EE 441 – CH5

DYNAMIC MEMORY ALLOCATION

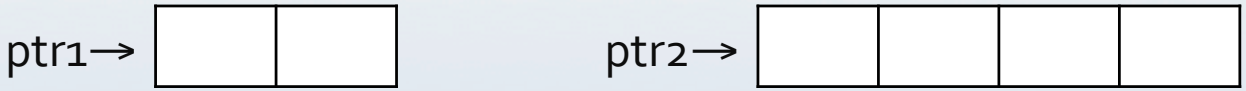
Instructor: UĞUR HALICI

MEMORY ALLOCATION OPERATOR NEW

```
T  *p; //declare p as a pointer
 p= new T // p is the address of Memory for data type T

int *ptr1; // size of int is 2
long *ptr2; // size of long in 4

ptr1= new int;
ptr2= new long;
```



by default, the content in memory have no initial value. If such a value is desired, it must be supplied as a parameter when the operator is used:

```
p=new T(value);

ptr2= new long(1000000)
```

DYNAMIC ARRAY ALLOCATION

p=new T [n]; // allocate an array of n items of type T

Example:

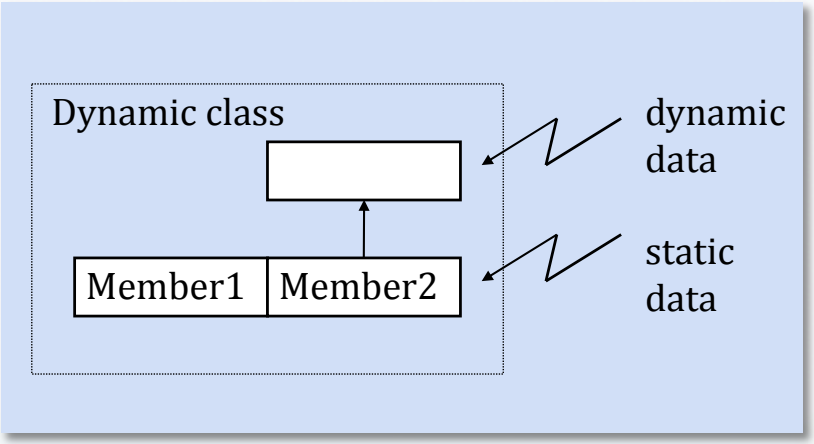
```
 long *p;
p=new long [50] // allocate an array of 50 long integers
if (p==NULL)
{
cerr<< "Memory allocation error!<< endl;
exit(1); // terminate the program
}
```

THE MEMORY DEALLOCATION OPERATOR DELETE

```
T *p, *q;      // p and q are pointers to type T
p=new T;       // points to a single item
q new T[n];  //points to an array of elements

delete p; // deallocates the pariable pointed by p
delete [ ] q; // deallocates the entire array pointed  by q
```

ALLOCATION OF OBJECT DATA



Example:

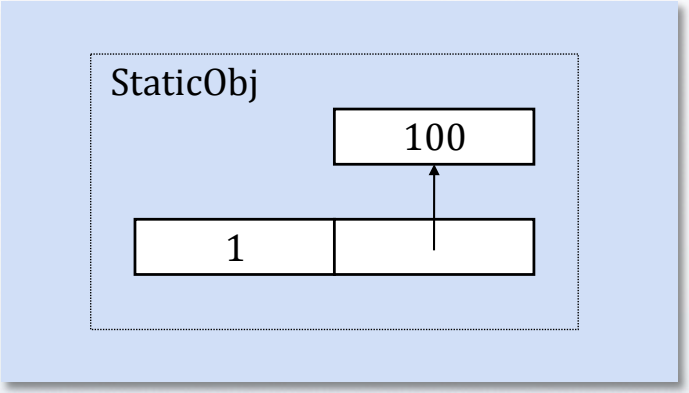
```
template <class T>
class DynamicClass
{ private:
T  member1;
T *member2
 // variable of type T and a pointer to data of type T

public:
//constructor with parameters to initialize member data
DynamicClass(const T  &m1, const  T &m2)
// copy constructor : create a copy of the input object
DynamicClass(const DynamicClass<T> & obj)
// some methods…
….
//assignment operator
DynamicClass<T> &operator=(cont DynamicClass<T> &rhs)
//destructor
~DynamicClass(void)
}
```

```
// class implementation
//constructor with parameters to initialize member data
Template  <classT>
DynamicClass<T>::DynamicClass(const T  &m1, const  T &m2)
{
// parameter m1 initializes static member
member1=m1;
//allocate dynamic memory and initialize it with value m2
member2=new T(m2)
cout << "Constructor:"<<member1<<'/'<<*member2<<endl;
}
```

