جَـامعة
المَـنارة
MANARA UNIVERSITY

Chapter 14 – More About Classes

14.1 Static Members

- If a member variable is declared *static*, all objects of that class have access to that variable. If a member function is declared *static*, it may be called before any instances of the class are defined.

Figure 14-1

جامعة المنارة
MANARA UNIVERSITY

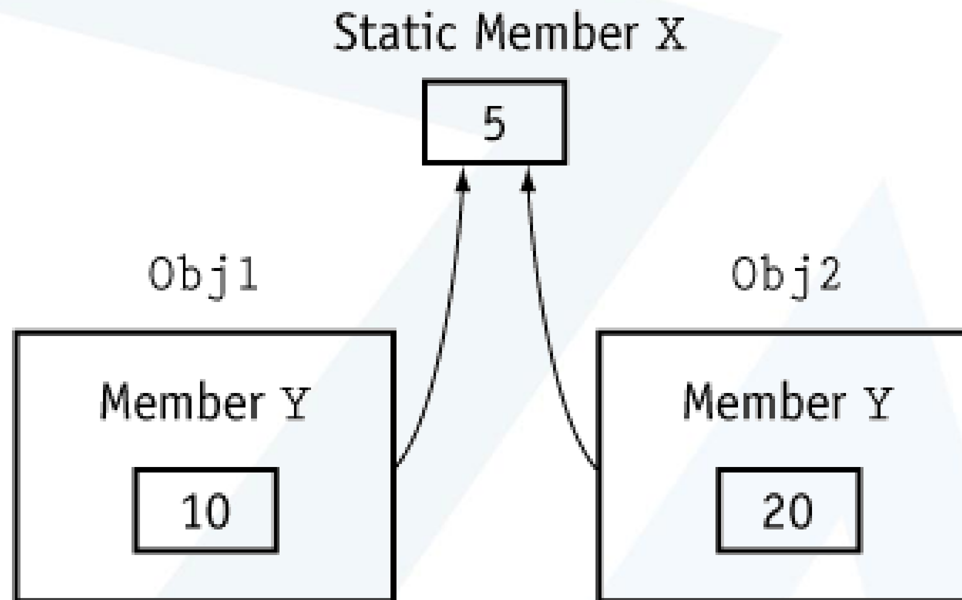
W1 Object

14.50	100
Price	Quantity

W2 Object

12.75	500
Price	Quantity

Figure 14-2



Program 14-1

Contents of budget.h

```
#ifndef BUDGET_H
#define BUDGET_H
// Budget class declaration
class Budget
{
  private:
        static float corpBudget;
        float divBudget;
  public:
        Budget(void) { divBudget = 0; }
        void addBudget(float b)
                { divBudget += b; corpBudget += divBudget; }
        float getDivBudget(void) { return divBudget; }
        float getCorpBudget(void) { return corpBudget; }
};
#endif
```

Program continues

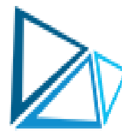
جَامعة
المَنارة
MANARA UNIVERSITY

Contents of main program, pr14-1.cpp

```
// This program demonstrates a static class member variable.
#include <iostream.h>
#include <iomanip.h>
#include "budget.h" // For Budget class declaration
float Budget::corpBudget = 0; // Definition of static member of Budget class
void main(void)
{
  Budget divisions[4];
  for (int count = 0; count < 4; count++)
  {
        float bud;
        cout << "Enter the budget request for division ";
        cout << (count + 1) << ": ";
        cin >> bud;
```

Program continues

```
        divisions[count].addBudget(bud);
  }
  cout.precision(2);
  cout.setf(ios::showpoint | ios::fixed);
  cout << "\nHere are the division budget requests:\n";
  for (int count = 0; count < 4; count++)
  {
        cout << "\tDivision " << (count + 1) << "\t$ ";
        cout << divisions[count].getDivBudget() << endl;
  }
  cout << "\tTotal Budget Requests:\t$ ";
  cout << divisions[0].getCorpBudget() << endl;
}
```



Program Output with Example Input

Enter the budget request for Division 1: **100000 [Enter]**
Enter the budget request for Division 2: **200000 [Enter]**
Enter the budget request for Division 3: **300000 [Enter]**
Enter the budget request for Division 4: **400000 [Enter]**
Here are the division budget requests:
Division 1 $ 100000.00
Division 1 $ 200000.00
Division 1 $ 300000.00
Division 1 $ 400000.00
Total Budget Requests: $ 1000000.00

Static Member Functions

static <return type><function name>(<parameter list>);

- Even though static member variables are declared in a class, they are actually defined outside the class declaration. The lifetime of a class’s static member variable is the lifetime of the program. This means that a class’s static member variables come into existence before any instances of the class are created.
- The static member functions of a class are callable before any instances of the class are created. This means that the static member functions of a class can access the class’s static member variables before any instances of the class are defined in memory. This gives you the ability to create very specialized setup routines for class objects.

Program 14-2

***Contents of* BUDGET2.H**

```
#ifndef BUDGET_H
#define BUDGET_H

class Budget
{
private:
   static float corpBudget;
   float divBudget;
public:
   Budget(void) { divBudget = 0; }
   void addBudget(float b)
         { divBudget += b; corpBudget += divBudget; }
   float getDivBudget(void) { return divBudget; }
   float getCorpBudget(void) { return corpBudget; }
   static void mainOffice(float);
};
#endif
```

Program continues

Contents of budget2.cpp

```
#include "budget2.h"

float Budget::corpBudget = 0; // Definition of static member of Budget
   class

// Definition of static member function MainOffice.
// This function adds the main office's budget request to
// the CorpBudget variable.

void Budget::mainOffice(float moffice)
{
   corpBudget += moffice;
}
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

Contents of main program, pr14-2.cpp

```
// This program demonstrates a static class member function.

#include <iostream.h>
#include <iomanip.h>
#include "budget2.h" // For Budget class declaration

void main(void)
{
  float amount;
  cout << "Enter the main office's budget request: ";
  cin >> amount;
  Budget::mainOffice(amount);
  Budget divisions[4];
```

Program continues

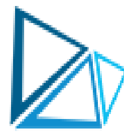
جَامعة
المَنارة

```
for (int count = 0; count < 4; count++)
{
	float bud;
	cout << "Enter the budget request for division ";
	cout << (count + 1) << ": ";
	cin >> bud;
	divisions[count].addBudget(bud);
}

cout.precision(2);
cout.setf(ios::showpoint | ios::fixed);
cout << "\nHere are the division budget requests:\n";
for (int count = 0; count < 4; count++)
{
	cout << "\tDivision " << (count + 1) << "\t$ ";
	cout << divisions[count].getDivBudget() << endl;
}
```

Program continues

جَامعة
المَـنارة
MANARA UNIVERSITY

