Chapter 10

Operator Overloading

Chapter 10 Topics(part 2)

- "=" Overloading
  - Default Assignment Operator Function
  - User-defined Assignment Operator Function
- "<< " and ">> " Overloading
  - Overloading Insertion Operator
  - Overloading Extraction Operator
- Data Type Conversion
  - Using Constructor
  - Using a Type Name as an Operator

Default Assignment Operator Function

SYNTAX

```
ClassName & ClassName ::operator = ( const ClassName & source )
{
  //copy the data members recursively
  ...
}
```

Default Assignment Operator Function for Class Complex

```
Complex & Complex::operator = (Complex & source)
{
   real= source.real;
   imag= source.imag;
   return *this;
}
```

Using Default Assignment Operator Function for Class `String`

```
#include <iostream>
#include <string>
using namespace std;
class String{
public:
   String(char * s)
   {
      ptr=new char[strlen(s)+1];
      strcpy(ptr,s);
   }
   ~String( ) { delete ptr; }
   void display( ) { cout<<ptr<<endl; }
private:
   char *ptr;
};
```

```
int main( )
{
   String s1("hello"); //
   cout<<"s1:";
   s1.display( );
   {
      String s2("hi"); //
      s2=s1; //
      cout<<"s2:";
      s2.display( );
    } //
    cout<<"s1:";
    s1.display( );
    return 0;
}
```

User-defined Assignment Operator Function

SYNTAX

```
class ClassName{
 public:
     ClassName & operator = ( const ClassName & );
  ...
 };
ClassName & ClassName ::operator = ( const ClassName & source )
{
  ...
}
```

User-defined Assignment Operator Function for Class `String`

```
String & String ::operator = ( const String & source )
{
  if(this==&source)  return *this;
  delete ptr;
  ptr=new char[strlen(source.ptr)+1];
  strcpy(ptr, source.ptr);
  return *this;
}
```

Using User-defined Assignment Operator Function for Class `String`

```
String & String ::operator = ( const String & source )
 { if(this==&source)  return *this;
    delete ptr;
    ptr=new char[strlen(source.ptr)+1];
    strcpy(ptr, source.ptr);
    return *this;
}
```

```
#include <iostream>
#include <string>
using namespace std;
class String{
public:
    String(char * s)
    {  ptr=new char[strlen(s)+1];
       strcpy(ptr,s); }
    ~String( ) { delete ptr; }
    void display( ) { cout<<ptr<<endl; }
 String & operator = ( const String &);
private:
    char *ptr;
};
```

```
int main( )
{  String s1("hello"); //
    cout<<"s1:";
    s1.display( );
    { String s2("hi"); //
       s2=s1; //
       cout<<"s2:";
       s2.display( ); } //
     cout<<"s1:";
     s1.display( );
     return 0;
}
```

Chapter 10 Topics(part 2)

- "=" Overloading
  - Default Assignment Operator Function
  - User-defined Assignment Operator Function
- "<< " and ">> " Overloading
  - Overloading Insertion Operator
  - Overloading Extraction Operator
- Data Type Conversion
  - Using Constructor
  - Using a Type Name as an Operator

Overloading Insertion Operator

PROTOTYPE

```
ostream & operator << (ostream & , ClassName & );
```

Output Complex Numbers

```
#include <iostream>
using namespace std;
class Complex{
public:
    Complex( ){real=0;imag=0;}
    Complex(double r,double i){real=r;imag=i;}
 friend Complex operator+(Complex &,Complex &);
friend ostream &operator<< (ostream &,Complex &);
private:
    double real;
    double imag;
};
Complex operator+(Complex &c1, Complex &c2)
{
    Complex temp;
    temp.real=c1.real+c2.real;
    temp.imag=c1.imag+c2.imag;
    return temp;
}
```

```
ostream &operator<< (ostream & output,
                                Complex &c)
{
    output<<“( ”<<c.real<<“ , ”
          <<c.imag<<“i )” <<endl;
    return output;
}
int main( )
{   Complex c1(3 , 4), c2(5 , -10), c3,c4;
    c3=c1+c2;
    c4=operator+(c1, c2);
    cout<<“c1=”<<c1;
    cout<< “c2=”<<c2;
    cout<< “c1+c2=”<<c3;
    cout<< “operator+(c1,c2)=”<<c4;
    return 0;
}
```