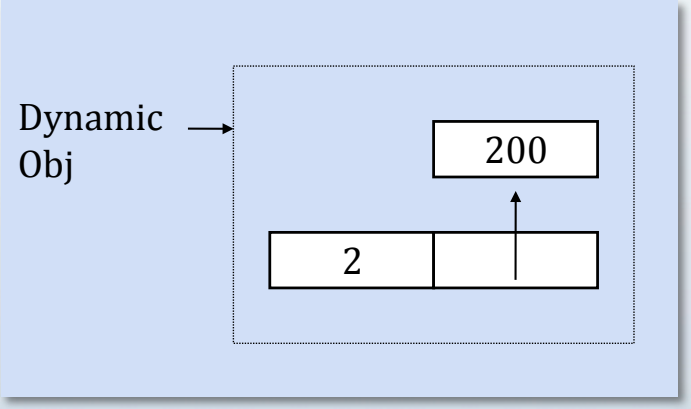
Example: The following statements define a static variable staticObj.The static Obj has parameters 1 and 100 that initialize the data members:

```
//Dynamic Class object
DynamicClass<int> staticObj(1,100)
```



In the following, the object dynamicObj is created by the new operator. Parameters 2 and 200 are supplied as parameters to the constructor:

```
//pointer variable
DynamicClass<int> *dynamicObj;
//allocate an object
dynamicObj=new dynamicClass<int>(2,200)
```



Running the program results in

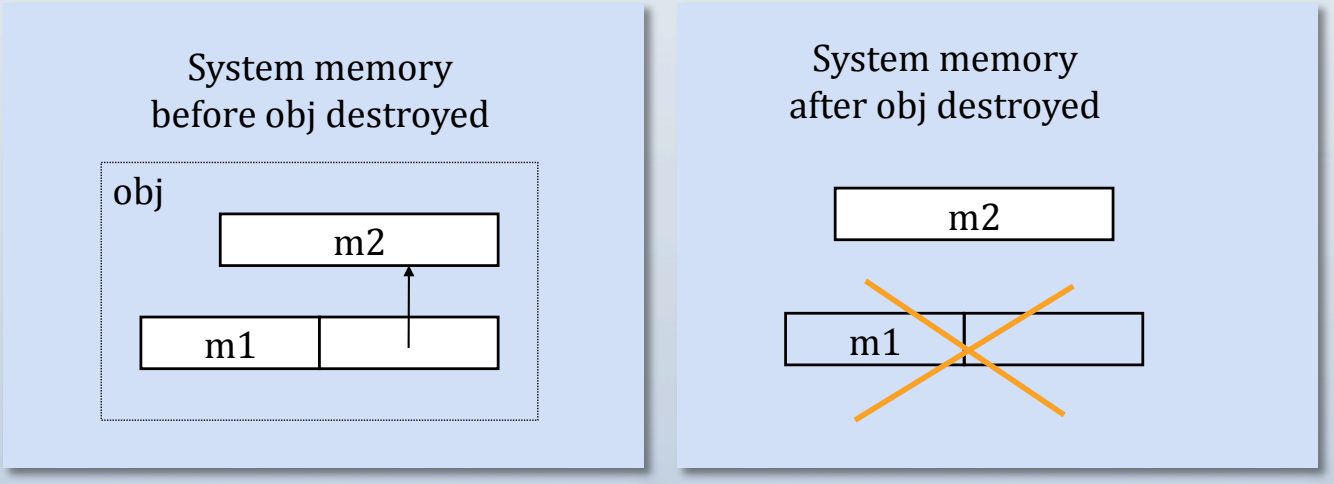
```
Constructor: 1/100
Constructor: 2/200
```

DEALLOCATION OBJECT DATA: THE DESTRUCTOR

Consider the function DestroyDemo that creates a DynamicClass object having integer data

```
void DestroyDemo(int m1,int m2)
{DynamicClass<int> obj(m1,m2);
}
```

Upon return from DestroyDemo, obj is destroyed; however the process does not deallocate the dynamic memory associated with the object:



For effective memory management, we need to deallocate the dynamic data within the object at the same time the object being destroyed.

The C++ language provides a member function, called the destructor, which is called automatically when an object is destroyed.

For DynamicClass, the destructor has the declaration:

```
~DynamicClass(void);
```

The character "~" represents "complement", so ~DynamicClass is the complement of the constructor DynamicClass.