```
  cout << "\tTotal Requests (including main office): $ ";
  cout << divisions[0].getCorpBudget() << endl;
}
```



Program Output with Example Input

Enter the main office's budget request: **100000 [Enter]**
Enter the budget request for Division 1: **100000 [Enter]**
Enter the budget request for Division 2: **200000 [Enter]**
Enter the budget request for Division 3: **300000 [Enter]**
Enter the budget request for Division 4: **400000 [Enter]**

Here are the division budget requests:
Division 1 $ 100000.00
Division 1 $ 200000.00
Division 1 $ 300000.00
Division 1 $ 400000.00
Total Requests (including main office): $ 1100000.00

14.2 Friends of Classes

- A friend is a function that is not a member of a class, but has access to the private members of the class.

friend <return type><function name>(<parameter type list>);

Program 14-3

Contents of auxil.h

```
#ifndef AUXIL_H
#define AUXIL_H

class Budget;  // Forward declaration of Budget class

// Aux class declaration
class Aux
{
private:
   float auxBudget;
public:
   Aux(void) { auxBudget = 0; }
   void addBudget(float, Budget &);
   float getDivBudget(void) { return auxBudget; }
};
#endif
```

جَامعة
المَـنارة
MANARA UNIVERSITY

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

Contents of budget3.h

```
#ifndef BUDGET3_H
#define BUDGET3_H
#include "auxil.h"    // For Aux class declaration

// Budget class declaration
class Budget
{
private:
   static float corpBudget;
   float divBudget;
public:
   Budget(void) { divBudget = 0; }
   void addBudget(float b)
         { divBudget += b; corpBudget += divBudget; }
```

Program continues

```
  float getDivBudget(void) { return divBudget; }
  float getCorpBudget(void) { return corpBudget; }
  static void mainOffice(float);
  friend void Aux::addBudget(float, Budget &);
};
#endif
```

Contents of budget3.cpp

```
#include "budget3.h"

float Budget::corpBudget = 0; // Definition of static member of Budget class

// Definition of static member function MainOffice.
// This function adds the main office's budget request to
// the CorpBudget variable.
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

```
void Budget::mainOffice(float moffice)
{
  corpBudget += moffice;
}
```

Contents of auxil.cpp

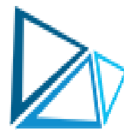
```
#include "auxil.h"
#include "budget3.h"

void Aux::addBudget(float b, Budget &div)
{
  auxBudget += b;
  div.corpBudget += auxBudget;
}
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

Contents of main program pr14-3.cpp

```
// This program demonstrates a static class member variable.
#include <iostream.h>
#include <iomanip.h>
#include "budget3.h"

void main(void)
{
  float amount;
  cout << "Enter the main office's budget request: ";
  cin >> amount;
  Budget::mainOffice(amount);
  Budget divisions[4];
  Aux auxOffices[4];
```



Program continues

```
for (int count = 0; count < 4; count++)
{
        float bud;
        cout << "Enter the budget request for division ";
        cout << (count + 1) << ": ";
        cin >> bud;
        divisions[count].addBudget(bud);
        cout << "Enter the budget request for division ";
        cout << (count + 1) << "'s\nauxiliary office: ";
        cin >> bud;
        auxOffices[count].addBudget(bud, divisions[count]);
}
cout.precision(2);
cout.setf(ios::showpoint | ios::fixed);
cout << "Here are the division budget requests:\n";
```

Program continues

```
for (count = 0; count < 4; count++)
{
	cout << "\tDivision " << (count + 1) << "\t\t\t$ ";

	cout << setw(7);
	cout << divisions[count].getDivBudget() << endl;
	cout << "\tAuxilary Office of Division " << (count+1);
	cout << "\t$  ";
	cout << auxOffices[count].getDivBudget() << endl;
}
cout << "\tTotal Requests (including main office): $ ";
cout << divisions[0].getCorpBudget() << endl;
}
```

Program Output with Example Input

Enter the main office's budget request: 100000 [Enter]
Enter the budget request for Division 1: 100000 [Enter]
Enter the budget request for Division 1's
auxilary office: 50000 [Enter]
Enter the budget request for Division 2: 200000 [Enter]
Enter the budget request for Division 2's
auxilary office: 40000 [Enter]
Enter the budget request for Division 3: 300000 [Enter]
Enter the budget request for Division 3's
auxilary office: 70000 [Enter]
Enter the budget request for Division 4: 400000 [Enter]
Enter the budget request for Division 4's
auxilary office: 65000 [Enter]
Here are the division budget requests:
Division 1: $ 100000.00
Auxilary office of Division 1: $ 50000.00
Division 2: $ 200000.00
Auxilary office of Division 2: $ 40000.00
Division 3: $ 300000.00
Auxilary office of Division 3: $ 70000.00
Division 4: $ 400000.00
Auxilary office of Division 4: $ 65000.00
Total Requests (including main office): $ 1325000.00

Friend classes

جَامعة
المَـنارة
MANARA UNIVERSITY

- As mentioned before, it is possible to make an entire class a friend of another class. The Budget class could make the Aux class its friend with the following declaration:

```
friend class Aux;
```

14.3 Memberwise Assignment

- The = operator may be used to assign one object to another, or to initialize one object with another object’s data. By default, each member of one object is copied to its counterpart in the other object.

Program 14-4

```
#include <iostream.h>

class Rectangle
{
private:
  float width;
  float length;
  float area;
  void calcArea(void) { area = width * length; }
 public:
  void setData(float w, float l)
        { width = w; length = l; calcArea(); }
  float getWidth(void)
        { return width; }
```

Program continues

```
  float getLength(void)
        { return length; }
  float getArea(void)
        { return area; }
};

void main(void)
{
  Rectangle box1, box2;
  box1.setData(10, 20);
  box2.setData(5, 10);
  cout << "Before the assignment:\n";
  cout << "Box 1's Width: " << box1.getWidth() << endl;
  cout << "Box 1's Length: " << box1.getLength() << endl;
  cout << "Box 1's Area: " << box1.getArea() << endl;
```

Program continues

