n";
}

void cat::set_age (int age)
{
its_age=age;
}
```

2.Создать класс, содержащий другие классы в качестве данных-членов на примере иерархии сложного класса "Прямоугольник", имеющий в качестве вершин "Точки". Приведенный листинг демонстрирует этот подход.

```
#include <iostream.h>
class Point     // holds x,y coordinates
{
   public:
      void SetX(int x) { itsX = x; }
      void SetY(int y) { itsY = y; }
```

```
        int GetX()const { return itsX;}
        int GetY()const { return itsY;}
    private:
        int itsX;
        int itsY;
};    // end of Point class declaration


class  Rectangle
{
    public:
        Rectangle (int top, int left, int bottom, int right);
        ~Rectangle () {}

        int GetTop() const { return itsTop; }
        int GetLeft() const { return itsLeft; }
        int GetBottom() const { return itsBottom; }
        int GetRight() const { return itsRight; }

        Point  GetUpperLeft() const { return itsUpperLeft; }
        Point  GetLowerLeft() const { return itsLowerLeft; }
        Point  GetUpperRight() const { return itsUpperRight; }
        Point  GetLowerRight() const { return itsLowerRight; }

        void SetUpperLeft(Point Location)  {itsUpperLeft = Location;}
        void SetLowerLeft(Point Location)  {itsLowerLeft = Location;}
        void SetUpperRight(Point Location)  {itsUpperRight = Location;}
        void SetLowerRight(Point Location)  {itsLowerRight = Location;}

        void SetTop(int top) { itsTop = top; }
        void SetLeft (int left) { itsLeft = left; }
        void SetBottom (int bottom) { itsBottom = bottom; }
        void SetRight (int right) { itsRight = right; }

        int GetArea() const;

    private:
        Point  itsUpperLeft;
        Point  itsUpperRight;
        Point  itsLowerLeft;
        Point  itsLowerRight;
        int    itsTop;
        int    itsLeft;
        int    itsBottom;
        int    itsRight;
};
// end Rect.hpp

++++++++++++++++++++++++++++++++++++++++++++++
   // Begin rect.cpp
   #include "rect.hpp"
  Rectangle::Rectangle(int top, int left, int bottom, int right)
   {
         itsTop = top;
         itsLeft = left;
         itsBottom = bottom;
         itsRight = right;

        itsUpperLeft.SetX(left);
        itsUpperLeft.SetY(top);

        itsUpperRight.SetX(right);
        itsUpperRight.SetY(top);
        itsLowerLeft.SetX(left);
        itsLowerLeft.SetY(bottom);

        itsLowerRight.SetX(right);
        itsLowerRight.SetY(bottom);
}
int Rectangle::GetArea() const
{
        int Width = itsRight-itsLeft;
        int Height = itsTop - itsBottom;
        return (Width * Height);
}

int main()
{
        //initialize a local Rectangle variable
        Rectangle MyRectangle (100, 20, 50, 80 );
```

```
        int Area = MyRectangle.GetArea();

        cout << "Area: " << Area << "\n";
        cout << "Upper Left X Coordinate: ";
        cout << MyRectangle.GetUpperLeft().GetX();
      return 0;
 }
```

Занятие 2. Указатели

1.Модифицировать класс, разработанный на занятии 1, таким образом, чтобы продемонстрировать возможности по размещению и удалению объектов в области динамической памяти.

Листинг программы.

```
#include <iostream>
class simple_cat
{
public:
  simple_cat(){its_age=2;}
  ~simple_cat(){};
  int get_age() const {return its_age;}
  void set_age (int age) {its_age=age;}
private:
  int its_age;
};


int main(int argc, char* argv[])
{
  simple_cat * frisky = new simple_cat;
  std::cout<< "frisky is" << frisky->get_age()<<"years old.\n";
  (*frisky).set_age(5);
  std::cout<< "frisky is" << frisky->get_age()<<"years old.\n";
  delete frisky;
  return 0;
}
```

2.Напишите программу демонстрирующую доступ к данным-членам и использование указателя this.

Листинг программы.