A destructor never has a parameter or a return type. For our sample class, the destructor is responsible to deallocate the dynamic data for member2.

```
// destructor: deallocates memory allocated by the constructor
template <class T>
DynamicClass<T>:~DynamicClass(void);
{cout<<"Destructor:"<<member1"<<'/'<<*member2<<endl;
  delete member2;
}
```

The destructor is called whenever an object is deleted. When a program terminates, all global objects or objects declared in the main program are destroyed. For local objects created within a block, the destructor is called when the program exits the block.

Example

```
void DestroyDemo(int m1, int m2)
{DynamicClass<int> Obj(m1,m2)   <----------   Constructor for Obj(3,300)
}      <-------------------------------------   Destructor for Obj
void main(void)
{DynamicClass<int> Obj1(1,100), *Obj2;  <---   Constructor for Obj1(1,100)
  Obj2=new DynamicClass<int>(2,200);    <---   Constructor for *Obj2(2,200)
  DestroyDemo(3,300);
  delete Obj2;     <-------------------------   Destructor for Obj2
}      <-------------------------------------   Destructor for Obj1
```

running the program results in the output:

```
Constructor: 1/100
Constructor: 2/200
Constructor: 3/300
Destructor: 3/300
Destructor: 2/200
Destructor: 1/100
```

ASSIGNMENT AND INITIALIZATION

Assignment and initialization are basic operation that apply to any object.

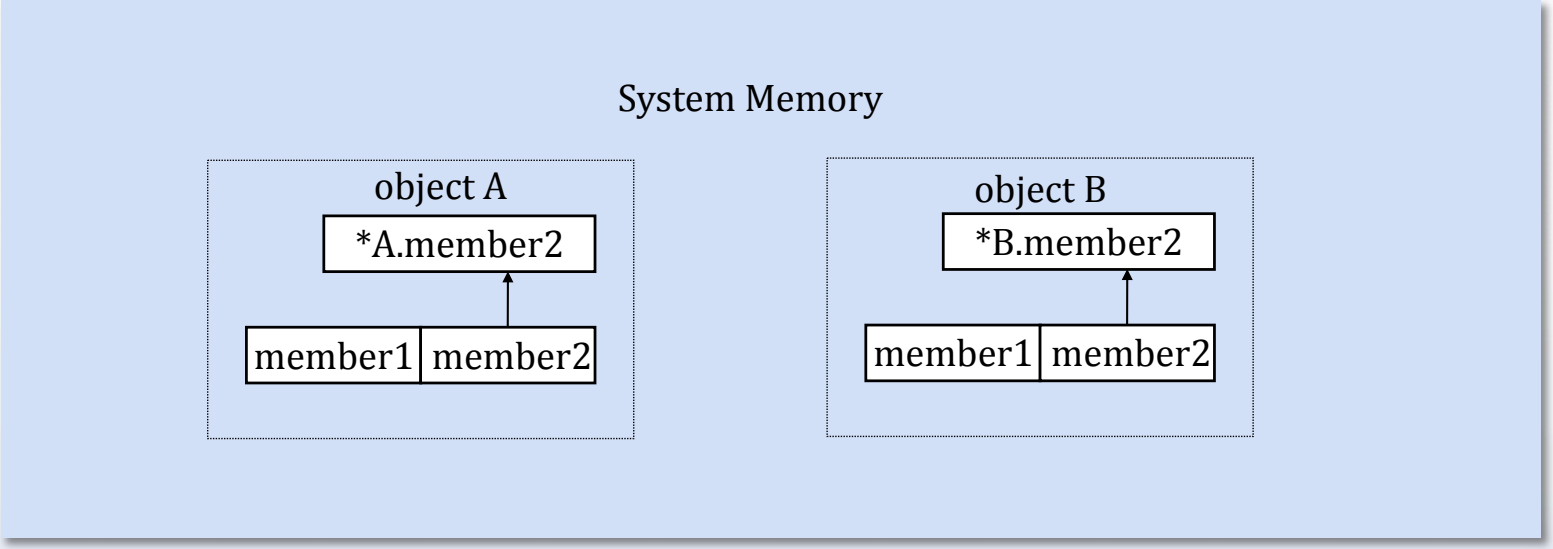
- The assignment Y=X causes a bitwise copy of the data from object X to the data in object Y.
- Initialization creates a new object that is a copy of another object. The operations are illustrated with objects X and Y.

```
// initialization
DynamicClass<int> X(20,50), Y=X;
//creates DynamicClass objects X and Y
// data in Y is initialized by data in X
```

```
// assignment
Y=X;
//data in Y is overwritten by data in X
```

Special consideration must be used with dynamic memory so that unintended errors are not created. We must create new methods that handle object assignment and initialization.

