Lecture 11.1

Topics

1. Object Pointer – **this**
2. An Example – Bad One Again

____

1. Object Pointer -- this

C++ provides a special pointer called **this** for class method members. **this** is a pointer that is automatically passed to any member method when it is called, and it is a pointer to the object that generates the call. Only member methods of the class from which an object created can be passed by **this** pointer.

Since **this** pointer is pointing to the current object, a pointer notation can be used to refer to other members in class that creates the object. In addition, **this** is a keyword so that no other valid qualifiers (i.e., variables, parameters, ...) can have the same name.

Normally, **this** pointer is not used in pointer operations or notations because the shorthand notation without **this** pointer would work just fine. However, **this** pointer is heavily used when operator overloading is forming; operator overloading will be presented in the next lecture.

Example 1

```
//Program Name:  cis25L1111.cpp
//Discussion:    this Pointer
#include <iostream>
using namespace std;

class OA {
  int x;
public:
  OA() {
    x = 0;
  }
  OA( int a ) {
    x = a;
  }
  ~OA() {
    cout << "\nOne Destruction!\n";
  }
  void setX( int value ) {
    x = value;
    return;
  }
  int getX( void ) {
    return x;
  }
};

class Assignment {
  int x;
public:
  Assignment() {
    this->x = 0;
  }
  Assignment( int a ) {
    this->x = a;
  }
  ~Assignment() {
    cout << "\nOne Destruction!\n";
  }
  void setX( int value ) {
    this->x = value;
    return;
  }
  int getX( void ) {
    return this->x;
  }
};

int main( void ) {
  OA oaObj1( 10 );
  cout << "\nValue of x in object oaObj1 is "
       << oaObj1.getX();

  Assignment aObj2( 20 );
  cout << "\nValue of x in object aObj2 is "
       << aObj2.getX();

  return 0;
}
```

OUTPUT

```
Value of x in object oaObj1 is 10
Value of x in object aObj2 is 20
One Destruction!

One Destruction!
```

In above example, two classes have similar structures except for the use of **`this`** pointer. As soon as method `getX()` from class `Assignment` is called through `oaObj2,` the **`this`** pointer is automatically passed to it.

2. An Example – Bad One Again

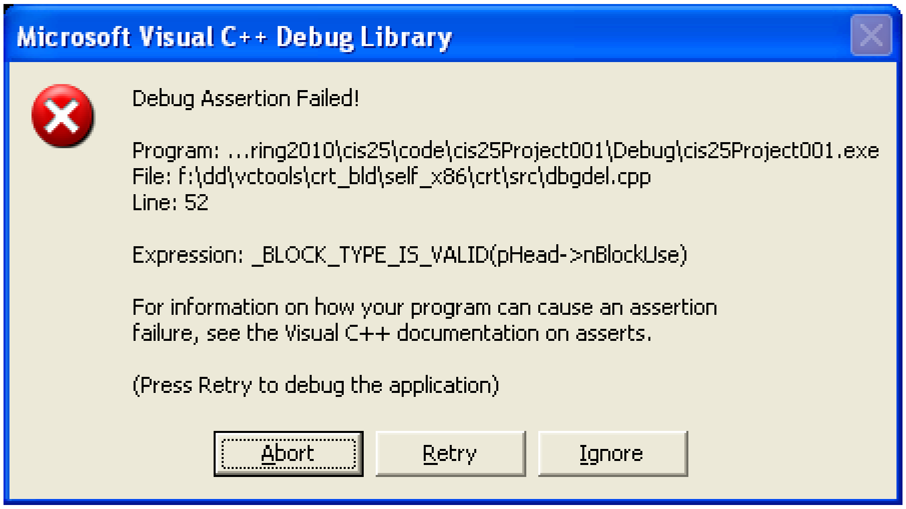
What is going on here?

Example 2

```
//Program Name: cis25L1112.cpp
//Discussion:   Class & Object –
//                Dynamic Memory Allocation & Functions
#include <iostream>
using namespace std;

class OA {
  int* x;
public:
  OA( void ) {
    x = new int;
    *x = 0;
  }

  ~OA() {
    delete x;
    cout << "\nOne Destruction!" << endl;
  }

  void setX( int value ) {
    *x = value;
    return;
  }

  int getX( void ) {
  return *x;
  }
};

int square( OA old ) {
  return ( old.getX() * old.getX() );
}

int main( void) {
  OA oa1;
  oa1.setX( 10 );     // Assigning to member x in one object

  cout << "Value of x in object oa1 is " << oa1.getX();
  cout << "\nSquare of x in oa1 is " << square( oa1 );

  return 0;
}
```

OUTPUT

```
Value of x in object oa1 is 10
One Destruction!

Square of x in oa1 is 100
```



Microsoft Visual C++ Debug Library

Debug Assertion Failed!

Program: ...ring2010\cis25\code\cis25Project001\Debug\cis25Project001.exe
File: f:\dd\vctools\crt_bld\self_x86\crt\src\dbgdel.cpp
Line: 52

Expression: _BLOCK_TYPE_IS_VALID(pHead->nBlockUse)

For information on how your program can cause an assertion failure, see the Visual C++ documentation on asserts.

(Press Retry to debug the application)

This example presents a problem of destroying allocated memory unwarrantedly. The dynamic memory in **oa1** is no longer available after square(**oa1**) is ended. This left the memory value of x in **oa1** useless and may be harmful to the system. The entire system may be hung up thereafter

One way to correct the above situation is to pass an address (or a reference) of the object to the function and not a copy of the object. Here, no new object would be created and the execution may be satisfactory.

Let’s consider a pass-by-reference approach – Discussion will be given in class.