SDF II(15B11CI211)

EVEN Semester 2021

2nd Semester , First Year

Jaypee Institute Of Information Technology (JIIT), Noida

Lecture 3 – Dynamic Memory Allocation in C++ using Pointers

Memory Allocation

- ✔ Memory allocation in programming is very important for storing values when you assign them to variables
- ✔ allocates memory for the variables declared by a programmer via the compiler
- ✔ allocation is done either before or at the time of program execution

Ways for memory allocation

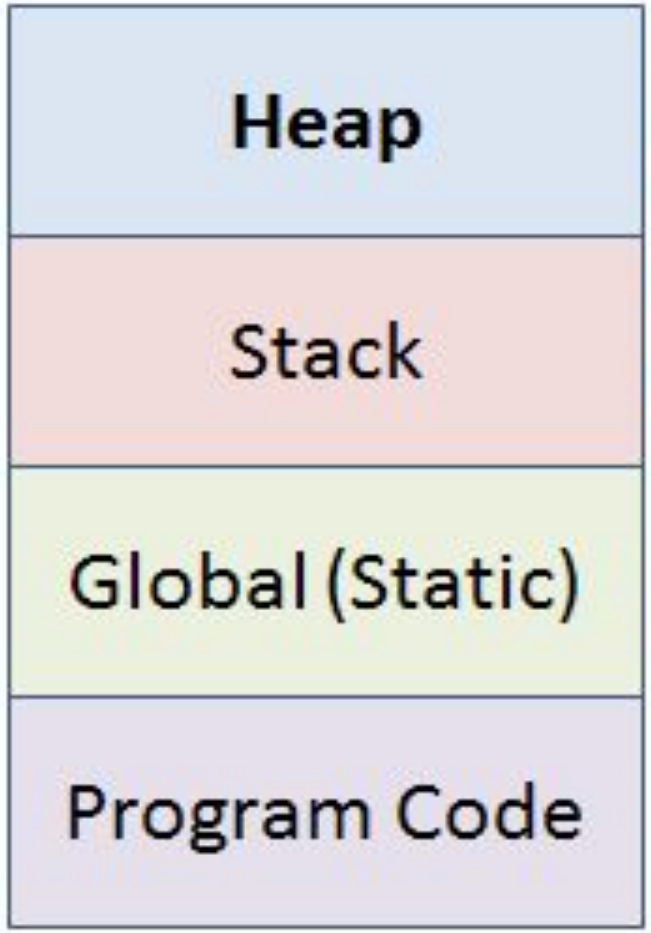
- **Compile time allocation or static allocation of memory:**
  - ✔ The memory for named variables is allocated by the compiler.
  - ✔ The memory allocated by the compiler is allocated on the stack.
  - ✔ Exact size and storage must be known at compile time.
  - ✔ for array declaration, the size has to be constant.
- **Runtime allocation or dynamic allocation of memory:**
  - ✔ The memory is allocated at runtime.
  - ✔ The memory space is allocated dynamically within the program run.
  - ✔ Dynamically allocated memory is allocated on the heap.
  - ✔ The exact space or number of the data items does not have to be known by the compiler in advance.
  - ✔ Pointers play a major role for dynamic memory allocation.

Dynamic Memory Allocation

- Memory de-allocation is also an important part of this concept
- The "clean-up" of storage space is done for variables or for other data storage.
- It is the job of the programmer to de-allocate dynamically created space.
- For de-allocating dynamic memory, **delete** operator is used.
- In other words, dynamic memory Allocation refers to performing memory management for dynamic memory allocation manually.

Memory Parts in C++ Program

- **stack:**
  - ✔ All variables declared inside any function takes up memory from the stack.
- **heap:**
  - ✔ It is the unused memory of the program and can be used to dynamically allocate the memory at runtime.



Source: https://study.com/academy/lesson/how-to-allocate-deallocate-memory-in-c-programming.html

The “new” Operator

- To allocate space dynamically, unary operator *new is used* , followed by the type being allocated for memory.

