INDEX

Index : 01 : Overload the-- operator for the crood class. Create both its prefix and postfix forms.

Solution :

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

class coord{
    int x, y; //coordinate values
public:
    coord() { x=0; y=0;}
    coord(int i, int j) { x=i; y=j; }
    void get_xy(int &i, int &j) { i=x; j=y; }
    coord operator<<(int i);
    coord operator>>(int i);
};
//overload <<.
coord coord :: operator<<(int i)
{
  coord temp;
  temp.x = x << i;
  temp.y = y << i;

  return temp;
}
//overload <<.
coord coord :: operator>>(int i)
{
  coord temp;
  temp.x = x >> i;
  temp.y = y >> i;

  return temp;
}

int main()
{
    coord o1(4, 4), o2;
    int x, y;
    o2 = o1 << 2; //ob << int
    o2.get_xy(x, y);
    cout << "(o1<<2) X: " << x << ", Y: " << y << "\n";
    o2 = o1 >> 2; //ob >> int
    o2.get_xy(x, y);
    cout << "(o1>>2) X: " << x << ", Y: " << y << "\n";
    return 0;
}
```

Index : 02 : Overload the +operator for the crood class so that it is both binary operator and a unany operator.

Solution :

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

class coord {
  int x, y; // coordinate values
public:
  coord() { x=0; y=0; }
  coord(int i, int j) { x=i; y=j; }
  void get_xy(int &i, int &j) { i=x; j=y; }
  coord operator+(coord ob2); // binary plus
  coord operator+(); // unary plus
};

// Overload + relative to coord class.
coord coord::operator+(coord ob2)
{
  coord temp;

  temp.x = x + ob2.x;
  temp.y = y + ob2.y;

  return temp;
}

// Overload unary + for coord class.
coord coord::operator+()
{
  if(x<0) x = -x;
  if(y<0) y = -y;

  return *this;
}

int main()
{
  coord o1(10, 10), o2(-2, -2);
  int x, y;
  o1 = o1 + o2; // addition
  o1.get_xy(x, y);
  cout << "(o1+o2) X: " << x << ", Y: " << y << "\n";

  o2 = +o2; // absolute value
  o2.get_xy(x, y);
  cout << "(+o2) X: " << x << ", Y: " << y << "\n";
```

Index : 03 : Overload the -and/ operators for the crood class using friend functions.

Solution :

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

class coord {
  int x, y; // coordinate values
public:
  coord() { x=0; y=0; }
  coord(int i, int j) { x=i; y=j; }
  void get_xy(int &i, int &j) { i=x; j=y; }
  friend coord operator-(coord ob1, coord ob2);
  friend coord operator/(coord ob1, coord ob2);
};

// Overload - relative to coord class using friend.
coord operator-(coord ob1, coord ob2)
{
  coord temp;

  temp.x = ob1.x - ob2.x;
  temp.y = ob1.y - ob2.y;

  return temp;
}

// Overload / relative to coord class using friend.
coord operator/(coord ob1, coord ob2)
{
  coord temp;

  temp.x = ob1.x / ob2.x;
  temp.y = ob1.y / ob2.y;

  return temp;
}

int main()
{
  coord o1(10, 10), o2(5, 3), o3;
  int x, y;

  o3 = o1 - o2;
  o3.get_xy(x, y);
  cout << "(o1-o2) X: " << x << ", Y: " << y << "\n";

  o3 = o1 / o2;
  o3.get_xy(x, y);
  cout << "(o1/o2) X: " << x << ", Y: " << y << "\n";
  return 0;
}
```

Index 04 : Using a friend, show how to overload the – relative to the crood class. Define both the prefix and postfix forms.

Solution :

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

class coord {
  int x, y; // coordinate values
public:
  coord() { x=0; y=0; }
  coord(int i, int j) { x=i; y=j; }
  void get_xy(int &i, int &j) { i=x; j=y; }
  friend coord operator--(coord &ob); // prefix
  friend coord operator--(coord &ob, int notused); // postfix
};

// Overload -- (prefix) for coord class using a friend.
coord operator--(coord &ob)
{
  ob.x--;
  ob.y--;
  return ob;
}

// Overload -- (postfix) for coord class using a friend.
coord operator--(coord &ob, int notused)
{
  ob.x--;
  ob.y--;
  return ob;
}

int main()
{
  coord o1(10, 10);
  int x, y;

  --o1; // decrement o1 an object
  o1.get_xy(x, y);
  cout << "(--o1) X: " << x << ", Y: " << y << "\n";

  o1--; // decrement o1 an object
  o1.get_xy(x, y);
  cout << "(o1--) X: " << x << ", Y: " << y << "\n";
  return 0;
}
```

Index 05 : Strtype overloads the [] operator. Have this operator return the character at the specified index?