```
    cout << "Box 2's Width: " << box2.getWidth() << endl;
    cout << "Box 2's Length: " << box2.getLength() << endl;
    cout << "Box 2's Area: " << box2.getArea() << endl;
    box2 = box1;
    cout << "----------------------\n";
    cout << "After the assignment:\n";
    cout << "Box 1's Width: " << box1.getWidth() << endl;
    cout << "Box 1's Length: " << box1.getLength() << endl;
    cout << "Box 1's Area: " << box1.getArea() << endl;
    cout << "Box 2's Width: " << box2.getWidth() << endl;
    cout << "Box 2's Length: " << box2.getLength() << endl;
    cout << "Box 2's Area: " << box2.getArea() << endl;
}
```

Program Output

جَامعة
المَـنارة
MANARA UNIVERSITY

Before the assignment:
Box 1's Width: 10
Box 1's Length: 20
Box 1's Area: 200
Box 2's Width: 5
Box 2's Length: 10
Box 2's Area: 50

After the assignment:
Box 1's Width: 10
Box 1's Length: 20
Box 1's Area: 200
Box 2's Width: 10
Box 2's Length: 20
Box 2's Area: 200

14.4 Copy Constructors

- A copy constructor is a special constructor, called whenever a new object is created and initialized with another object's data.
- PersonInfo person1("Maria Jones-Tucker",25);
- PersonInfo person2 = person1;

Figure 14-3

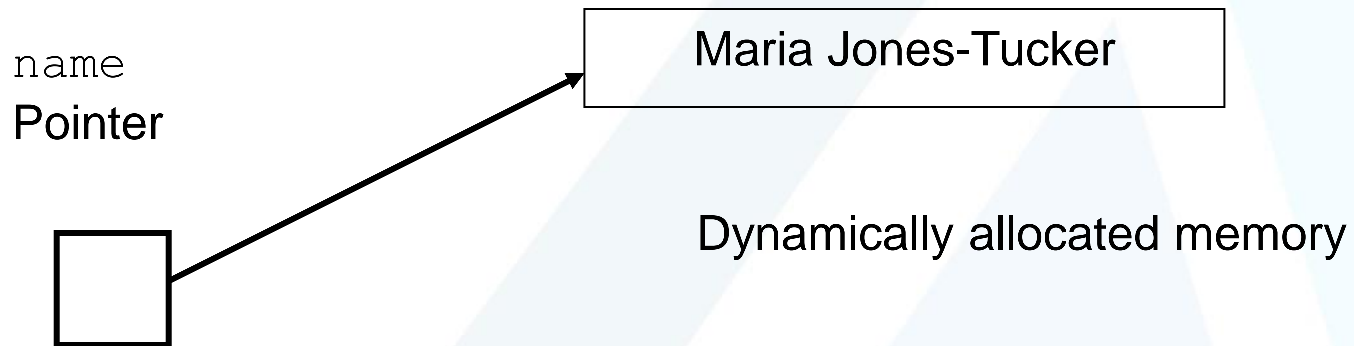
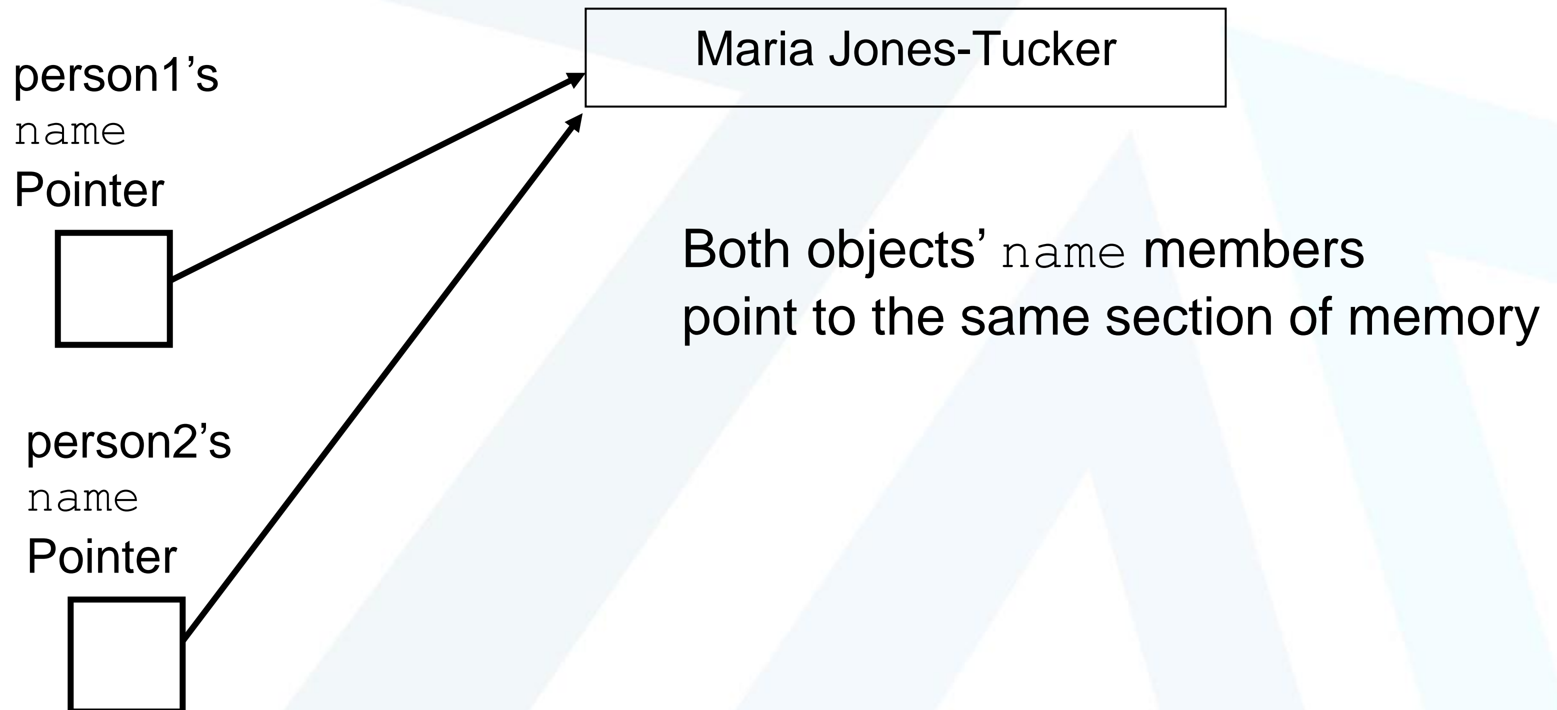


Figure 14-4



Using const Parameters

- Because copy constructors are required to use reference parameters, they have access to their argument's data. They should not be allowed to change the parameters, therefore, it is a good idea to make the parameter const so it can't be modified.

```
PersonInfo(const PersonInfo &Obj)
{   Name= new char[strlen(Obj.Name) + 1];
    strcpy(Name, Obj.Name);
    Age = Obj.Age;   }
```

The Default Copy Constructor

- If a class doesn’t have a copy constructor, C++ automatically creates a default copy constructor. The default copy constructor performs a memberwise assignment.

14.5 Operator Overloading

- C++ allows you to redefine how standard operators work when used with class objects.

Overloading the = Operator

```
void operator = (const PersonInfo &Right);
```

- Will be called with a statement like:

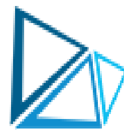
```
person2 = person1;
Or
person2.operator=(person1);
```

  - Note that the parameter, Right, was declared as a reference for efficiency purposes – the reference prevents the compiler form making a copy of the object being passed into the function.
  - Also note that the parameter was declared constant so the function will not accidentally change the contents of the argument.

Program 14-5

جَامعة
المَنارة
MANARA UNIVERSITY

```
// This program demonstrates the overloaded = operator.