- ✔ new int; //dynamically allocates memory for an integer type
- ✔ new double; // dynamically allocates memory an double type
- ✔ new int[30]; // dynamically allocates memory for integer array

Allocating space for new

- int * ptr=NULL; // declares a pointer ptr
- ptr = new int; // dynamically allocate an int for loading the address in ptr
- double * i; // declares a pointer i
- i = new double; // dynamically allocate a double and loading the address in i

In the above example,

✔ we have declared a pointer variable ‘ptr’ to integer and initialized it to null.

✔ Using the “new” operator memory will be allocate to the “ptr” variable.

Dynamic Memory Allocation for Arrays

- To allocate memory for an array of characters, i.e., a string of 50 characters. Using that same syntax, memory can be allocated dynamically
- char* str = NULL; // Pointer initialized with NULL
- str= new char[50]; // Dynamic Allocation will be done

- Here, new operator allocates 50 continuous elements of type characters to the pointer variable str and returns the pointer to the first element of str.

Dynamic Memory Allocation for Linked List

Let us take a structure of a linked list node:

```
struct node
{
 int data;
node *next;
 };
node *temp=new  node; // dynamic memory allocation
```

- ✔ Memory is allocated required for a node by the **new** operator.
- ✔ ‘temp’ points to a node (or space allocated for the node).

Example: Linked List using new operator


```
#include <iostream>
using namespace std;
struct node
{
   int data;
   node *next;
};
class linked_list
{
private:
   node *head,*tail;
public:
   linked_list()
   {
      head = NULL;
      tail = NULL;
   }

   void add_node(int n)
   {
      node *tmp = new node;
      tmp->data = n;
      tmp->next = NULL;
       if(head == NULL)
              {
                     head = tmp;
                     tail = tmp;
              }
              else
              {
                 tail->next = tmp;
                 tail = tail->next;
              }
      }
};
int main()
{
   linked_list a;
   a.add_node(1);
   a.add_node(2);
   return 0;
}
```

Source: https://www.codesdope.com/blog/article/c-linked-lists-in-c-singly-linked-list/

The Delete Operator

- The memory allocated dynamically using the new operator has to be freed explicitly by the programmer. For this purpose, the “delete” operator is used
- **The general syntax of the delete operator is:**

delete pointer_variable;

- **So memory allocated to the ptr variable can be freed as:**

delete ptr;

- This statement frees the memory allocated to the variable “ptr” back to the memory pool.
- delete operator can also be used to free the memory allocated to arrays.

The Delete Operator

- **the memory allocated to the array str above can be freed as follows:**

delete[] str;

- Note the subscript operator used with the delete operator.
- This is because, as we have allocated the array of elements, we need to free all the locations.
- **Instead, if the following statement had executed:**

delete str;

- the above statement will only delete the first element of the array.
- Using subscript "[]" all the memory allocated is to be freed.

Malloc()

- The malloc() function from C, also exists in C++,
- But it is recommended to avoid using malloc() function.
- malloc() allocates requested size of bytes and returns a pointer to the first byte of allocated space.
- The main benefit of new over malloc() is that new doesn't just allocate memory, it constructs objects which is a prime concept of C++.

Dynamic memory allocation Programming Example:1

```
#include <iostream>
using namespace std;
 int main()
{
int* i= NULL;
i= new int;
*i= 5;
cout << "Value is : " << *i<< endl;
 delete i;
}
```

Example 2 : new and delete operators in C++.

```
#include <iostream>
#include <string>
using namespace std;
  int main()
 {
   int *p= NULL;
   p= new int();
   int *a= new int(10);

   if(!p)
   {
        cout<<"bad memory allocation"<<endl;
    }
   else
  {
       cout<<"memory allocated successfully"<<endl;
       *p= 5;
       cout<<"*p= "<<*p<<endl;
       cout<<"*a= "<<*a<<endl;
}
```

```
double *arr= NULL;
arr= new double[10];

if(!arr)
 {cout<<"memory not allocated"<<endl;}
else
   {
     for(int i=0;i<5;i++)
       arr[i] = i+1;
       cout<<“Array values : ";
     for(int i=0;i<5;i++)
       cout<<arr[i]<<"\t";
   }
delete p;
delete a;
delete[] arr;

return 0;
```

