DYNAMIC MEMORY
THE BIG FOUR

Problem Solving with Computers-II

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

Read the syllabus. Know what's required. Know how to get help.

Learning Goals (Last Week)

- Review basics of classes
  - Defining classes and declaring objects
  - Access specifiers: private, public
  - Different ways of initializing objects and when to use each:
    - Default constructor
    - Parametrized constructor
    - Parameterized constructor with default values
    - Initializer lists

Learning Goals (today)

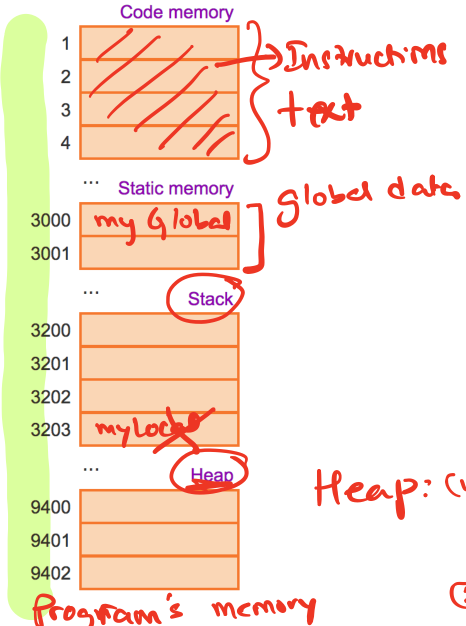
- Develop a mental model of how programs are represented in memory.
- Identify situations when data needs to be created on the heap vs. stack
- Identify the big four and when you need to implement these vs. use the default versions provided by C++

C++ Program's Memory Regions

$ g++ program.cpp
$ ./a.out

program.cpp g++ → a.out Executable.

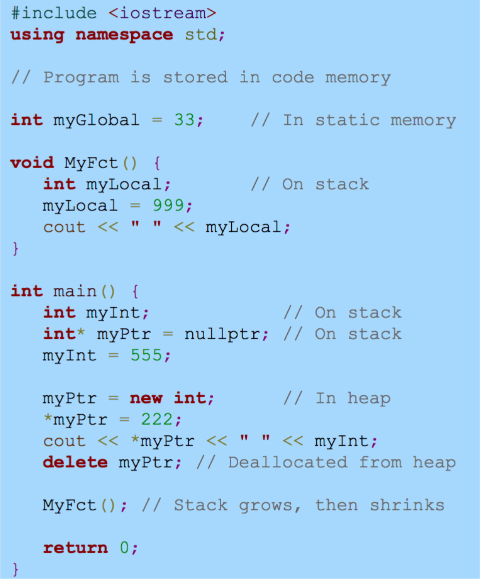
```
#include <iostream>
using namespace std;

// Program is stored in code memory

int myGlobal = 33;      // In static memory

void MyFct() {
   int myLocal;         // On stack
   myLocal = 999;
   cout << " " << myLocal;
}

int main() {
   int myInt;              // On stack
   int* myPtr = nullptr;   // On stack
   myInt = 555;

   myPtr = new int;        // In heap
   *myPtr = 222;
   cout << *myPtr << " " << myInt;
   delete myPtr; // Deallocated from heap

   MyFct(); // Stack grows, then shrinks

   return 0;
}
```



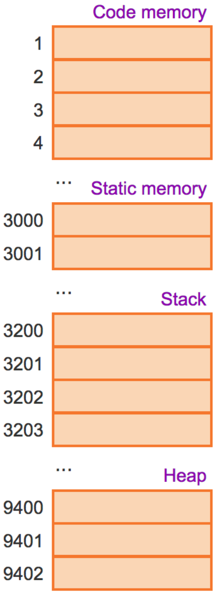
Heap: (1) Region of program memory.
(2) Managed by the programmer

The code regions store program instructions. myGlobal is a global variable and is stored in the static memory region. Code and static regions last for the entire program execution.

C++ Program’s Memory Regions



```
#include <iostream>
using namespace std;

// Program is stored in code memory

int myGlobal = 33;     // In static memory

void MyFct() {
   int myLocal;         // On stack
   myLocal = 999;
   cout << " " << myLocal;
}

int main() {
   int myInt;              // On stack
   int* myPtr = nullptr; // On stack
   myInt = 555;

   myPtr = new int;        // In heap
   *myPtr = 222;
   cout << *myPtr << " " << myInt;
   delete myPtr; // Deallocated from heap

   MyFct(); // Stack grows, then shrinks

   return 0;
}
```



- Stack: Segment of memory managed automatically using a Last in First Out (LIFO) principle.
- Heap: Segment of memory managed by the programmer
  - Data created on the heap stays there
  - FOREVER or
  - until the programmer explicitly deletes it

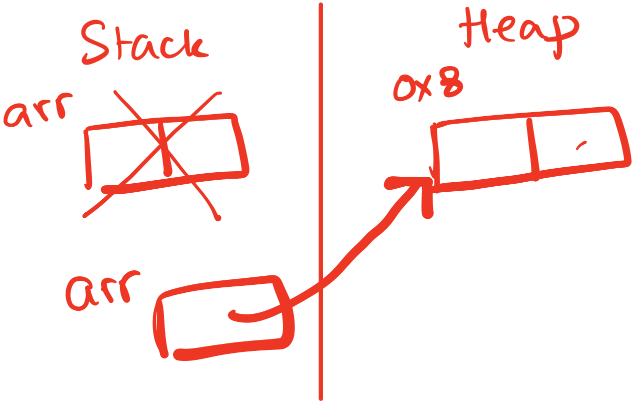
The code regions store program instructions. myGlobal is a global variable and is stored in the static memory region. Code and static regions last for the entire program execution.