```
#include <iostream>
class simple_cat
{
public:
  simple_cat();
  ~simple_cat();
  int get_age() const {return this->its_age;}
  void set_age (int age) {this->its_age=age;}
  int get_weight () const {return *its_weight;}
  void set_weight (int weight) {*its_weight=weight;}
private:
  int its_age;
  int *its_weight;
};

simple_cat::simple_cat()
{
  its_age=2;
  its_weight=new int (5);
}

simple_cat::~simple_cat()
{

  delete its_weight;
}


int main(int argc, char* argv[])
{
  simple_cat * frisky = new simple_cat;
  std::cout<< "frisky is" << frisky->get_age()<<"years old.\n";
  (*frisky).set_age(5);
  std::cout<< "frisky is" << frisky->get_age()<<"years old.\n";
  delete frisky;
  return 0;
}
```

Занятие 3. Ссылки

1.Модифицируйте класс Cat, так чтобы продемонстрировать работу ссылки как псевдонима объекта.

Листинг программы.

```
#include <iostream>
class simple_cat
{
public:
  simple_cat(int age, int weight);
  ~simple_cat(){};
  int get_age() {return its_age;}
  void set_age (int age) {its_age=age;}
  int get_weight () {return its_weight;}
  void set_weight (int weight) {its_weight=weight;}
private:
  int its_age;
  int its_weight;
};

simple_cat::simple_cat(int age, int weight)
{
  its_age=age;
  its_weight=weight;
}



int main(int argc, char* argv[])
{
  simple_cat frisky (5,8);
  simple_cat r_cat=frisky;
  std::cout<< "frisky is " << r_cat.get_age()<<" years old.\n";
  std::cout<< "frisky is " << r_cat.get_weight()<<" pounds.\n";
  return 0;
}
```

2.На основе примера, разобранного на лекции создать функцию, возвращающую несколько значений, например, квадрат и куб заданного числа.

3. На примере класса Cat продемонстрировать передачу по ссылке объекта с помощью указателя.

Листинг программы.

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

class simple_cat
{
public:
  simple_cat();
  simple_cat(simple_cat&);
  ~simple_cat();
};

simple_cat::simple_cat()
{
  cout<<"simple cat constructor\n";
}

simple_cat::simple_cat(simple_cat&)
{
  cout<<"simple cat copy constructor\n";
}

simple_cat::~simple_cat()
{
  cout<<"destructor\n";
}

simple_cat f1(simple_cat the_cat);
simple_cat* f2(simple_cat  *the_cat);


int main(int argc, char* argv[])
{
```

```
  cout<<"make a cat\n";
  simple_cat frisky;
  cout<<"calling f1\n";
  f1(frisky);
  cout<<"calling f2\n";
  f2(&frisky);
  return 0;
}

simple_cat f1(simple_cat the_cat)
{
  cout<<"f1 returning\n";
  return the_cat;
}

simple_cat* f2(simple_cat  *the_cat)
{
  cout<<"f2 returning\n";
  return the_cat;
}
```

4.Модифицировать листинг из п.3, так, чтобы продемонстрировать передачу постоянного указателя.

Листинг программы.

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

class simple_cat
{
public:
  simple_cat();
  simple_cat(simple_cat&);
  ~simple_cat();

  int get_age() const {return its_age;}
  void set_age (int age) {its_age=age;}

private:
  int its_age;
};

simple_cat::simple_cat()
{
  cout<<"simple cat constructor\n";
  its_age=1;
}

simple_cat::simple_cat(simple_cat&)
{
  cout<<"simple cat copy constructor\n";
}

simple_cat::~simple_cat()
{
  cout<<"destructor\n";
}

const simple_cat* const f2(const simple_cat const *the_cat);


int main(int argc, char* argv[])
{
  cout<<"make a cat\n";
  simple_cat frisky;
  cout<<"frisky is"<<frisky.get_age()<<"years old\n";
  int age=5;
  frisky.set_age(age);
  cout<<"frisky is"<<frisky.get_age()<<"years old\n";

  cout<<"calling f2\n";
  f2(&frisky);
  cout<<"frisky is"<<frisky.get_age()<<"years old\n";

  return 0;
}

const simple_cat* const f2(const simple_cat const *the_cat)
{
```

```
  cout<<"f2 returning\n";
  //the_cat->set_age(7);
  cout<<"frisky is now "<<the_cat->get_age()<<"years old\n";

  return the_cat;
}
```

Занятие 4. Дополнительные возможности функций

1.Продемонстрировать перегрузку функций-членов класса "Прямоугольник", функция член – "Нарисовать". Различать случаи собственно прямоугольника и квадрата.

Листинг программы.

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

class rectangle
{
public:
  rectangle(int width, int height);
  ~rectangle(){}

  void draw () const;
  void draw (int w, int h) const;

private:
  int its_width;
  int its_height;
};

rectangle::rectangle(int width, int height)
{
  its_width=width;
  its_height=height;
}

void rectangle::draw() const
{
  draw(its_width,its_height);
}

void rectangle::draw(int w, int h) const
{
  for(int i=0; i<h; i++)
  {
          for (int j=0;j<w;j++)
          {
                  cout<<"*";
          }
          cout<<"\n";
  }
}


int main(int argc, char* argv[])
{
  rectangle rect(30,5);
  cout<<"draw\n";
  rect.draw();

  cout<<"draw\n";
  rect.draw(10,4);

  return 0;
}
```