#include <iostream.h>
#include <string.h>  // For strlen

class PersonInfo
{
  private:
        char *name;
        int age;

  public:
        PersonInfo(char *n, int a)
                { name = new char[strlen(n) + 1];
                  strcpy(name, n);
                  age = a; }
```



Program continues

```
        PersonInfo(const PersonInfo &obj) // Copy constructor
                { name = new char[strlen(obj.name) + 1];
                  strcpy(name, obj.name);
                  age = obj.age; }
        ~PersonInfo(void)
                { delete [] name; }
        char *getName(void)
                { return name; }
        int getAge(void)
                { return age; }
        void operator=(const PersonInfo &right)
                { delete [] name;
                  name = new char[strlen(right.name) + 1];
                  strcpy(name, right.name);
                  age = right.age; }
};
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

```
void main(void)
{
  PersonInfo jim("Jim Young", 27),
             bob("Bob Faraday", 32),
             clone = jim;
  cout << "The Jim Object contains: " << jim.getName();
  cout << ", " << Jim.GetAge() << endl;
  cout << "The Bob Object contains: " << bob.getName();
  cout << ", " << bob.getAge() << endl;
  cout << "The Clone Object contains: " << clone.getName();
  cout << ", " << clone.getAge() << endl;
  cout << "Now the clone will change to Bob and ";
  cout << "Bob will change to Jim.\n";
  clone = bob;        // Call overloaded = operator
  bob = jim;   // Call overloaded = operator
```

Program continues

```
  cout << "The Jim Object contains: " << jim.getName();
  cout << ", " << jim.getAge() << endl;
  cout << "The Bob Object contains: " << bob.getName();
  cout << ", " << bob.getAge() << endl;
  cout << "The Clone Object contains: " << clone.getName();
  cout << ", " << clone.getAge() << endl;
}
```

Program Output

جَـامعة
المَـنارة
MANARA UNIVERSITY

The Jim Object contains: Jim Young, 27
The Bob Object contains: Bob Faraday, 32
The Clone Object contains: Jim Young, 27
Now the clone will change to Bob and Bob will change to Jim.
The Jim Object contains: Jim Young, 27
The Bob Object contains: Jim Young, 27
The Clone Object contains: Bob Faraday, 32

The = Operator's Return Value

- If the operator= function returns a void, as in the above example, then multiple assignment statements won't work.
- To enable a statement such as

```
person3 = person2 = person1;
```

- You must have the following prototype:

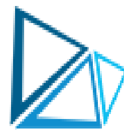
```
PersonInfo operator=(const PersonInfo &right);
```

The this Pointer

- *this is a special built in pointer that is available in any member function. *this contains the address of the object that called the member function.
- The this pointer is passed as a hidden argument to all non-static member functions.

Program 14-6

```
// This program demonstrates the overloaded = operator.

#include <iostream.h>
#include <string.h>  // For strlen

class PersonInfo
{
  private:
        char *name;
        int age;

  public:
        PersonInfo(char *n, int a)
                { name = new char[strlen(n)+ 1];
                  strcpy(name, n);
                  age = a; }
```



Program continues

```
    PersonInfo(const PersonInfo &obj) // Copy constructor
        { name = new char[strlen(obj.name)+ 1];
          strcpy(name, obj.name);
          age = obj.age; }
    ~PersonInfo(void)
        { delete [] name; }
    char *getName(void)
        { return name; }
    int getAge(void)
        { return age; }
    PersonInfo operator=(const PersonInfo &right)
        { delete [] name;
          name = new char[strlen(right.name) + 1];
          strcpy(name, right.name);
          age = right.age;
          return *this; }
};
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

```
void main(void)
{
   PersonInfo jim("Jim Young", 27),
              bob("Bob Faraday", 32),
              clone = jim;

   cout << "The Jim Object contains: " << jim.getName();
   cout << ", " << jim.getAge() << endl;
   cout << "The Bob Object contains: " << bob.getName();
   cout << ", " << bob.getAge() << endl;
   cout << "The Clone Object contains: " << clone.getName();
   cout << ", " << clone.getAge() << endl;
   cout << "Now the clone and Bob will change to Jim.\n";
   clone = bob = jim;   // Call overloaded = operator
   cout << "The Jim Object contains: " << jim.getName();
```

Program continues

```
  cout << ", " << jim.getAge() << endl;
  cout << "The Bob Object contains: " << bob.getName();
  cout << ", " << bob.getAge() << endl;
  cout << "The Clone Object contains: " << clone.getName();
  cout << ", " << clone.getAge() << endl;
}
```

Program Output

جَامعة
المَنارة
MANARA UNIVERSITY

The Jim Object contains: Jim Young, 27
The Bob Object contains: Bob Faraday, 32
The Clone Object contains: Jim Young, 27
Now the clone and Bob will change to Jim.
The Jim Object contains: Jim Young, 27
The Bob Object contains: Jim Young, 27
The Clone Object contains: Jim Young, 2

Some General Issues of Operator Overloading

- You can change an operator's entire meaning when you overload it. (But don't.)
- You cannot change the number of operands taken by an operator. For example, the = symbol must always be a binary operator. Likewise, ++ and – must always be unary operators.
- You cannot overload the ?: . .* :: and sizeof operators.

Table 14-1

+ - * / % ^ & | ~ ! = <

> += -= *= /= %= ^= &= |= << >> >>=

< == != <= >= && || ++ -- ->* , ->

[] () new delete

Program 14-7

Contents of feetinc2.h

```
#include <stdlib.h>  // Needed for abs()

#ifndef FEETINCHES_H
#define FEETINCHES_H

// A class to hold distances or measurements expressed in feet and inches.
class FeetInches
{
  private:
        int feet;
        int inches;
        void simplify(void); // Defined in feetinc2.cpp
  public:
        FeetInches(int f = 0, int i = 0)
                { feet = f; inches = i; simplify(); }
```

جَامعة المَنارة
MANARA UNIVERSITY

Program continues

```
        void setData(int f, int i)
                { feet = f; inches = i; simplify(); }
        int getFeet(void)
                { return feet; }
        int getInches(void)
                { return inches; }
        FeetInches operator + (const FeetInches &); // Overloaded +
        FeetInches operator - (const FeetInches &); // Overloaded -
};
#endif
```

Program continues

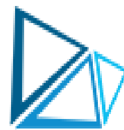
Contents of feetinc2.cpp