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

class strtype {
  char *p;
  int len;
public:
  strtype(char *s);
  ~strtype() {
     cout << "Freeing " << (unsigned) p << '\n';
     delete [] p;
   }
  char *get() { return p; }
  strtype &operator=(strtype &ob);
  char &operator[](int i);
};

strtype::strtype(char *s)
{
  int l;

  l = strlen(s)+1;

  p = new char [l];
  if(!p) {
    cout << "Allocation error\n";
    exit(1);
  }

  len = l;
  strcpy(p, s);
}

// Assign an object.
strtype &strtype::operator=(strtype &ob)
{
  // see if more memory is needed
  if(len < ob.len) { // need to allocate more memory
    delete [] p;
    p = new char [ob.len];
    if(!p) {
      cout << "Allocation error\n";
      exit(1);
    }
  }
  len = ob.len;
  strcpy(p, ob.p);
  return *this;
}

// Index characters in string.
char &strtype::operator[](int i)
{
  if(i<0 || i>len-1) {
    cout << "\nIndex value of ";
    cout << i << " is out-of-bounds.\n";
    exit(1);
  }
  return p[i];
}

int main()
{
  strtype a("Hello"), b("There");

  cout << a.get() << '\n';
  cout << b.get() << '\n';

  a = b; // now p is not overwritten

  cout << a.get() << '\n';
  cout << b.get() << '\n';

  // access characters using array indexing
  cout << a[0] << a[1] << a[2] << "\n";

  // assign characters using array indexing
  a[0] = 'X';
  a[1] = 'Y';
  a[2] = 'Z';

  cout << a.get() << "\n";

  return 0;
}
```

Index 06 : Overload the >> and<< shift operators relative to the crood class so that the following types of operations are allowed.

Solution :

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

class coord {
  int x, y; // coordinate values
public:
  coord() { x=0; y=0; }
  coord(int i, int j) { x=i; y=j; }
  void get_xy(int &i, int &j) { i=x; j=y; }
  coord operator<<(int i);
  coord operator>>(int i);
};

// Overload <<.
coord coord::operator<<(int i)
{
  coord temp;

  temp.x = x << i;
  temp.y = y << i;

  return temp;
}

// Overload >>.
coord coord::operator>>(int i)
{
  coord temp;

  temp.x = x >> i;
  temp.y = y >> i;

  return temp;
}

int main()
{
  coord o1(4, 4), o2;
  int x, y;

  o2 = o1 << 2;  // ob << int
  o2.get_xy(x, y);
  cout << "(o1<<2) X: " << x << ", Y: " << y << "\n";

  o2 = o1 >> 2; // ob >> int
  o2.get_xy(x, y);
  cout << "(o1>>2) X: " << x << ", Y: " << y << "\n";
  return 0;
}
```

Index : 07 : Given the following skeleton,fill in the constructor function for my derived. Have it pass along a pointer to an initialization string my base also have mydrived() initialize len to the length of the string.

Solution :

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

class mybase {
 char str[80];
public:
  mybase (char *s) {strcpy (str, s); }
  char *get() {return str; }
};

class myderived : public mybase {
  int len;
public:
    myderived(char *s) : mybase(s) {
    len = strlen(s);
    }
    int getlen() {return len; }
    void show() {cout << get() << '\n'; }
    };

    int main()
    {
        myderived ob("hello");

        ob.show();
        cout << ob.getlen() << '\n';

        return 0;
    }
```

Index : 08 : Using the following skeleton,create a appropriate car() and truck constructor function.

Solution :

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

//A base class for various types of vehicles.
class vehicle {
   int num_wheels;
   int range;
public:
    vehicle(int w,int r)
    {
    num_wheels = w; range = 1;
    }
    void showv()
    {
    cout << "Wheels: " << num_wheels << '\n';
    cout << "Range: " << range << '\n';
    }
 };

 class car : public vehicle {
 int passengers;
 public:
     car(int p, int w, int r) : vehicle(w,r)
     {
     passengers = p;
     }
     void show()
     {
         showv();
         cout << "Passengers: " << passengers << '\n';
     }
     };

     class truck : public vehicle {
     int loadlimit;
 public:
     truck(int l, int w, int r) : vehicle(w,r)
     {
     loadlimit = l;
     }
     void show()
     {
     showv();
     cout << "loadlimit: " << loadlimit << '\n';
     }
     };

    int main()
    {
        car c(5, 4, 500);
        truck t(30000, 12, 1200);

        cout << "Car: \n";
        c.show();
        cout << "\nTruck: \n";
        t.show();

        return 0;
    }
```

Index : 09 : Create a generic base class called building that stores the number of floors a building has.