2.Продемонстрировать перегрузку конструкторов класса "Прямоугольник"

Листинг программы.

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

class rectangle
{
public:
  rectangle();
  rectangle(int width, int height);
```

```
  ~rectangle(){}
  int get_width () const {return its_width;}
  int get_lenght () const {return its_lenght;}
private:
  int its_width;
  int its_lenght;
};

rectangle::rectangle()
{
  its_width=5;
  its_lenght=6;
}

rectangle::rectangle(int width, int lenght)
{
  its_width=width;
  its_lenght=lenght;
}


int main(int argc, char* argv[])
{
  rectangle rect1;
  cout<<"rect1 width"<< rect1.get_width()<<"\n";
  cout<<"rect1 lenght"<< rect1.get_lenght()<<"\n";

  int a,b;

  cout<<"enter a width";
  cin>>a;

  cout<<"enter a lenght";
  cin>>b;

  rectangle rect2(a,b);
  cout<<"rect1 width"<< rect2.get_width()<<"\n";
  cout<<"rect1 lenght"<< rect2.get_lenght()<<"\n";

  return 0;
}
```

3.Создать класс Counter (счетчик). Перегрузить в нем операции инкремента и ++.

Листинг программы.

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

class counter
{
public:
  counter();
  ~counter(){}
  int get_its_value () const {return its_value;}
  void set_its_value (int x) {its_value=x;}
  void operator++ () {++its_value;}
  void increment  () {++its_value;}
private:
  int its_value;
};

counter::counter()
{
  its_value=0;
}

int main(int argc, char* argv[])
{
  counter i;
  cout<<"value of counter is "<< i.get_its_value()<<"\n";
  i.increment();
  cout<<"value of counter is "<< i.get_its_value()<<"\n";
  ++i;
  cout<<"value of counter is "<< i.get_its_value()<<"\n";
  return 0;
}
```

4.Модифицируйте листинг из п.3 для демонстрации перегрузки префиксного и постфиксного оператора инкремента.

Листинг программы.

```
#include <iostream>
using namespace std;
class counter
{
public:
  counter();
  ~counter(){}
  int get_its_value () const {return its_value;}
  void set_its_value (int x) {its_value=x;}
  const counter& operator++ ();
  const counter operator++ (int flag);
private:
  int its_value;
};

counter::counter()
{
  its_value=0;
}

const counter& counter::operator ++()
{
  ++its_value;
  return *this;
}

const counter counter::operator++ (int flag)
{
  counter temp(*this);
  ++its_value;
  return temp;
}

int main(int argc, char* argv[])
{
  counter i;
  cout<<"value of counter is "<< i.get_its_value()<<"\n";
  i++;
  cout<<"value of counter is "<< i.get_its_value()<<"\n";
  ++i;
  cout<<"value of counter is "<< i.get_its_value()<<"\n";

  counter a=++i;
  cout<<"value of a is "<< a.get_its_value()<<"\n";
  cout<<"value of counter is "<< i.get_its_value()<<"\n";

  a=i++;
  cout<<"value of a is "<< a.get_its_value()<<"\n";
  cout<<"value of counter is "<< i.get_its_value()<<"\n";

  return 0;
}
```

5.Перегрузите оператор + для объекта класса Counter.

Листинг программы.

```
#include <iostream>
using namespace std;
class counter
{
public:
  counter();
  counter(int initial_value);
  ~counter(){}
  int get_its_value () const {return its_value;}
  void set_its_value (int x) {its_value=x;}
  counter operator+ (const counter & rhs);
private:
  int its_value;
};

counter::counter()
{
  its_value=0;
}

counter::counter(int initial_value)
{
```

```
  its_value=initial_value;
}

counter counter::operator+ (const counter & rhs)
{
  return counter (its_value+rhs.get_its_value());
}

int main(int argc, char* argv[])
{
  int a=2;
  counter var1=a, var2(5),var3;
  var3=var1+var2;
  cout<<"var1="<<var1.get_its_value()<<"\n";
  cout<<"var2="<<var2.get_its_value()<<"\n";
  cout<<"var3="<<var3.get_its_value()<<"\n";
  return 0;
}
```

Занятие 5. Наследование

1.Продемонстрируйте пример простого наследования на примере иерархии классов "Животные", "Собаки".

