CS-2001 Data Structures

Fall'21

Week # 01

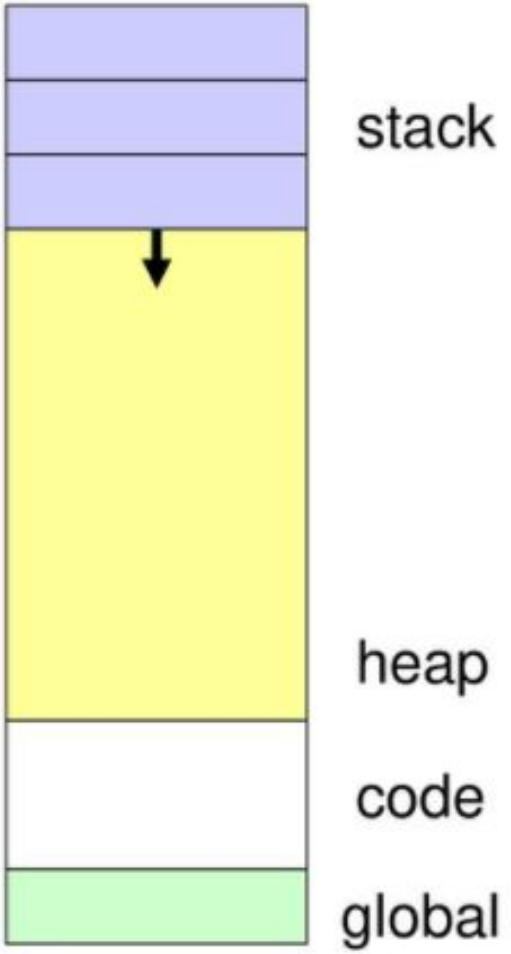
Lecture 03

Agenda

- Dynamic memory management
- constructors, destructors
- copy constructor and assignment operator
- their usage and issues
- function pointers in classes

Overview of Memory Layout in C++

- 4 major memory segments
  - Global: variables outside stack, heap
  - Code (a.k.a. text): the compiled program
  - Heap: dynamically allocated variables
  - Stack: parameters, automatic and temporary variables
- Key differences from Java
  - Destructors of automatic variables called when stack frame where declared pops
  - No garbage collection: program must explicitly free dynamic memory
- Heap and stack use varies dynamically
- Code and global use is fixed
- Code segment is “read-only”



Dynamic Memory Management

- Refers to performing memory allocation manually by programmer.
- Dynamically allocated memory is allocated on **Heap** and non-static and local variables get memory allocated on **Stack.**

- **Why ?**
- We are free to allocate and deallocate memory whenever we need and whenever we don't need anymore. There are many cases where this flexibility helps. Examples of such cases are Linked List, Tree, etc.

- **How?**
- Malloc()/Calloc()
- New/delete

Normal Vs Dynamically allocated memory

- For normal variables like “int a”, “char str[10]”, etc, memory is automatically allocated and deallocated.
- For dynamically allocated memory like “int *p = new int[10]”, it is programmers responsibility to deallocate memory when no longer needed.
- **Memory Leak:**
- Memory leak occurs when programmers create a memory in heap and forget to delete it.
- it reduces the performance of the computer by reducing the amount of available memory.

Malloc()/calloc()

- library functions that allocate memory dynamically on runtime.
- a pointer to the block of memory is returned otherwise **NULL** value is returned which indicates the failure of allocation.
- malloc() doesn't initialize the allocated memory while calloc() allocates the memory and also initializes the allocated memory block to zero.

- void* malloc(size_t size);
- void* calloc(size_t num, size_t size);

New/ delete

- C++ supports malloc/calloc functions and also has two operators **new** and **delete** that perform the task of allocating and freeing the memory in a better and easier way.
- new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

int *p = new int; (declaring)

int *p = new int(25); (initializing)

float *q = new float(75.25);

int *p = new int[10] (allocation of array)

- normal arrays are deallocated by compiler,dynamically allocated arrays always remain there until either they are deallocated by programmer or program terminates

delete p;

delete[] p; (freeing array)

Delete vs free

- delete operator should only be used either for the pointers pointing to the memory allocated using new operator or for a NULL pointer, and free() should only be used either for the pointers pointing to the memory allocated using malloc() or for a NULL pointer.
- The most important reason why free() should not be used for de-allocating memory allocated using NEW is that, it does not call the destructor of that object while delete operator does.

Lets see some coding examples.

