Overloading =

The compiler will provide a *default assignment operator*, =, for every user defined class. This default assignment operator will make a *shallow copy* of the object, i.e., it will copy only the statically defined variables but not the dynamically defined ones.

This can lead to unexpected results and program crashes.

When creating a new class it is the author's responsibility to overload the assignment operator.

A good rule of thumb is that whenever it is necessary to provide a user defined copy constructor you should also provide a user defined =.

In the next slides we will see an example of a class that *should but doesn't* provide a user defined $=$. There will be an example of an unexpected result and a possible program crash.

```
//bad_thing3.h
//Header file for new thing class
//uses default assignment operator

class thing
{
public:
   thing(const thing& T); //copy constructor
   thing(int x_val, int y_val, int z_val );
   thing();
   ~thing();

   void set(int x_val, int y_val, int z_val);
   void display();

private:
   int* x;
   int  y,z;
};
```

```
//bad_thing3.cpp
//Implementation file for thing class
#include <iostream.h>
#include <stddef.h>  // for NULL
#include "bad_thing3.h"

thing::thing(const thing& T): y(T.y), z(T.z)
{ cout << "Copy Constructing new thing"<<endl;
  x = new int;
  *x = *(T.x);
}//end copy constructor

thing::thing(int x_val, int y_val, int z_val): y(y_val), z(z_val)
{ cout << "Constructing new thing with given values"<<endl;
  x = new int;
  *x = x_val;
}

thing::thing(): y(0), z(0)
{ cout << "Constructing new thing using default constructor" << endl;
  x = new int;
  *x = 0;
}

thing::~thing()
{ cout << "Deleting thing" << endl;
  delete x;
  x = NULL;}

void thing::set(int x_val, int y_val, int z_val)
{ *x = x_val; y = y_val; z = z_val;}

void thing::display()
{ cout << "(*x,y,z) = (";
  cout << *x <<",";
  cout << y  <<",";
  cout << z  <<")";}
```

```
//bad_Assign.cpp
//Same as Assign.cpp except for
//including "bad_thing3.h"
//Assignment vs. copy constructor

#include <iostream.h>
#include "bad_thing3.h"
void main()
{ thing T(1,2,3);
  cout << "T"; T.display(); cout << endl << endl;

  thing A=T;   // copy constructor called
  cout << "A"; A.display(); cout << endl << endl;

  thing B;
  cout << "B"; B.display(); cout << endl << endl;
  B = T; cout << "Assigning B =T"<< endl;
  cout << "B"; B.display(); cout << endl << endl;

  T.set(4,5,6);
  cout << "Setting T" << endl;
  cout << "T"; T.display(); cout << endl;
  cout << "B"; B.display(); cout << endl;
  cout << "A"; A.display(); cout << endl << endl;}
```

```
****************** OUTPUT **************
Constructing new thing with given values
T(*x,y,z) = (1,2,3)

Copy Constructing new thing
A(*x,y,z) = (1,2,3)

Constructing new thing using default constructor
B(*x,y,z) = (0,0,0)

Assigning B =T
B(*x,y,z) = (1,2,3)

Setting T
T(*x,y,z) = (4,5,6)
B(*x,y,z) = (4,2,3)
A(*x,y,z) = (1,2,3)

Deleting thing
Deleting thing
Deleting thing
```

```
//Really_bad_Assign.cpp
//Assignment vs. copy constructor

#include <iostream.h>
#include "bad_thing3.h"

void Do(thing &T)
{ thing newT;
  newT = T;
}

void main()
{ thing T(1,2,3);
  cout << "T"; T.display(); cout << endl << endl;

  cout << "Entering Do"<< endl;
  Do(T);
  cout << "Returned from Do" << endl << endl;

  T.set(4,5,6);  //PROBABLE PROGRAM CRASH. Why?
  cout << "T"; T.display(); cout << endl << endl;
}
```

To fix these problems we need to provide a user defined assignment operator, =.

- C++ requires that the assignment operaor be a member function.
- `thing A,B; A=B`
  is translated by compiler into
  `A.operator=(B)`
  Assignment operator has a calling member and only *one* parameter, which is of the same type as the calling parameter.
- Even though they may both use the = symbol the assignment operator is different from the copy constructor.

```
thing A(2);
thing B=A;   /Use of copy constructor

thing C;
thing C = A; //Use of assignment operator
```

```
//thing3.h

class thing
{public:
   thing(const thing& T); //copy constructor
   thing(int x_val, int y_val, int z_val );
   thing();
   ~thing();

   // Member Function!
   void  operator=(const thing& T);

   void  set(int x_val, int y_val, int z_val);
   int   get_x();
   int   get_y();
   void  display();

 private:
   int* x;
   int  y,z;
};
```

```
//thing3.cpp
//Implementation file for thing class
#include <iostream.h>
#include <stddef.h>  // for NULL
#include "thing3.h"

thing::thing(const thing& T): y(T.y), z(T.z)
{ cout << "Copy Constructing new thing"<<endl;
  x = new int;
  *x = *(T.x);
}//end copy constructor

thing::thing(int x_val, int y_val, int z_val): y(y_val), z(z_val)
{ cout << "Constructing new thing with given values"<<endl;
  x = new int;
  *x = x_val;
}

thing::thing(): y(0), z(0)
{ cout << "Constructing new thing using default constructor" << endl;
  x = new int;
  *x = 0;
}

thing::~thing()
{ cout << "Deleting thing with val = " << *x << endl;
  delete x;
  x = NULL;
}

void thing::set(int x_val, int y_val, int z_val)
{ *x = x_val; y = y_val; z = z_val;}

void thing::display()
{ cout << "(*x,y,z) = ("; cout << *x <<",";
  cout << y  <<","; cout << z  <<")";
}
```

```
void thing::operator=(const thing& T)
{ cout << "Assigning"<<endl;
  x = new int;
  *x = *(T.x);
  y = T.y;
  z = T.z;
}
```

```
//Assign.cpp
//same as bad_Assign.cpp
//Assignment vs. copy constructor
#include <iostream.h>
#include "thing3.h"

void main()
{ thing T(1,2,3);
  cout << "T"; T.display(); cout << endl << endl;

  thing A=T;   // copy constructor called
  cout << "A"; A.display(); cout << endl << endl;

  thing B;
  cout << "B"; B.display(); cout << endl << endl;
  B = T; cout << "Assigning B =T"<< endl;
  cout << "B"; B.display(); cout << endl << endl;

  T.set(4,5,6);
  cout << "Setting T" << endl;
  cout << "T"; T.display(); cout << endl;
  cout << "B"; B.display(); cout << endl;
  cout << "A"; A.display(); cout << endl << endl;
}
```

```
*************OUTPUT********************
Constructing new thing with given values
T(*x,y,z) = (1,2,3)

Copy Constructing new thing
A(*x,y,z) = (1,2,3)

Constructing new thing using default constructor
B(*x,y,z) = (0,0,0)

Assigning
Assigning B =T
B(*x,y,z) = (1,2,3)

Setting T
T(*x,y,z) = (4,5,6)
B(*x,y,z) = (1,2,3)
A(*x,y,z) = (1,2,3)

Deleting thing with val = 1
Deleting thing with val = 1
Deleting thing with val = 4
```

The overloaded assignment operator we provided works but can be improved.

- What happens if we set
  `thing A; A=A;`
- What happens if we set
  `thing A,B,C; A=B=C;`

These problems need to be fixed. We will see later how to write a new, improved, overloaded assignment operator.