Object Oriented Programming using C++

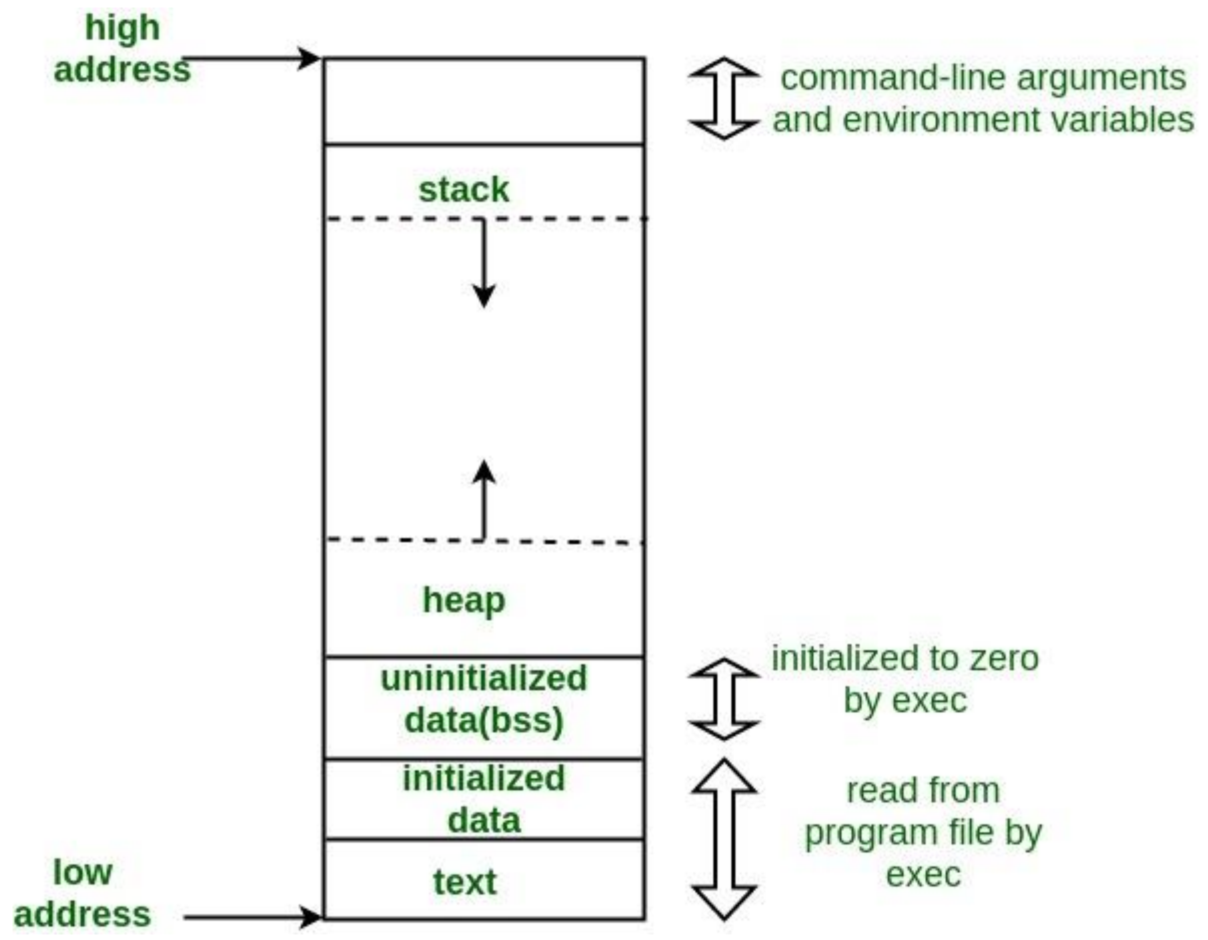
CSE-III
CSE206-OOP

Dr. Priyanka Singh

Dynamic memory allocation

- Dynamic memory allocation in C++ 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
- One use of dynamically allocated memory is to allocate memory of variable size which is not possible with compiler allocated memory except variable length arrays.
- The most important use is flexibility provided to programmers. 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.

A typical memory layout of a running process


high address
stack
heap
uninitialized data(bss)
initialized data
text
low address
command-line arguments and environment variables
initialized to zero by exec
read from program file by exec

Introduction

- Although C uses malloc() and calloc() function to allocate memory dynamically at run time and uses free() function to free dynamically allocated memory.

- C++ supports these functions and in addition has two operators new and delete that perform the task of allocating and freeing the memory in a better and easier way.

malloc and calloc()

- The malloc() and calloc() are library functions that allocate memory dynamically. It means that memory is allocated during runtime (execution of the program) from the heap segment.
- **Initialization: malloc()** allocates memory block of given size (in bytes) and returns a pointer to the beginning of the block. ***malloc() doesn't initialize the allocated memory.*** If we try to access the content of memory block (before initializing) then we'll get segmentation fault error (or maybe garbage values).

```
void* malloc(size_t size);
```

- **calloc()** allocates the memory and also initializes the allocated memory block to zero. If we try to access the content of these blocks then we'll get 0.

```
void* calloc(size_t num, size_t size);
```

malloc and calloc()

- **Number of arguments:** Unlike malloc(), calloc() takes two arguments:

1) Number of blocks to be allocated.
2) Size of each block.

- **Return Value:** After successful allocation in malloc() and calloc(), a pointer to the block of memory is returned otherwise NULL value is returned which indicates the failure of allocation.