Листинг программы.

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

enum breed {golden, cairn, dandie, shetland, doberman, lab};

class mammal
{
public:
  mammal(){its_age=2; its_weight=5;}
  ~mammal(){}

  int get_age () const {return its_age;}
  void set_age (int age) {its_age=age;}
  int get_weight () const {return its_weight;}
  void set_weight (int weight) {its_weight=weight;}

  void speak () const {cout<<"mammal sound\n";}
  void sleep () const {cout<<"mammal sleep\n";}
protected:
  int its_age;
  int its_weight;

};

class dog: public mammal
{
public:
  dog():its_breed (golden){};
  ~dog(){}

  breed get_breed () const {return its_breed;}
  void set_breed (breed br){its_breed=br;}

  void wag_tail () const {cout<<"tail wadding\n";}
  void beg_for_food () const {cout << "begging for food\n";}
private:
  breed its_breed;
};

int main(int argc, char* argv[])
{
  dog fido;
  fido.speak();
  fido.wag_tail();
  cout<<"fido is"<<fido.get_age()<<"years old\n";

  return 0;
}
```

2.Модифицируйте программу из п.1 чтобы продемонстрировать перегрузку конструкторов в производных классах.

3.Модифицируйте программу из п.1, чтобы продемонстрировать отсечение данных при передаче в виде значения.

Занятие 6. Полиморфизм

1.Продемонстрировать для одиночного наследования операцию переноса вверх (на примере Пегаса как лошади)

Листинг программы.

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

class horse
{
public:
  void gallop () {cout<<"Gallopping...\n";}
  virtual void fly () {cout<<"Horses can't fly.\n";}
private:
  int its_age;
};

class pegasus : public horse
{
public:
  virtual void fly () {cout<<"I can fly!!\n";}
};

const int number_of_horses=5;

int main(int argc, char* argv[])
{
  horse* ranch[number_of_horses];
  horse* p_horse;
  int choice, i;

  for (i=0; i<number_of_horses; i++)
  {
        cout<<"(1) - horse  (2) - pegas\n";
        cin>>choice;
        if (choice == 2)
               p_horse=new pegasus;
        else
               p_horse=new horse;
        ranch[i]=p_horse;
  }

  for (i=0; i<number_of_horses; i++)
  {
        ranch[i]->fly();
        delete ranch[i];
  }
  return 0;
}
```

2. Продемонстрировать для одиночного наследования операцию приведения вниз (на примере Пегаса как лошади)

Листинг программы.

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

enum type {HORSE, PEGAS};

class horse
{
public:
  virtual void gallop () {cout<<"Gallopping...\n";}
private:
  int its_age;
};

class pegasus : public horse
{
public:
  virtual void fly () {cout<<"I can fly!!\n";}
};

const int number_of_horses=5;

int main()
{
  horse* ranch[number_of_horses];
```

```
  horse* p_horse;
  int choice, i;

  for (i=0; i<number_of_horses; i++)
  {
         cout<<"(1) - horse  (2) - pegas\n";
         cin>>choice;
         if (choice == 2)
                p_horse=new pegasus;
         else
                p_horse=new horse;
         ranch[i]=p_horse;
  }