Program Output

```
memory allocated successfully
*i=  5
*a=  10
Array values:1 2      3      4      5
```

Dynamically Allocating Arrays:

The major use of the concept of dynamic memory allocation is for allocating memory to an array when we have to declare it by specifying its size but are not sure about it.

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

int main()

{
int len, sum = 0;
cout << "Enter the no. of students in the class" << endl;

 cin >> len;

int *marks = new int[len];                    //Dynamic memory allocation

cout << "Enter the marks of each student" << endl;

}
```

```
for( int i = 0; i < len; i++ )
        {
                cin >> *(marks+i);
         }

for( int i = 0; i < len; i++ )
        {
                sum += *(marks+i);
        }

cout << "sum is " << sum << endl;

return 0;
```

Explanation:

In this example first we ask the user for the number of students in a class and we store its value in the len variable.

Then we declare an array of integer and allocate it space in memory dynamically equal to the value stored in the len variable using this statement
int *marks = new int[length];

thus it is allocated a space equal to ‘length * (size of 1 integer)’.

The rest of the code is self-explanatory.

- There is a substantial difference between declaring a normal array and allocating dynamic memory for a block of memory using new.
- The most important difference is that the size of a regular array needs to be a *constant expression*, and thus its size has to be determined at the moment of designing the program, before it is run, whereas the dynamic memory allocation performed by new allows to assign memory during runtime using any variable value as size.
- The dynamic memory requested by our program is allocated by the system from the memory heap.
- However, computer memory is a limited resource, and it can be exhausted.
- Therefore, there are no guarantees that all requests to allocate memory using operator new are going to be granted by the system.

Dynamic Memory Allocation for Objects

- We can also dynamically allocate objects.
- As we know that Constructor a special class member function used to initialize an object and Destructor is also a class member function which is called whenever the object goes out of scope.
- Destructor is can be used to release the memory assigned to the object. It is called in the following conditions.
- When a local object goes out of scope
- For a global object, when an operator is applied to a pointer to the object of the class
- We can again use pointers while dynamically allocating memory to objects.

Let's see an example of an array of objects.

```
#include <iostream>
using namespace std;
class Random
{
public:
Random() {
cout << "Constructor" << endl;
}
~Random() {
cout << "Destructor" << endl;
}
};
int main()
{
Random* a = new Random[3];
delete [] a; // Delete array
return 0;
}
```

```
Constructor
Constructor
Constructor
Destructor
Destructor
Destructor
[Finished in 2.6s]
```

Explanation:

The Constructor will be called three times since we are allocating memory to three objects of the class Random.

The Destructor will also be called three times during each of these objects.

'Random* a = new Random[3];'

This statement is responsible for dynamic memory allocation of our object.

The major differences between static and dynamic memory allocations

Static Memory Allocation	Dynamic Memory Allocation
Memory Allocates variables permanently	Memory Allocates variables only if program unit gets active
Memory Allocation is done before program execution	Memory Allocation is done during program execution
Use stack data structure for implementation	Use heap data structure for implementation
Less efficient	More efficient
no Memory reusable	memory reusable and can be freed when not required

https://www.w3schools.in/difference-in-static-and-dynamic-memory-allocation/

References

- Robert Lafore, Object Oriented Programming in C++, SAMS, 2002
- https://www.softwaretestinghelp.com/new-delete-operators-in-cpp/
- https://www.softwaretestinghelp.com/stack-in-cpp/
- https://www.geeksforgeeks.org/
- https://www.tutorialspoint.com/abstract-data-type-in-data-structures