Solution :

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

class building {
protected:
    int floors;
    int rooms;
    double footage;
};

class house : public building {
    int bedrooms;
    int bathrooms;
public:
    house(int f, int r, double ft, int br, int bth) {
    floors = f; rooms = r; footage = ft;
    bedrooms = br; bathrooms = bth;
  }
  void show() {
```

```
  cout << "floors: " << floors << '\n';
  cout << "rooms: " << rooms << '\n';
  cout << "square footage: " << footage << '\n';
  cout << "bedrooms: " << bedrooms << '\n';
  cout << "bathrooms: " << bathrooms << '\n';
  }
 };

 class office : public building {
  int phones;
  int extinguishers;
public:
  office(int f, int r, double ft, int p, int ext) {
   floors = f; rooms = r; footage = ft;
   phones = p; extinguishers = ext;
   }
   void show() {
   cout << "floors: " << floors << '\n';
   cout << "rooms: " << rooms << '\n';
   cout << "square footage: " << footage << '\n';
   cout << "Telephones: " << phones << '\n';
   cout << "fire extinguishers: ";
   cout << extinguishers << '\n';
   }
 };

 int main()
 {
     house h_ob(2, 12, 5000, 6, 4);
     office o_ob(4, 25, 12000, 30, 8);

     cout << "House: \n";
     h_ob.show();

     cout << "\nOffice: \n";
     o_ob.show();
     return 0;
 }
```

Index : 10 : Write a program that creates a base class called num. Have this class hold an integer value and contain a virtual function called shownum(). Create two derived classes called outhex and outoct that inherit num. Have the derived classes override shownum() so that it displays the value in hexadecimal and octal,respectively.

Solution :

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

```
class num
{
public:
   int i;
   num(int x){i=x;}
   virtual void shownum()  {cout<< "Integer : " << i<< "\n";}      //virtual function declaration in
base class.
};

class outhex:public num                   //outhex class inherits num class as public.
{
public:
   outhex(int n):num(n){}      //passing an integer value to base class’ constructor.
   void shownum(){cout << "Hexademical : " <<hex << i << '\n';}  //converting an integrer  to
hexademical
};

class outoct:public num                   //outoct class inherits num class as public.
{
public:
   outoct(int n):num(n){}      //passing an integer value to base class’ constructor.
   void shownum(){cout << "Octal : " <<oct << i << '\n';}             //converting an integrer  to octal.
};

main()
{
   num d(20191);
   outoct o(20191);
   outhex h(20191);

   d.shownum();
   o.shownum();
   h.shownum();

}
```

Index : 11 : Write a program that create a base class called dist that stores the distance between two points in a double variable. In dist, create a virtual function called trav_time() that outputs the time it takes to travel the distance. The spreed is 60 miles per hour. In a derived class called metric, override trav_time() so that it outputs the travel time assuming that the distance is in kilometers and the spreed is 100 kilometers per hour.

Solution :

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

class dist
{
public:
   double d;
```

```
    dist(int f){d=f;}
   //virtual function declaration in base class.
    virtual void trav_time()
    {
cout<<"Travle time at 60 mph: "<<d/60<<"\n";
     }
};

class metric : public dist                    // metric class inherits dist class as public.
{
public:
    metric(double f):dist(f){}
    void trav_time(){cout<<"Travle time at 100 kmp: "<<d/100<<"\n";}
};

main()
{
    dist *p;      //creating an object pointer.
    dist mph(90.56);
    metric kph(90);

    p=&mph;                //assigning the address of an object to the object pointer.
    p->trav_time();

    p=&kph;
    p->trav_time();
}
```

Index : 12 : Write a generic function, called min(), that returns the lesser of its two arguments. Demonstrate your function in a program.

Solution :

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

template <class X>

X minc(X a, X b)
{
    return a;
}


int main()
{
    int a=3,b=4;
    char c='c',d='a';
    cout<<minc(a,b)<<endl;
    cout<<minc(c,d)<<endl;
}
```