```
#include "feetinc2.h"
// Definition of member function Simplify. This function checks for values
// in the Inches member greater than twelve and less than zero. If such a
// value is found, the numbers in Feet and Inches are adjusted to conform
// to a standard feet & inches expression.
void FeetInches::simplify(void)
{
   if (inches >= 12)
   {
         feet += (inches / 12); // Integer division
         inches = inches % 12;
   }
   else if (inches < 0)
   {
         feet -= ((abs(inches) / 12) + 1);
         inches = 12 - (abs(inches) % 12);
   }
}
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

```
// Overloaded binary + operator.
FeetInches FeetInches::operator+(const FeetInches &right)
{
   FeetInches temp;
   temp.inches = inches + right.inches;
   temp.feet = feet + right.feet;
   temp.simplify();
   return temp;
}
// Overloaded binary - operator.
FeetInches FeetInches::operator-(const FeetInches &right)
{
   FeetInches temp;
   temp.inches = inches - right.inches;
   temp.feet = feet - right.feet;
   temp.simplify();
   return Temp;
}
```



Program continues

Contents of the main program file, pr14-7.cpp

```
// This program demonstrates the FeetInches class's overloaded
// + and - operators.

#include <iostream.h>
#include "feetinc2.h"

void main(void)
{
   FeetInches first, second, third;
   int f, i;
   cout << "Enter a distance in feet and inches: ";
   cin >> f >> i;
   first.setData(f, i);
   cout << "Enter another distance in feet and inches: ";
   cin >> f >> i;
```

Program continues

```
  second.setData(f, i);
  third = first + second;
  cout << "First + Second = ";
  cout << third.getFeet() << " feet, ";
  cout << third.getInches() << " inches.\n";
  third = first - second;
  cout << "First - Second = ";
  cout << third.getFeet() << " feet, ";
  cout << third.getInches() << " inches.\n";
}
```

Program Output with Example Input

Enter a distance in feet and inches: **6 5 [Enter]**

Enter another distance in feet and inches: **3 10 [Enter]**

First + Second = 10 feet, 3 inches.

First - Second = 2 feet, 7 inches.

Overloading the Prefix ++ Operator

```
FeetInches FeetInches::operator++(void)
{
  ++inches;
  simplify();
  return *this;
}
```

Overloading the Postfix ++ Operator

```
FeetInches FeetInches::operator++(int)
{
  FeetInches temp(feet,inches);
  inches++;
  simplify();
  return temp;
}
```

Program 14-8

Contents of feetinc3.h

```
#include <stdlib.h>  // Needed for abs()
#ifndef FEETINCHES_H
#define FEETINCHES_H

// A class to hold distances or measurements expressed
// in feet and inches.
class FeetInches
{
  private:
        int feet;
        int inches;
        void simplify(void); // Defined in feetinc3.cpp
  public:
        FeetInches(int f = 0, int i = 0)
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

```
		{ feet = f; inches = i; simplify(); }
	void setData(int f, int i)
		{ feet = f; inches = i; simplify(); }
	int getFeet(void)
		{ return feet; }
	int getInches(void)
		{ return inches; }
	FeetInches operator + (const FeetInches &); // Overloaded +
	FeetInches operator - (const FeetInches &); // Overloaded -
	FeetInches operator++(void);  // Prefix ++
	FeetInches operator++(int); // Postfix ++
};

#endif
```

Program continues

Contents of feetinc3.cpp

```
#include "feetinc3.h"

// Definition of member function Simplify. This function
// checks for values in the Inches member greater than
// twelve and less than zero. If such a value is found,
// the numbers in Feet and Inches are adjusted to conform
// to a standard feet & inches expression. For example,
// 3 feet 14 inches would be adjusted to 4 feet 2 inches and
// 5 feet -2 inches would be adjusted to 4 feet 10 inches.

void FeetInches::simplify(void)
{
   if (inches >= 12)
   {
         feet += (inches / 12); // Integer division
         inches = inches % 12;
   }
```

Program continues

```
  else if (inches < 0)
  {
        feet -= ((abs(inches) / 12) + 1);
        inches = 12 - (abs(inches) % 12);
  }
}
// Overloaded binary + operator.
FeetInches FeetInches::operator+(const FeetInches &right)
{
  FeetInches temp;
  temp.inches = inches + right.inches;
  temp.feet = feet + right.feet;
  temp.simplify();
  return temp;
}
```

Program continues

```
// Overloaded binary - operator.
FeetInches FeetInches::operator-(const FeetInches &right)
{
   FeetInches temp;
   temp.inches = inches - right.inches;
   temp.feet = feet - right.feet;
   temp.simplify();
   return temp;
}
// Overloaded prefix ++ operator. Causes the Inches member to
// be incremented. Returns the incremented object.
FeetInches FeetInches::operator++(void)
{
   ++inches;
   simplify();
   return *this;
}
```

Program continues

```
// Overloaded postfix ++ operator. Causes the Inches member to
// be incremented. Returns the value of the object before the
// increment.
FeetInches FeetInches::operator++(int)
{
  FeetInches temp(feet, inches);
  inches++;
  simplify();
  return temp;
}
```

Contents of the main program file, pr14-8.cpp

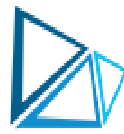
```
// This program demonstrates the FeetInches class's overloaded
// prefix and postfix ++ operators.
#include <iostream.h>
#include "feetinc3.h"
```

Program continues

```
void main(void)
{
  FeetInches first, second(1, 5);
  cout << "Demonstrating prefix ++ operator.\n";
  for (int count = 0; count < 12; count++)
  {
        first = ++second;
        cout << "First: " << first.getFeet() << " feet, ";
        cout << first.getInches() << " inches. ";
        cout << "Second: " << second.getFeet() << " feet, ";
        cout << second.getInches() << " inches.\n";
  }
  cout << "\nDemonstrating postfix ++ operator.\n";
  for (count = 0; count < 12; count++)
  {
        first = second++;
```

Program continues

```
        cout << "First: " << first.getFeet() << " feet, ";
        cout << first.getInches() << " inches. ";
        cout << "Second: " << second.getFeet() << " feet, ";
        cout << second.getInches() << " inches.\n";
  }
}
```



Program Output with Example Input

Demonstrating prefix ++ operator.
First: 1 feet 6 inches. Second: 1 feet 6 inches.
First: 1 feet 7 inches. Second: 1 feet 7 inches.
First: 1 feet 8 inches. Second: 1 feet 8 inches.
First: 1 feet 9 inches. Second: 1 feet 9 inches.
First: 1 feet 10 inches. Second: 1 feet 10 inches.
First: 1 feet 11 inches. Second: 1 feet 11 inches.
First: 2 feet 0 inches. Second: 2 feet 0 inches.
First: 2 feet 1 inches. Second: 2 feet 1 inches.
First: 2 feet 2 inches. Second: 2 feet 2 inches.
First: 2 feet 3 inches. Second: 2 feet 3 inches.
First: 2 feet 4 inches. Second: 2 feet 4 inches.
First: 2 feet 5 inches. Second: 2 feet 5 inches.

Output continues

جَامعة
المَنارة
MANARA UNIVERSITY

Demonstrating postfix ++ operator.
First: 2 feet 5 inches. Second: 2 feet 6 inches.
First: 2 feet 6 inches. Second: 2 feet 7 inches.
First: 2 feet 7 inches. Second: 2 feet 8 inches.
First: 2 feet 8 inches. Second: 2 feet 9 inches.
First: 2 feet 9 inches. Second: 2 feet 10 inches.
First: 2 feet 10 inches. Second: 2 feet 11 inches.
First: 2 feet 11 inches. Second: 3 feet 0 inches.
First: 3 feet 0 inches. Second: 3 feet 1 inches.
First: 3 feet 1 inches. Second: 3 feet 2 inches.
First: 3 feet 2 inches. Second: 3 feet 3 inches.
First: 3 feet 3 inches. Second: 3 feet 4 inches.
First: 3 feet 4 inches. Second: 3 feet 5 inches.

Overloading Relational Operators