  for (i=0; i<number_of_horses; i++)
  {
         pegasus *p_peg = dynamic_cast<pegasus *> (ranch[i]);
         if(p_peg)
                p_peg->fly();
         else
                cout<<"Just a horse\n";
  delete ranch[i];
  }
  return 0;
}
```

3.Продемонстрировать множественное наследование (на примере Пегаса)

Листинг программы.

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

enum type {HORSE, PEGAS};

class horse
{
public:
  horse () {cout << "horse constructor\n";}
  virtual ~horse () {cout << "horse destructor\n";}
  virtual void whinny() const {cout<<"Whinny...\n";}
private:
  int its_age;
};

class bird
{
public:
  bird() {cout << "bird constructor\n";}
  virtual ~bird() {cout << "bird destructor\n";}
  virtual void chirp () const {cout << "chirp..\n";}
  virtual void fly () const {cout<<"I can fly!\n";}
private:
  int its_weight;
};

class pegasus : public horse, public bird
{
public:
  void chirp () const {whinny();}
  pegasus () {cout<<"pegasus constructor\n";}
  ~pegasus () {cout<<"pegasus destructor\n";}
};

const int num=2;

int main()
{
  horse* ranch[num];
  bird* aviary[num];

  horse * p_horse;
  bird * p_bird;

  int choice,i;

  for(i=0; i<num; i++)
  {
         cout<<"\n (1) -- horse  (2) -- pegasus\n";
         cin>> choice;
```

```
        if(choice==2)
                p_horse= new pegasus;
        else
                p_horse= new horse;
        ranch[i]=p_horse;
  }

  for(i=0; i<num; i++)
  {
        cout<<"\n (1) -- bird  (2) -- pegasus\n";
        cin>> choice;
        if(choice==2)
                p_bird= new pegasus;
        else
                p_bird= new bird;
        aviary[i]=p_bird;
  }

  cout<<"\n";
  for(i=0; i<num; i++)
  {
        cout<<"\n ranch["<<i<<"]: ";
        ranch[i]->whinny();
        delete ranch[i];
  }


  for(i=0; i<num; i++)
  {
        cout<<"\n aviary["<<i<<"]: ";
        aviary[i]->chirp();
        aviary[i]->fly();
        delete aviary[i];
  }

  return 0;
}
```

4.Продемонстрировать создание абстрактных типов данных и производных от них на примере класса "Фигура" и "Прямоугольник" и "Окружность".

Листинг программы.

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

class shape
{
public:
  shape(){}
  virtual ~shape(){}
  virtual long get_area(){return -1;}
  virtual long get_perim(){return -1;}
  virtual void draw (){}
private:
};

class circle: public shape
{
  circle(int radius);
  ~circle(){}
  long get_area () {return 3.1415926*its_radius*its_radius;}
  long get_perim() {return 2*3.1415926*its_radius;}
  void draw();
private:
  int its_radius;
};

circle::circle(int radius):
  its_radius(radius)
  {}

void circle::draw()
{
  cout<<"circle draw here\n";
}

class rectangle: public shape
{
public:
```

```
  rectangle(int len, int width):
    its_lenght(len),its_width(width){}
  virtual ~rectangle(){}
  virtual long get_area(){return its_lenght*its_width;}
  virtual long get_perim(){return 2*(its_lenght+its_width);}
  virtual int get_lenght(){return its_lenght;}
  virtual int get_width(){return its_width;}
  virtual void draw();
private:
  int its_width;
  int its_lenght;
};

void rectangle::draw()
{
  cout<<"rect\n";
  for(int i=1; i<3;i++)
  {
          for(int j=1; i<5;j++)
          {
                  cout<<"x";
          }
          cout<<"\n";
  }
}

class square :public rectangle
{
public:
  square(int len);
  square(int len, int width);
  ~square(){}
  long get_perim(){return 4*get_lenght();}
};

square::square(int len):
rectangle(len,len)
{}

square::square(int len, int width):
rectangle(len,width)
{
  if(get_lenght()!=get_width())
          cout<<"not a square\n";
}

int main()
{
  int choice;
  bool fquit=false;
  shape * sp;
/*
//        while(!fquit)
//        {
          cout<<"(1) - circle (2) - rectangle (3) - square (4) - quit: ";
          cin>>choice;
          switch (choice)
          {
                  case 0: fquit=true;
                          break;
                  case 1: //sp=new circle(4); sp->draw();
                          break;
                  case 2: sp=new rectangle(4,6); sp->draw();
                          break;
                  case 3: sp=new square(5); sp->draw();
                          break;
                  default: cout<<"enter a number between 0 and 3\n";
//                                continue;
                          break;
//                }
//                if(!fquit)
                  sp->draw();
          delete sp;
          sp=0;
          cout<<"\n";
  }
*/
  sp=new rectangle(5,4);
  //sp->draw();
  return 0;
```