ASSIGNMENT ISSUES



The assignment statement of B=A causes the data in A to be copied to B

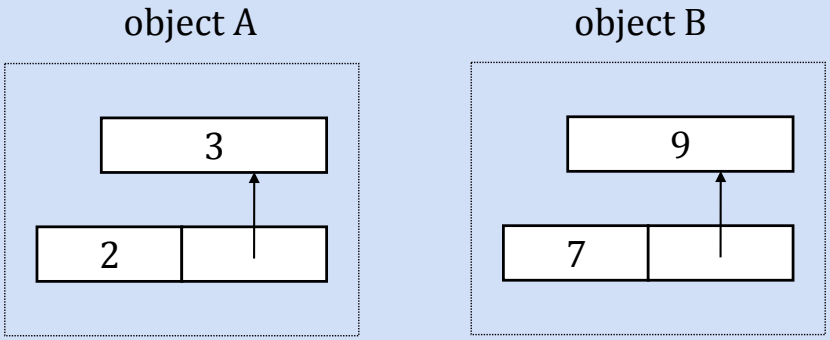
member1 of B=member1 of A //copies static data from A to B

member2 of B=member2 of A //copies pointer from A to B

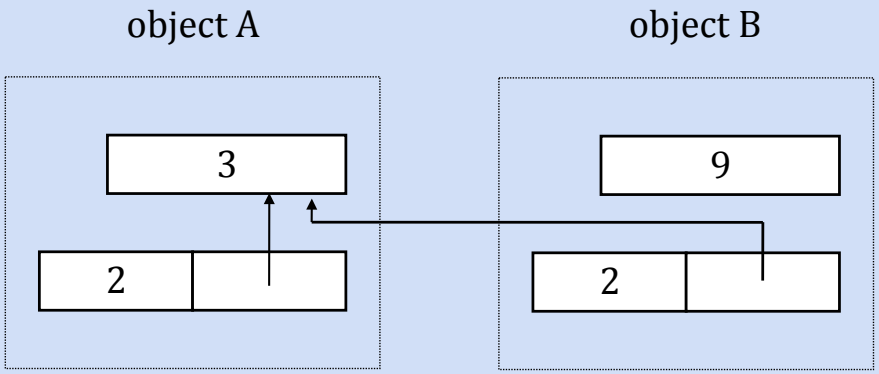
Example

```
void F(void)
{DynamicClass<int> A(2,3), B(7,9);
B=A
}
```

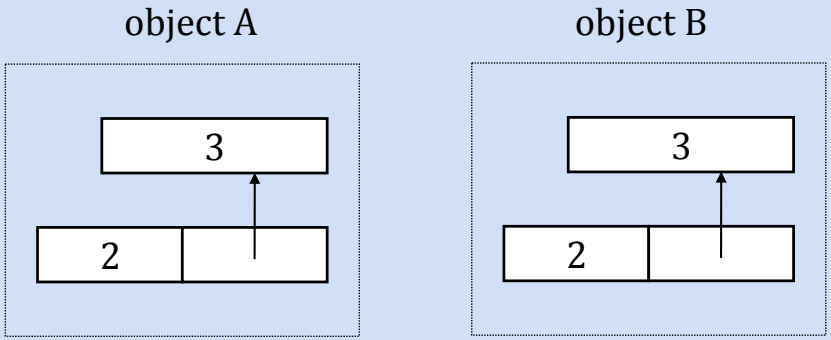
After execution of
DynamicClass<int> A(2,3), B(7,9);



After execution of B=A we have



although it was desired



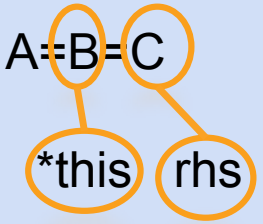
Solution is Overloading the assignment operator

```
// Overloaded assignment operator = returns a reference
// to the current object
template <class T>
DynamicClass<T>& operator= (const DynamicClass <T>& rhs)
{//copy static data member from rhs to the current object
member1=rhs.member1
// content of the dynamic memory must be same as that of rhs
*member2=*rhs.member2;
cout <<"Assignment Operator: "<<member1<<'/'<<*member2<<endl
return *this;
 //reserved word this is used to return a reference to the current object
}
void main (void)
{ DynamicClass <int> A(2,3), B(7,9);
B=A;
}
```