```
if (distance1 < distance2)
{
  ... code ...
}
```

Relational operator example

```
int FeetInches:: operator>(const FeetInches &right)
{
  if (feet > right.feet)
    return 1;
  else
    if (feet == right.feet &&
        inches > right.inches)
    return 1;
  else
    return 0;
}
```

Overloading the [] Operator

- In addition to the traditional operators, C++ allows you to change the way the [] symbols work.

Program 14-12

```
#include <iostream.h>
#include "intarray.h"
void main(void)
{
  IntArray table(10);
  // Store values in the array.
  for (int x = 0; x < 10; x++)
        table[x] = x;
  // Display the values in the array.
  for (int x = 0; x < 10; x++)
        cout << table[x] << " ";
  cout << endl;
  // Use the built-in + operator on array elements.
   for (int x = 0; x < 10; x++)
          table[x] = table[x] + 5;
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

```
  // Display the values in the array.
  for (int x = 0; x < 10; x++)
          cout << table[x] << " ";
  cout << endl;
  // Use the built-in ++ operator on array elements.
   for (int x = 0; x < 10; x++)
              table[x]++;
  // Display the values in the array.
  for (int x = 0; x < 10; x++)
          cout << table[x] << " ";
  cout << endl;
}
```

Program Output

0 2 4 6 8 10 12 14 16 18
5 7 9 11 13 15 17 19 21 23
6 8 10 12 14 16 18 20 22 24

Program 14-13

```
#include <iostream.h>
#include "intarray.h"
void main(void)
{
   IntArray table(10);
   // Store values in the array.
   for (int x = 0; x < 10; x++)
         table[x] = x;
   // Display the values in the array.
   for (int x = 0; x < 10; x++)
         cout << table[x] << " ";
   cout << endl;
   cout << “Now attempting to store a value in table[11].
   table[11] = 0;
}
```

Program Output

0 1 2 3 4 5 6 7 8 9

Now attempting to store a value in table[11].

ERROR: Subscript out of range.

14.6 Object Conversion

- Special operator functions may be written to convert a class object to any other type.

```
FeetInches::operator float(void)
{
    float temp = feet;
    temp += (inches / 12.0);
    return temp;
}
```

Note:

No return type is specified in the function header for the previous example. Because it is a FeetInches-to-float conversion function, it will always return a float.

Program 14-13

Contents of feetinc6.h

```
#include <iostream.h> // Needed to overload << and >>
#include <stdlib.h>   // Needed for abs()
#ifndef FEETINCHES_H
#define FEETINCHES_H

// A class to hold distances or measurements expressed
// in feet and inches.
class FeetInches
{
  private:
        int feet;
        int inches;
        void simplify(void); // Defined in feetinc3.cpp
```

Program continues

```
public:
        FeetInches(int f = 0, int i = 0)
                { feet = f; inches = i; simplify(); }
        void setData(int f, int i)
                { feet = f; inches = i; simplify(); }
        int getFeet(void)
                { return feet; }
        int getInches(void)
                { return inches; }
        FeetInches operator + (const FeetInches &); // Overloaded +
        FeetInches operator - (const FeetInches &); // Overloaded -
        FeetInches operator++(void);  // Prefix ++
        FeetInches operator++(int); // Postfix ++
        int operator>(const FeetInches &);
        int operator<(const FeetInches &);
        int operator==(const FeetInches &);
```

Program continues

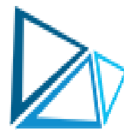
```
        operator float(void);
        operator int(void) // Truncates the Inches value
                { return feet; }
        friend ostream &operator<<(ostream &, FeetInches &);
        friend istream &operator>>(istream &, FeetInches &);
};
#endif
```

Contents of feetinc6.cpp

```
#include "feetinc6.h"
// Definition of member function Simplify. This function
// checks for values in the Inches member greater than
// twelve and less than zero. If such a value is found,
// the numbers in Feet and Inches are adjusted to conform
// to a standard feet & inches expression.
```

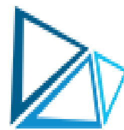
Program continues

```
void FeetInches::simplify(void)
{
  if (inches >= 12)
  {
        feet += (inches / 12); // Integer division
        inches = inches % 12;
  }
  else if (Inches < 0)
  {
        feet -= ((abs(inches) / 12) + 1);
        inches = 12 - (abs(inches) % 12);
  }
}
```



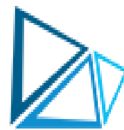
Program continues

```
// Overloaded binary + operator.
FeetInches FeetInches::operator+(const FeetInches &right)
{
  FeetInches temp;
  temp.inches = inches + right.inches;
  temp.feet = feet + right.feet;
  temp.simplify();
  return temp;
}
// Overloaded binary - operator.
FeetInches FeetInches::operator-(const FeetInches &right)
{
  FeetInches temp;
  temp.Inches = inches - right.inches;
  temp.feet = feet - right.feet;
  temp.simplify();
  return temp;
}
```



Program continues

```
// Overloaded prefix ++ operator. Causes the Inches member to
// be incremented. Returns the incremented object.
FeetInches FeetInches::operator++(void)
{
   ++inches;
   simplify();
   return *this;
}
// Overloaded postfix ++ operator. Causes the Inches member to
// be incremented. Returns the value of the object before the increment.
FeetInches FeetInches::operator++(int)
{
   FeetInches temp(feet, inches);
   inches++;
   simplify();
   return temp;
}
```

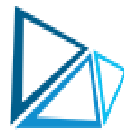


Program continues

```
// Overloaded > operator. Returns 1 if the current object is
// set to a value greater than that of Right.
int FeetInches::operator>(const FeetInches &right)
{
   if (feet > right.feet)
        return 1;
   else if (feet == right.feet && inches > right.inches)
        return 1;
   else return 0;
}
// Overloaded < operator. Returns 1 if the current object is
// set to a value less than that of Right.
int FeetInches::operator<(const FeetInches &right)
{
   if (feet < right.feet)
        return 1;
```