```
}
```

Занятие 7. Задачи для самостоятельной разработки

Задание 1. Классы.

Создайте класс с именем Complex для выполнения арифметических действий с комплексными числами. Напишите программу для проверки вашего класса.

Комплексные числа имеют форму RealPart + ImaginaryPart * j, где j — квадратный корень из –1.

Используйте переменные с плавающей точкой для представления закрытых данных этого класса. Создайте функцию конструктор, которая позволяет объекту этого класса принимать начальные значения при его объявлении. Создайте открытые функции-члены для каждого из следующих пунктов:

1) Сложение двух комплексных чисел.
2) Вычитание двух комплексных чисел.
3) Умножение двух комплексных чисел.
4) Печать комплексного числа в форме (a, b), где a — действительная часть, b — мнимая часть числа.

Задание 2. Виртуальные функции

1. Разработать абстрактный класс - форма с интерфейсом площадь, периметр и распечатать. Создать производные от формы - круг, прямоугольник, треугольник, реализовав эти интерфейсы.
2. Разработать абстрактный тип - животные с интерфейсом печати, проверки живородящие или яйцо несущее. Создать производный класс - собаки, утконосы, реализовав в них соответствующие интерфейсы.

Задание 3. Классы.

1. Создать класс с именем Rational для выполнения арифметических действий с дробями. Напишите программу драйвера для проверки вашего класса.

Используйте целые данные для представления закрытых элементов класса — числителя и знаменателя. Создайте функцию конструктора, которая позволяет объекту этого класса принимать начальные значения при его объявлении. Конструктор должен содержать значения по умолчанию и должен хранить дроби в сокращенном виде (2/4 как ½). Создайте открытые функции элементы для следующих действий (результат должен храниться в сокращенной форме):

1) Сложение двух чисел.
2) Вычитание двух чисел.
3) Умножение двух чисел.
4) Деление двух чисел.

5) Печать чисел в форме a / b.
6) Печать чисел в форме с плавающей точкой.

Задание 4. Перегрузка операторов

1. Создать класс Дата, с конструкторами, деструктором, установить дату, увеличить на 1 день (++), уменьшить на один день, добавить дни +=. Перегрузить инкремент и декремент как постфиксный и префиксный, перегрузить ввод и вывод в поток.

Задание 5. Классы.

1. Создайте класс HugeInteger, который использует массив из 40 элементов для хранения целых чисел вплоть до очень больших целых, содержащих до 40 цифр. Создайте функции элементы для ввода, вывода, сложения и вычитания этих чисел. Создайте функции элементы для сравнения этих чисел (для реализации операций =, <>, <, <=, >, >=). Эти функции-предикаты должны возвращать значения Истина (1) или Ложь (0) после выполнения операции сравнения. Создайте предикатную функцию isZero (=0). На основе упомянутых функций создайте функции для умножения и деления таких чисел.

Задание 6. Перегрузка операторов

0. Создать класс rationalNumber для представления дробей со следующими возможностями:
1. Создайте конструктор, который предотвращает равенство нулю знаменателя дроби, сокращает и упрощает дроби, если они не сокращенной форме.
2. Перегрузите операции сложения, вычитания, умножения и деления для этого класса.
3. Перегрузите операции отношения и равенства для этого класса.

Задание 7. Перегрузка операторов

1. Создать класс Polinomials (полиномы) для представления полиномов вида $a_0 + a_1x^1 + a_2x^2 + \ldots + a_nx^n$. Внутренним представлением класса является массив членов полинома. Каждый член полинома содержит коэффициент и степень полинома. Член $2x^4$ имеет коэффициент 2 и показатель степени 4. Разработайте полный класс, содержащий конструктор, деструктор, функции set и get. Класс обеспечивать путем использования перегруженных операций следующие возможности:

1. Перегрузить операцию (+) для сложения полиномов.
2. Перегрузить операцию (–) для вычитания полиномов.
3. Перегрузить операцию (*) для умножения полиномов.
4. Перегрузить операции (+=), (–=), (*=).

Задание 8. Классы.

Цифровой счетчик - это переменная с ограниченным диапазоном, которая сбрасывается, когда ее целочисленное значение достигает определенного максимума. Примеры: цифровые часы, счетчик километража. Опишите класс такого счетчика. Обеспечьте возможность установления максимального и минимального значений, увеличения значений счетчика на 1, возвращения текущего значения. Класс реализовать на языке C++ .

Задание 9. Перегрузка операторов

2. Создать класс Matrix (матрицы) для представления квадратных матриц. Внутренним представлением матрицы является массив коэффициентов. Разработайте полный класс, содержащий конструктор, деструктор, функции set и get. Класс обеспечивать путем использования перегруженных операций следующими возможностями:

1. Перегрузить операцию (+) для сложения матриц.
2. Перегрузить операцию (–) для вычитания матриц.
3. Перегрузить операцию (*) для произведения матриц.