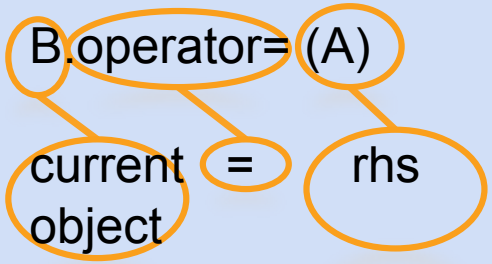
Solution is Overloading the assignment operator

```
// Overloaded assignment operator = returns a reference
// to the current object
template <class T>
DynamicClass<T>& operator= (const DynamicClass <T>& rhs)
{//copy static data member from rhs to the current object
member1=rhs.member1
// content of the dynamic memory must
*member2=*rhs.member2;
cout <<"Assignment Operator: "<<mem                    dl
return *this;
 //reserved word this is used to return a reference to the current object
}
void main (void)
{ DynamicClass <int> A(2,3), B(7,9);
B=A;
}
```

Necessary for assignments like:



here B=A is equivalent to



INITIALIZATION ISSUES

Object initialization is an operation that creates a new object that is a copy of another object. Like assignment, when the object has dynamic data, the operation requires a specific member function, called the copy constructor.

DynamicClass<int> A(3,5), B=A; //initialize object B with A

The declaration of A creates an object a whose initial data are member1=3 and *member2=5.

The declaration of B creates an object with two data members that are then structured to store the same data values found in A.

In addition to performing initialization when declaring (1) objects, initialization also occurs when passing an object as a value parameter (2) in a function. For instance, assume function F has a value parameter X of type DynamicClass<int>.

```
DynamicClass<int> F(DynamicClass<int> X) // value parameter
{DynamicClass<int> obj;
…..
return obj
}
```

When calling block uses object A as the actual parameter, the local object X is created by copying A:

```
DynamicClass<int> A(3,5), B(0,0);    //declare objects
B=F(A)                               //call F by copying A to X
```

When the return is made from F, a copy of obj is made by copy constructor (3), the destructor for the local object X and obj are called, and the copy of obj is returned as the value of the function

CREATING A COPY CONSTRUCTOR

In order to properly handle classes that allocate dynamic memory, C++ provides the copy constructor to allocate dynamic memory for the new object and initialize its data values

The copy constructor is a member function that is declared with the class name and a single parameter. Because it is a constructor, it does not have a return value

```
//copy constructor: initialize new object to have the same data as obj.
template <class T>
DynamicClass<T>:: DynamicClass(const DynamicClass<T>& obj)
{// copy static data member from obj to current object
member1=obj.member1;
//allocate dynamic memory and initialize it with value *obj.member2
member2=new T(*(obj.member2));
cout<<"Copy Constructor:"<<member1<<'/'<<member2<<endl;
}
```

If a class has a copy constructor, it is used by the compiler whenever it needs to perform initialization. The copy constructor is used only when an object is created

Despite their similarity, assignment and initialization are clearly different operations. Assignment is done when the object on the left-hand side already exists. In the case of initialization, a new object is created by copying data from an existing object.

The parameter in a copy constructor must be passed by reference.

Parameter passing by value may result in catastrophic effects:

```
DynamicClass(DynamicClass<T> X)
```

copy constructor

parameter pass by value calls copy constructor recursively

The problem is solved by using reference in parameter passing.

```
DynamicClass(const DynamicClass<T>& X)
```

In addition, the reference parameter X should be declared constant, since we certainly do not want to modify the object we are copying.

```
#include <iostream.h>
# include "dynamic.h"
template <class T>
DynamicClass<int> Demo(DynamicClass<T> one, DynamicClass& two, T m)
{ DynamicClass<T> obj(m,m);
  return obj;}
void main()
{ DynamicClass<int> A(3,5), B=A, C(0,0);
  C=Demo(A,B,5);}
```

Running the program results in

```
Constructor: 3/5                // construct A
Copy Constructor: 3/5           // construct B
Constructor: 0/0                // construct C
Copy Constructor: 3/5           // construct one
Constructor: 5/5                // construct obj
Copy Constructor: 5/5           // construct return object for Demo
Destructor: 5/5                 // destruct obj  upon return
Destructor: 3/5                 //  destruct one
Assignment Operator: 5/5        // assign return object of Demo to C
Destructor: 5/5                 // destruct return  object of demo
Destructor: 5/5                 // destruct C
Destructor: 3/5                 // destruct B
Destructor: 3/5                 // destruct A
```