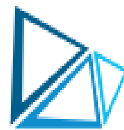
Program continues

```
  else if (Feet == Right.Feet && Inches < Right.Inches)
        return 1;
  else return 0;
}

// Overloaded == operator. Returns 1 if the current object is
// set to a value equal to that of Right.
int FeetInches::operator==(const FeetInches &right)
{
  if (feet == right.feet && inches == right.inches)
        return 1;
  else return 0;
}
```



Program continues

```
// Conversion function to convert a FeetInches object to a float.
FeetInches::operator float(void)
{
  float temp = feet;
  temp += (inches / 12.0);
  return temp;
}

// Overloaded << operator. Gives cout the ability to
// directly display FeetInches objects.
ostream &operator<<(ostream &strm, FeetInches &obj)
{
  strm << obj.feet << " feet, " << obj.inches << " inches";
  return strm;
}
```



Program continues

```
// Overloaded >> operator. Gives cin the ability to
// store user input directly into FeetInches objects.
istream &operator>>(istream &strm, FeetInches &obj)
{
  cout << "Feet: ";
  strm >> obj.feet;
  cout << "Inches: "
  strm >> obj.inches;
  return strm;
}
```

Main program file, pr14-13.cpp

```
// This program demonstrates the << and >> operators,
// overloaded to work with the FeetInches class.
#include <iostream.h>
#include "feetinc5.h"
```

Program continues

جَامعة
المَنارة
MANARA UNIVERSITY

```
void main(void)
{
   FeetInches distance;
   float f;
   int i;
   cout << "Enter a distance in feet and inches:\n ";
   cin >> distance;
   f = distance;
   i = distance;
   cout << "The value " << distance;
   cout << " is equivalent to " << f << " feet\n";
   cout << "or " << i << " feet, rounded down.\n";
}
```

Program Output with Example Input

Enter a distance in feet and inches:
Feet: **8 [Enter]**
Inches: **6 [Enter]**
The value 8 feet, 6 inches is equivalent to 8.5 feet
or 8 feet, rounded down.

14.7 Creating a String Class

- This section shows the use of a C++ class to create a string data type.

The MyString class

- Memory is dynamically allocated for any string stored in a MyString object.
- Strings may be assigned to a MyString object with the = operator.
- One string may be concatenated to another with the += operator.
- Strings may be tested for equality with the == operator.

MyString

جَامعة
المَنارة
MANARA UNIVERSITY

```
#ifndef MYSTRING_H
#define MYSTRING_H
#include <iostream.h>
#include <string.h>  // For string library functions
#include <stdlib.h>  // For exit() function
// MyString class. An abstract data type for handling strings.
class MyString
{  private:
        char *str;
        int len;
        void memError(void);
   public:
        MyString(void) { str = NULL; len = 0; }
        MyString(char *);
        MyString(MyString &); // Copy constructor
        ~MyString(void) { if (len != 0) delete [] str; }
```

MyString continues

```
        int length(void) { return len; }
        char *getValue(void) { return str; };
        MyString operator+=(MyString &);
        char *operator+=(const char *);
        MyString operator=(MyString &);
        char *operator=(const char *);
        int operator==(MyString &);
        int operator==(const char *);
        int operator!=(MyString &);
        int operator!=(const char *);
        int operator>(MyString &);
        int operator>(const char *);
        int operator<(const char *);
        int operator<(MyString &);
        int operator>=(MyString &);
        int operator<=(const char *);
        friend ostream &operator<<(ostream &, MyString &);
        friend istream &operator>>(istream &, MyString &);
};
#endif
```

Contents of mystring.cpp

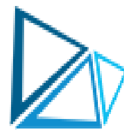
```
#include "mystring.h"
// Definition of member function MemError.
// This function is called to terminate a program
// if a memory allocation fails.
void MyString::memError(void)
{
   cout << "Error allocating memory.\n";
   exit(0);
}
// Constructor to initialize the Str member
// with a string constant.
MyString::MyString(char *sptr)
{
   len = strlen(sptr);
   str = new char[len + 1];
   if (str == NULL)
         memError();
   strcpy(str, sptr);
}
```

MyString continues

```
// Copy constructor
MyString MyString::operator=(MyString &right)
{
  str = new char[right.length() + 1];
  if (str == NULL)
        memError();
  strcpy(str, right.getValue());
  len = right.length();
}
```

MyString continues

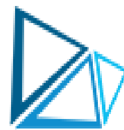
```
// Overloaded = operator. Called when operand
// on the right is another MyString object.
// Returns the calling object.
MyString MyString::operator=(MyString &right)
{
  if (len != 0)
         delete [] str;
  str = new char[right.length() + 1];
  if (str == NULL)
         memError();
  strcpy(str, right.getValue());
  len = right.length();
  return *this;
}
```



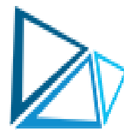
MyString continues

```
// Overloaded = operator. Called when operand
// on the right is a string.
// Returns the Str member of the calling object.
char *MyString::operator=(const char *right)
{
  if (len != 0)
        delete [] str;
  len = strlen(right);
  str = new char[len + 1];
  if (str == NULL)
        memError();
  strcpy(str, right);
  return str;
}
```

MyString continues




```
// Overloaded += operator. Called when operand on the right is another
// MyString object. Concatenates the Str member of Right to the Str member of
// the calling object. Returns the calling object.
MyString MyString::operator+=(MyString &right)
{
  char *temp = str;
  str = new char[strlen(str) + right.length() + 1];
  if (str == NULL)
        memError();
  strcpy(str, temp);
  strcat(str, right.getvalue());
  if (len != 0)
        delete [] temp;
  len = strlen(str);
  return *this;
}
```

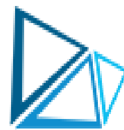


MyString continues

```
// Overloaded += operator. Called when operand on the right is a string.
// Concatenates the Str member of Right to the Str member of the calling
  object.

// Returns the Str member of the calling object.
char *MyString::operator+=(const char *right)
{
  char *temp = str;
  str = new char[strlen(str) + strlen(right) + 1];
  if (str == NULL)
        memError();
  strcpy(str, temp);
  strcat(str, right);
  if (len != 0)
        delete [] temp;
  return str;
}
```

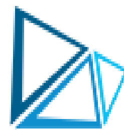
MyString continues

```
// Overloaded == operator.  Called when the operand on the right is a
  MyString
// object. Returns 1 if Right.Str is the same as Str.
int MyString::operator==(MyString &right)
{
  return !strcmp(str, right.getValue());
}

// Overloaded == operator. Called when the operand on the right is a
  string.
// Returns 1 if Right is the same as Str.
int MyString::operator==(const char *right)
{
  return !strcmp(str, right);
}
```



