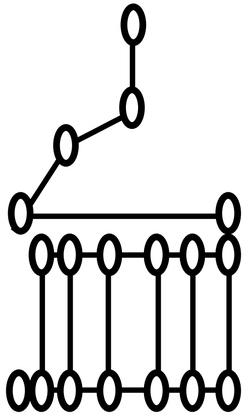
Lecture: Destructor basics

ENGR 2730: Computers in Engineering



Constructor/destructor motivation

Constructor/destructor motivation

How are member variables initialized when objects are instantiated (created)?

```
SomeClass myObject;
```

What happens when objects are destroyed?

Constructor/destructor motivation

How are member variables initialized when objects are instantiated (created)?

```
SomeClass myObject;
```

We will want to specify what happens when an object is created with special member functions call constructors.

- Does memory need to be allocated?
- Does validation need to be done?
- Do I register this object with central server?
- Open a log file on disk?

What happens when objects are destroyed?

If we need to "clean-up" we will want to specify what happens when objects are destroyed with a destructor.

- Does memory need to be de-allocated?
- De-register from central server?
- Close log file on disk?

Destructor basics

Destructor basics

*"The **destructor** of a class is a member function that is automatically activated when an object becomes inaccessible.*
The destructor has no arguments and its name must be the character '~' followed by the class name.

The destructor is called automatically when an object is destroyed.

The primary responsibility of most destructors is releasing resources: dynamic memory, open files, network connections, etc…"

from Data Structures and Other Objects Using C++

Destructor basics

*"The **destructor** of a class is a member function that is automatically activated when an object becomes inaccessible.*

The destruct *haracter ~*

Because the de *nake explicit*

The compiler will automatically provide an empty destructor if you do not define one. Thus, unless you need to release dynamically allocated memory, you often do not need to explicitly define one.

The primary responsibility of most destructors is releasing resources: dynamic memory, open files, network connections, etc…"

from Data Structures and Other Objects Using C++

Destructor Example

```
class MyClass {
public:
    MyClass(size_t number = 1); // constructor
    ~MyClass(); // destructor
private:
    float * m_Memory;
    size_t  m_size; // size_t is a positive int type
};
MyClass::MyClass(size_t number)
{
    m_Memory = new float[number];  // Memory Allocated
    m_size = number;
    cout << "MyClass object with size = " << m_size << " created. " << endl;
}
MyClass::~MyClass()
{
    delete [] m_Memory; // Memory De-allocated
    cout << "MyClass object with size = " << m_size << " destroyed. " << endl;
}
```

```
int main()
{
    MyClass object0;
    MyClass object1(2);
    MyClass object2(3);
}
```

Output:

```
MyClass object with size = 1 created.
MyClass object with size = 2 created.
MyClass object with size = 3 created.
MyClass object with size = 3 destroyed.
MyClass object with size = 2 destroyed.
MyClass object with size = 1 destroyed.
```

CQ: What will be printed as a result of calling the following function (assume MyClass defined as on prior slides)?

```
int main()
{
    MyClass object(5);
    myFunction(object);
    return 0;
}

void myFunction(MyClass &obj)
{
    cout << "In function." << endl;
}
```

A.

```
MyClass object with size = 5 created.
MyClass object with size = 5 created.
In function.
MyClass object with size = 5 destroyed.
MyClass object with size = 5 destroyed.
```

B.

```
MyClass object with size = 5 created.
In function.
MyClass object with size = 5 destroyed.
MyClass object with size = 5 destroyed.
```

C.

```
MyClass object with size = 5 created.
In function.
MyClass object with size = 5 destroyed.
```

D.

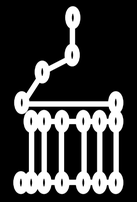
```
MyClass object with size = 5 created.
MyClass object with size = 0 created.
In function.
MyClass object with size = 0 destroyed.
MyClass object with size = 5 destroyed.
```

Example Class: Dynamic Memory Allocation and Destructor

```
class MyString {
public:
    MyString( char *text );  // Declare constructor
    ~MyString();             //  and destructor.

    friend ostream &operator<<(ostream &os, const MyString &string1);

private:
    char  * m_text = nullptr;
    size_t  m_sizeOfText = 0;
};
```



Example Class: Dynamic Memory Allocation and Destructor

```
class MyString {
public:
    MyString( char *text );  // Declare constructor
    ~MyString();             //  and destructor.

    friend ostream &operator<<(ostream &os

private:
    char  * m_text = nullptr;
    size_t  m_sizeOfText = 0;
};
```

```
// Define the constructor.
MyString::MyString( char * text ) {

    // Compute number of characters in the input text buffer
    char * tmpPtr = text;
    m_sizeOfText = 0;
    while (*tmpPtr != '\0'){
        m_sizeOfText++;
        tmpPtr++;
    }

    // Dynamically allocate the correct amount of memory.
    m_text = new char[ m_sizeOfText + 1 ];

    // If the allocation succeeds, copy the initialization string.
    if( m_text != nullptr ){
        for (int i = 0; i <= m_sizeOfText; i++){
            m_text[i] = text[i];
        }
    }
}
```

Example Class: Dynamic Memory Allocation and Destructor

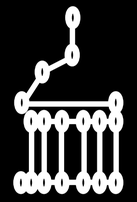
```
class MyString {
public:
    MyString( char *text );  // Declare constructor
    ~MyString();             //  and destructor.

    friend ostream &operator<<(ostream &os, const MyString &string1);

private:
    char  * m_text = nullptr;
    size_t  m_sizeOfText = 0;
};
```

```
// Define the destructor.
MyString::~MyString() {
    // Deallocate the memory reserved for string.
    if( m_text != nullptr ){
        delete[] m_text;
        m_text != nullptr;
    }
}

// Overload the stream output operator to print string
ostream &operator<<(ostream &os, const MyString &string1) {
    os << string1.m_text;
    return os;
}
```



Example Class: Dynamic Memory Allocation and Destructor

```
class MyString {
public:
    MyString( char *text );  // Declare constructor
    ~MyString();             //  and destructor.

    friend ostream &operator<<(ostream &os, const MyString &string1);

private:
    char  * m_text = nullptr;
    size_t  m_sizeOfText = 0;
};
```

```
int main() {
    char buffer[] = "Example of a class that uses dynamic
memory allocation and a destructor.";

    MyString str(buffer);

    cout << str << endl;

    return 0;
}
```

OUTPUT -> Example of a class that uses dynamic memory allocation and a destructor.