Program to demonstrate the use of calloc() and malloc()

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int* arr;
    // malloc() allocate the memory for 5 integers containing garbage values
    arr = (int*)malloc(5 * sizeof(int)); // 5*4bytes = 20 bytes
    // Deallocates memory previously allocated by malloc() function
    free(arr);
    // calloc() allocate the memory for 5 integers and set 0 to all of them
    arr = (int*)calloc(5, sizeof(int));
    // Deallocates memory previously allocated by calloc() function
    free(arr);
    return (0);
}
```

Note:

- It is be better to use malloc over calloc, unless we want the zero-initialization because malloc is faster than calloc.
- So if we just want to copy some stuff or do something that doesn’t require filling of the blocks with zeros, then malloc would be a better choice.

new operator

- The new operator denotes a request for memory allocation on the Free Store. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

Syntax: `pointer-variable = new data-type;`

- Here, pointer-variable is the pointer of type data-type. Data-type could be any built-in data type including array or any user defined data types including structure and class. **Example:**

```
// Pointer initialized with NULL // Then request memory for the
variable
int *p = NULL;
p = new int;
            OR
// Combine declaration of pointer and their assignment
int *p = new int;
```

Memory allocation:

- **Initialize memory:** `pointer-variable = new data-type(value);`
- `Example:`

```
int *p = new int(25);
float *q = new float(75.25);
```

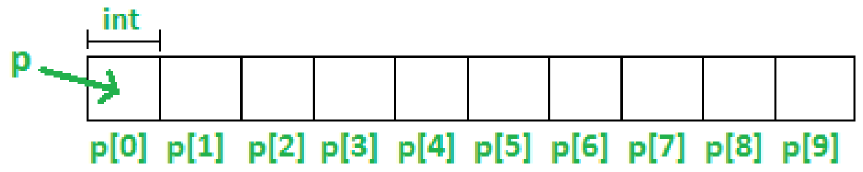
- **Allocate block of memory:** new operator is also used to allocate a block(an array) of memory of type data-type.

```
pointer-variable = new data-type[size];
```

where size (a variable) specifies the number of elements in an array.

Example: `int *p = new int[10]`

- It dynamically allocates memory for 10 integers continuously of type int and returns pointer to the first element of the sequence, which is assigned to p(a pointer). p[0] refers to first element, p[1] refers to second element and so on.



Normal Array Declaration vs Using new

- There is a difference between declaring a normal array and allocating a block of memory using new.
- The most important difference is, normal arrays are deallocated by compiler (If array is local, then deallocated when function returns or completes).
- However, dynamically allocated arrays always remain there until either they are deallocated by programmer or program terminates.

What if enough memory is not available during runtime?

- If enough memory is not available in the heap to allocate, the new request indicates failure by throwing an exception of type std::bad_alloc, unless “nothrow” is used with the new operator, in which case it returns a NULL pointer. Therefore, it may be good idea to check for the pointer variable produced by new before using it program. i.e.

```
int *p = new(nothrow) int;
if (!p)
{
   cout << "Memory allocation failed\n";
}
```

What if enough memory is not available during runtime?

- 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. If programmer doesn’t deallocate memory, it causes memory leak (memory is not deallocated until program terminates).
- Since it is programmer’s responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

Syntax: `delete pointer-variable;`

- Release memory pointed by pointer-variable: Here, pointer-variable is the pointer that points to the data object created by new.

Example:

```
delete p;
delete q;
```

delete

- To free the dynamically allocated array pointed by pointer-variable, use following form of delete:

```
// Release block of memory  pointed by pointer-variable
delete[] pointer-variable;
```

Example:

```
    delete[] p;

// It will free the entire array pointed by p.
```

// C++ program to illustrate dynamic allocation and deallocation of memory using new and delete

```
#include <iostream>
using namespace std;
int main ()
{
        // Pointer initialization to null
        int* p = NULL;
        // Request memory for the variable using new operator
        p = new(nothrow) int;
        if (!p)
                cout << "allocation of memory failed\n";
        else
        {       // Store value at allocated address
                *p = 29;
                cout << "Value of p: " << *p << endl;
        }
// Request block of memory using new operator
        float *r = new float(75.25);
        cout << "Value of r: " << *r << endl;
        // Request block of memory of size n
        int n = 5;
        int *q = new(nothrow) int[n];
        if (!q)
                cout << "allocation of memory failed\n";
        else
        {
                for (int i = 0; i < n; i++)
                        q[i] = i+1;
                cout << "Value store in block of memory: ";
                for (int i = 0; i < n; i++)
                        cout << q[i] << " ";
        }
        // freed the allocated memory
        delete p;
        delete r;
        // free the block of allocated memory
        delete[] q;
        return 0;
}
```

delete and free() in C++

- In C++, 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.

delete and free() in C++

```
#include<stdio.h>
#include<stdlib.h>
int main()
{       int x;
        int *ptr1 = &x;
        int *ptr2 = (int *)malloc(sizeof(int));
        int *ptr3 = new int;
        int *ptr4 = NULL;
        /* delete Should NOT be used like below because x is allocated on stack frame */
        delete ptr1;
        /* delete Should NOT be used like below because x is allocated using malloc() */
        delete ptr2;
        /* Correct uses of delete */
        delete ptr3;
        delete ptr4;
        getchar();
        return 0;
}
```

Exercise

- Write a program in C to find the largest element using Dynamic Memory Allocation.