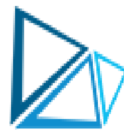
MyString continues

```
// Overloaded != operator. Called when the operand on the right is a
  MyString
// object. Returns 1 if Right.Str is not equal to Str.
int MyString::operator!=(MyString &right)
{
  return strcmp(str, right.getValue());
}

// Overloaded != operator. Called when the operand on the right is a
  string.
// Returns 1 if Right is not equal to Str.
int MyString::operator!=(const char *right)
{
  return strcmp(str, right);
}
```



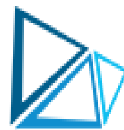
MyString continues

```
// Overloaded > operator. Called when the operand on the right is a
  MyString
// object. Returns 1 if Right.Str is greater than Str.
int MyString::operator>(MyString &right)
{
  if (strcmp(str, right.getValue()) > 0)
        return 1;
  else
        return 0;
}
```



MyString continues

```
// Overloaded > operator. Called when the operand on the right is a string.
// Returns 1 if Right is greater than Str.
int MyString::operator>(const char *right)
{
   if (strcmp(str, right) > 0)
         return 1;
   else
         return 0;
}
// Overloaded < operator. Called when the operand on the right is a
// MyString object. Returns 1 if Right.Str is less than Str.
int MyString::operator<(MyString &right)
{
   if (strcmp(str, right.getValue()) < 0)
         return 1;
   else
         return 0;
}
```



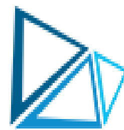
MyString continues

```
// Overloaded < operator. Called when the operand on the right is a string.
// Returns 1 if right is less than str.
int MyString::operator<(const char *right)
{
  if (strcmp(str, right) < 0)
        return 1;
  else
        return 0;
}
// Overloaded >= operator. Called when the operand on the right is a
// MyString object. Returns 1 if right.str is greater than or equal to str.
int MyString::operator>=(MyString &right)
{
  if (strcmp(str, right.getValue()) >= 0)
        return 1;
  else
        return 0;
}
```

MyString continues

جَامعة
المَنارة
MANARA UNIVERSITY

```
// Overloaded >= operator. Called when the operand on the right is a
// string. Returns 1 if right is greater than or equal to str.
int MyString::operator>=(const char *right)
{
   if (strcmp(str, right) >= 0)
          return 1;
   else
          return 0;
}
// Overloaded <= operator. Called when the operand on the right is a
// MyString object. Returns 1 if right.str is less than or equal to str.
int MyString::operator<=(MyString &right)
{
   if (strcmp(str, right.getValue()) <= 0)
          return 1;
   else
          return 0;
}
```



MyString continues

```
// Overloaded <= operator. Called when the operand on the right is a
// string. Returns 1 if right is less than or equal to str.
int MyString::operator<=(const char *right)
{
  if (strcmp(str, right) <= 0)
        return 1;
  else
        return 0;
}
// Overloaded stream insertion operator (<<).
ostream &operator<<(ostream &strm, MyString &obj)
{
  strm << obj.str;
  return strm;
}
```

MyString continues

جامعة المنارة
MANARA UNIVERSITY

```
// Overloaded stream extraction operator (>>).
istream &operator>>(istream &strm, MyString &obj)
{
  strm.getline(obj.str);
  strm.ignore();
  return strm;
}
```

Program 14-15

جَامعة
المَنارة
MANARA UNIVERSITY

```
// This program demonstrates the MyString class. Be sure to
// compile this program with mystring.cpp.
#include <iostream.h>
#include "mystring.h"

void main(void)
{
   MyString object1("This"), object2("is");
   MyString object3("a test.");
   MyString object4 = object1;  // Call copy constructor.
   MyString object5("is only a test.");
   char string1[] = "a test.";
   cout << "Object1: " << object1 << endl;
   cout << "Object2: " << object2 << endl;
   cout << "Object3: " << object3 << endl;
```

Program continues

جَامعة المَنارة

MANARA UNIVERSITY

```
  cout << "Object4: " << object4 << endl;
  cout << "Object5: " << object5 << endl;
  cout << "String1: " << string1 << endl;
  object1 += " ";
  object1 += object2;
  object1 += " ";
  object1 += object3;
  object1 += " ";
  object1 += object4;
  object1 += " ";
  object1 += object5;
  cout << "Object1: " << object1 << endl;
}
```

Program Output

جَامعة
المَنارة
MANARA UNIVERSITY

Object1: This
Object2: is
Object3: a test.
Object4: This
Object5: is only a test.
String1: a test.
Object1: This is a test. This is only a test.

Program 14-16

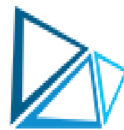
```
// This program demonstrates the MyString class. Be sure to
// compile this program with mystring.cpp.
#include <iostream.h>
#include "mystring.h"

void main(void)
{
   MyString name1("Billy"), name2("Sue");
   MyString name3("joe");
   MyString string1("ABC"), string2("DEF");

   cout << "Name1: " << name1.getValue() << endl;
   cout << "Name2: " << name2.getValue() << endl;
   cout << "Name3: " << name3.getValue() << endl;
   cout << "String1: " << string1.getValue() << endl;
   cout << "String2: " << string2.getValue() << endl;
```

Program continues

جَامعة
المَـنارة
MANARA UNIVERSITY

```
if (name1 == name2)
        cout << "Name1 is equal to Name2.\n";
else
        cout << "Name1 is not equal to Name2.\n";
if (name3 == "joe")
        cout << "Name3 is equal to joe.\n";
else
        cout << "Name3 is not equal to joe.\n";
if (string1 > string2)
        cout << "String1 is greater than String2.\n";
else
        cout << "String1 is not greater than String2.\n";
if (string1 < string2)
        cout << "String1 is less than String2.\n";
else
```

Program continues




```
        cout << "String1 is not less than String2.\n";
if (string1 >= string2)
        cout << "String1 is greater than or equal to "
            << "String2.\n";
else
        cout << "String1 is not greater than or equal to "
            << "String2.\n";
if (string1 >= "ABC")
        cout << "String1 is greater than or equal to "
            << "ABC.\n";
else
        cout << "String1 is not greater than or equal to "
            << "ABC.\n";
if (string1 <= string2)
        cout << "String1 is less than or equal to "
            << "String2.\n";
```

Program continues

```
else
        cout << "String1 is not less than or equal to "
            << "String2.\n";
if (string2 <= "DEF")
        cout << "String2 is less than or equal to "
            << "DEF.\n";
else
        cout << "String2 is not less than or equal to "
            << "DEF.\n";
}
```

Program Output

جَامعة
المَـنارة
MANARA UNIVERSITY

Name1: Billy
Name2: Sue
Name3: joe
String1: ABC
String2: DEF
Name1 is not equal to Name2.
Name3 is equal to joe.
String1 is not greater than String2.
String1 is less than String2.
String1 is not greater than or equal to String2.
String1 is greater than or equal to ABC.
String1 is less than or equal to String2.
String2 is less than or equal to DEF.

14.8 Object Composition

- Object composition occurs when a class contains an instance of another class.
- Creates a “has a” relationship between classes.