Overloading Extraction Operator

PROTOTYPE

```
istream & operator >> (istream & , ClassName & );
```

Input Complex Numbers

```
#include <iostream>
using namespace std;
class Complex{
public:
   Complex( ){real=0;imag=0;}
   Complex(double r,double i){real=r;imag=i;}
friend ostream &operator<< (ostream &,Complex &);
friend istream &operator>> (istream &,Complex &);
private:
   double real;
   double imag;
};
ostream &operator<< (ostream & output,Complex &c)
{
   output<<"( "<<c.real<<" , "
          <<c.imag<<"i )" <<endl;
   return output;
}
```

```
istream &operator>> (istream & input,
                     Complex &c)
{
   cout<<"Input the real part and"
       <<"imaginary part of a"
       <<"complex number:";
   input>>c.real>>c.imag;
   return input;
}
int main( )
{  Complex c1,c2;
   cin>>c1>>c2;
   cout<<"c1="<<c1;
   cout<< "c2="<<c2;
   return 0;
}
```

Chapter 10 Topics(part 2)

- "=" Overloading
  - Default Assignment Operator Function
  - User-defined Assignment Operator Function
- "<< " and ">> " Overloading
  - Overloading Insertion Operator
  - Overloading Extraction Operator
- Data Type Conversion
  - Using Constructor
  - Using a Type Name as an Operator

Using Constructor

A constructor that takes a single argument will be interpreted by the compiler as a rule for converting from the argument type to the class type.

```
class ClassName{
  public:
     ClassName(DataType );
  ...
};
```

Convert from double to Complex

```
#include <iostream>
using namespace std;
class Complex{
public:
   Complex( ){real=0;imag=0;}
   Complex(double r,double i){real=r;imag=i;}
   Complex(double r) {real=r;imag=0;}
 friend Complex operator+(Complex &,Complex &);
friend ostream &operator<< (ostream &,Complex &);
private:
   double real;
   double imag;
};
Complex operator+(Complex &c1, Complex &c2)
{   Complex temp;
   temp.real=c1.real+c2.real;
   temp.imag=c1.imag+c2.imag;
   return temp;
}
```

```
ostream &operator<< (ostream & output,
                                Complex &c)
{
   output<<“( ”<<c.real<<“ , ”
          <<c.imag<<“i )” <<endl;
   return output;
}
int main( )
{   Complex c1(3 , 4), c2,c3=3.2;
   c2=c1+Complex(2.5);
   cout<<“c1=”<<c1;
   cout<< “c2=”<<c2;
   cout<< “c3=”<<c3;
   return 0;
}
```

Using a Type Name as an Operator

As with all operators in C++, the programmer can provide a new meaning by defining the operator as a method.

```
class ClassName{
  public:
      operator DataType( );
  ...
}
```

Convert from Complex to double

```
#include <iostream>
using namespace std;
class Complex{
public:
    Complex( ){real=0;imag=0;}
    Complex(double r,double i){real=r;imag=i;}
    Complex(double r) {real=r;imag=0;}
    operator double( ) {return real;}
 friend Complex operator+(Complex &,Complex &);
friend ostream &operator<< (ostream &,Complex &);
private:
    double real;
    double imag;
};
Complex operator+(Complex &c1, Complex &c2)
{   Complex temp;
    temp.real=c1.real+c2.real;
    temp.imag=c1.imag+c2.imag;
    return temp;
}
```

```
ostream &operator<< (ostream & output,
                                    Complex &c)
{
    output<<“( ”<<c.real<<“ , ”
            <<c.imag<<“i )” <<endl;
    return output;
}
int main( )
{   Complex c1(3 , 4);
    double d;
    d=c1+2.5;
    cout<<“c1=”<<c1;
    cout<< “d=”<<d<<endl;
    return 0;
}
```