- New/ delete: newdlt.cpp
- Dynamic Memory Allocation for Arrays: DMAA.cpp
- Dynamic Memory Allocation for Objects:DMAO.cpp

Constructors

- They should be declared in the public section
- They do not have any return type, not even void
- They get automatically invoked when the objects are created
- They cannot be inherited though derived class can call the base class constructor
- Like other functions, they can have default arguments
- You cannot refer to their address
- Constructors cannot be virtual

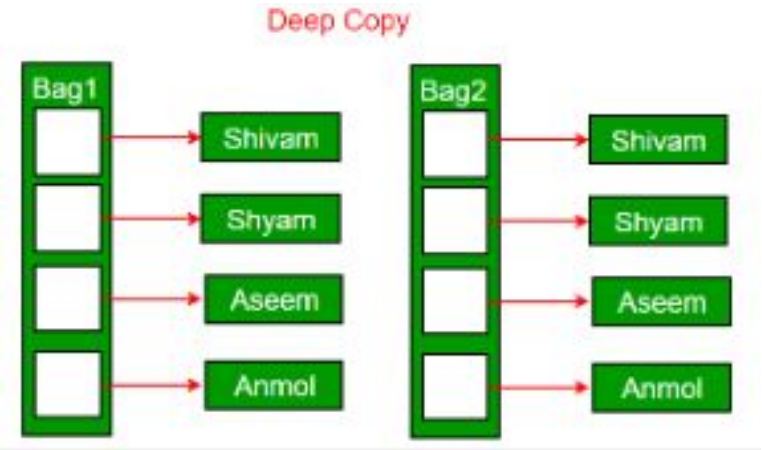
Types of constructors

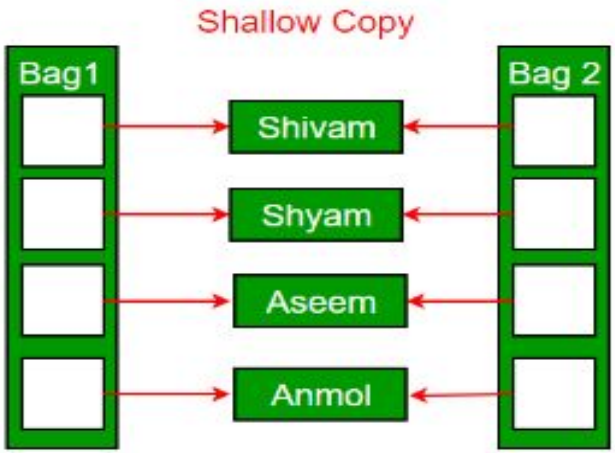
- **Default constructor** is the constructor which doesn't take any argument. It has no parameters.
- **Parameterized constructor:** It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created.
- **Copy constructor:** A copy constructor is a member function which initializes an object using another object of the same class.

Copy constructor vs assignment operator

- Copy constructor is called when a new object is created from an existing object, as a copy of the existing object.
- Assignment operator is called when an already initialized object is assigned a new value from another existing object.
- Refer to typesofconstructors.cpp

Deep copy vs shallow copy





Copy Constructor	Assignment Operator
The Copy constructor is basically an overloaded constructor	Assignment operator is basically an operator.
This initializes the new object with an already existing object	This assigns the value of one object to another object both of which are already exists.
Copy constructor is used when a new object is created with some existing object	This operator is used when we want to assign existing object to new object.
Both the objects uses separate memory locations.	One memory location is used but different reference variables are pointing to the same location.
If no copy constructor is defined in the class, the compiler provides one.	If the assignment operator is not overloaded then bitwise copy will be made

Destructor:

- Destructor is a member function which destructs or deletes an object.
- Destructor function is automatically invoked when the objects are destroyed.
- It cannot be declared static or const.
- The destructor does not have arguments.
- It has no return type not even void.
- A destructor should be declared in the public section of the class.
- The programmer cannot access the address of destructor.
- There can only one destructor in a class with class name preceded by ~, no parameters and no return type.
- The main reason of a destructor is to wipe of all the data members initialize by a constructor and the object life cycle.

Initializer list

- Initializer List is used in initializing the data members of a class. The list of members to be initialized is indicated with constructor as a comma-separated list followed by a colon.

```
Point(int i = 0, int j = 0):x(i), y(j) {}
    constructor can be written as:
        Point(int i = 0, int j = 0) {
            x = i;
            y = j;
        }
```

Rule of three

- This rule basically states that **if a class defines one (or more) of the following, it should explicitly define all three:**
  - destructor
  - copy constructor
  - copy assignment operator

  The default constructors and assignment operators do shallow copy and we create our own constructor and assignment operators when we need to perform a deep copy (For example when a class contains pointers pointing to dynamically allocated resources).

Example

- Now, suppose our class does not have a copy constructor. Copying an object will copy all of its data members to the target object. In this case when the object is destroyed the destructor runs twice. Also the destructor has the same information for each object being destroyed. In the absence of an appropriately defined copy constructor, the destructor is executed twice when it should only execute once. This duplicate execution is a source for trouble.
- Refer to ruleofthree.cpp

Function pointers

POINTERS TO FUNCTIONS

- A pointer to the function, it is very handy for a lot of situations.
- Function Pointers can only hold compatible functions

•return_type (* function (parameters)

```
// pointer to functions
#include <iostream>
using namespace std;

int addition (int a, int b)
{ return (a+b); }

int subtraction (int a, int b)
{ return (a-b); }

int operation (int x, int y, int
(*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
  return (g);
}

int main ()
{
  int m,n;
  int (*minus)(int,int) = subtraction;

  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  cout <<n;
  return 0;
}
```