Heap vs. stack

```
#include <iostream>
using namespace std;
int [ ]
int* createAnIntArray(int len){

    int arr[len];
    return arr;

}
```

on the stack

```
int *arr = new int [len];
return arr;
```



Does the above function correctly return an array of integers?

A. Yes

[✓ B. No]

The Big Four

1. Constructor
2. Destructor
3. Copy Constructor
4. Copy Assignment

Constructor and Destructor

complex c{4,5};

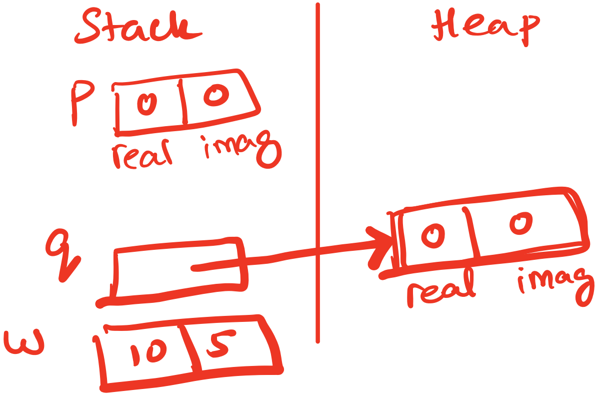
Every class has the following special methods:

- Constructor: Called right AFTER new objects are created in memory

- Destructor: Called right BEFORE an object is deleted from memory

The compiler automatically generates default versions, but you can override them

```
void foo(){
     complex p;
     complex* q = new complex;
     complex w{10, 5};
}
```



How many times is the constructor called above?

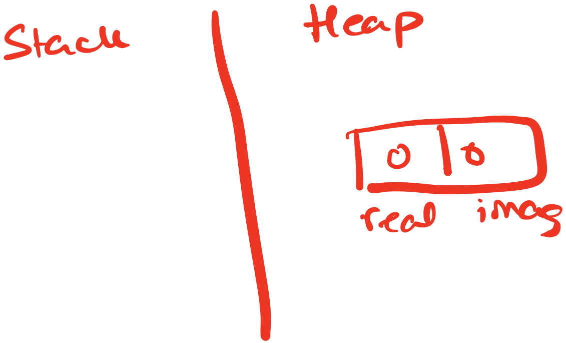
A. Never

B. Once

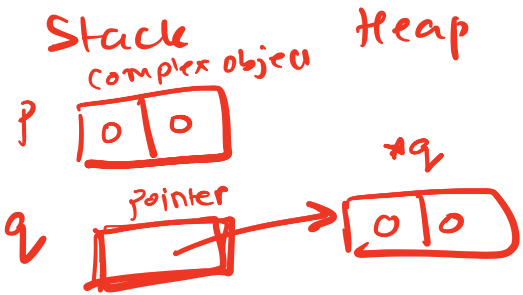
C. Two times

[✓ D. Three times]

foo has a Memory leak



```
void foo(){
    complex p;
    complex *q = new complex;
}
```



The destructor of which of the objects is called after foo() returns?

[✓ A.] p

p is the only complex object on the stack.

B. q

C. *q

D. None of the above

✓Copy constructor

- Creates a new object and initializes it using an existing object

Complex c1 ;

Using c1

0x8

4	5

:

1) complex c2 {c1} ;

Existing object to initialize the new object c2 copy constructor is called,

2) complex c2 (c1) ;

3) complex c2 = c1 ;

0xa

c2

4	5

In which of the following cases is the copy constructor called?

A.
```
complex p1;
complex p2{1, 2};
```

B.
```
complex p1{1, 2};
complex p2{p1};
```

C.
```
complex *p1 = new complex{1, 2};
complex p2 = *p1;
```

D. B&C

E. A, B & C

Copy assignment

- Default behavior: Copies the member variables of one object into another

```
complex p1{1, 2}; // Parametrized constructor
complex p2;
p2 = p1; // Copy assignment function is called
```

```
double foo(complex p){
   return p.magnitude();
}
int main(){
    complex q{1, 2};
    foo(q);
    }
```

Which of the following special methods is called as a result of calling foo?

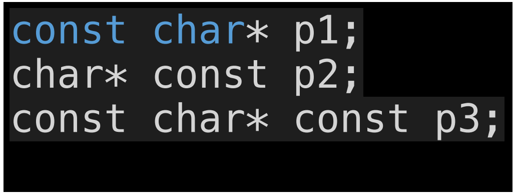
A. Parameterized constructor

B. Copy constructor

C. Copy Assignment

D. Destructor

Constant pointers and pointers to constants



```
const char* p1;
char* const p2;
const char* const p3;
```

Operator Overloading

We would like to be able to compare two objects of the class using the following operators

==

!=

and possibly others

```
bool operator==(const complex & c1, const complex &c2){
    return c1.real==c2.real && c1.imag == c2.imag;

}
```

Summary

- Classes have member variables and member functions (method). An object is a variable where the data type is a class.
- You should know how to declare a new class type, how to implement its member functions, how to use the class type.
- Frequently, the member functions of an class type place information in the member variables, or use information that's already in the member variables.
- New functionality may be added using non-member functions, friend functions, and operator overloading (next lectures)

Next time

- Linked Lists and the rule of three