Data Structure Lab

Lab # 01

Dynamic Memory Allocation

Instructor: Muhammad Saad Khan

saad.khan@nu.edu.pk

Course Code: CL2001

Department of Computer Science,

National University of Computer and Emerging Sciences FAST Peshawar Campus

C++ Dynamic Memory

A good understanding of how dynamic memory really works in C++ is essential to becoming a good C++ programmer. Memory in your C++ program is divided into two parts −

- **The stack** − All variables declared inside the function will take up memory from the stack.
- **The heap** − This is unused memory of the program and can be used to allocate the memory dynamically when program runs.

Many times, you are not aware in advance how much memory you will need to store particular information in a defined variable and the size of required memory can be determined at run time.

You can allocate memory at run time within the heap for the variable of a given type using a special operator in C++ which returns the address of the space allocated. This operator is called **new** operator.

If you are not in need of dynamically allocated memory anymore, you can use **delete** operator, which de-allocates memory that was previously allocated by new operator.

new and delete operators in C++ for dynamic memory.

Dynamic memory allocation in C/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 (Refer Memory Layout C Programs for details).
What are applications?

- 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.

How is it different from memory allocated to normal variables?
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).

Dynamic Variable:

Syntax:

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

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;
```

```
// C++ program to demonstrate how to create dynamic variable
// using new
#include <iostream>
using namespace std;

int main()
```

```
{
    // pointer to store the address returned by the new
    int* ptr;
    // allocating memory for integer
    ptr = new int;

    // assigning value using dereference operator
    *ptr = 4;

    // printing value and address
    cout << "Address: " << ptr << endl;
    cout << "Value: " << *ptr;

   delete ptr;



    return 0;
}
```

Dynamic Array:

A dynamic array is quite similar to a regular array, but its size is modifiable during program runtime. Dynamic Array elements occupy a contiguous block of memory.

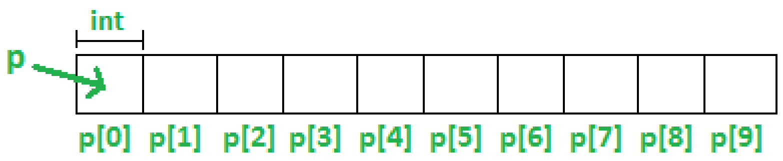
Syntax:

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

Example:

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

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



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

int main() {
   int size;

   // Ask the user for the size of the array
   cout << "Enter the size of the array: ";
   cin >> size;

   // Dynamically allocate memory for the 1D array
   int* arr = new int[size];

   // Insert elements into the array using pointer arithmetic
   cout << "Enter " << size << " elements: " << endl;
   for (int i = 0; i < size; i++) {
      cin >> *(arr + i);  // Inserting elements using pointers
   }

   // Display the array elements using pointer arithmetic
   cout << "Array elements are: ";
   for (int i = 0; i < size; i++) {
      cout << *(arr + i) << " ";  // Accessing elements using pointers
   }
   cout << endl;
   // Deallocate the memory
   delete[] arr;

   return 0;
}
```

Delete Operator:

Since it is the programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator in C++ language.

Syntax:

```
// Release memory pointed by pointer-variable
delete pointer-variable;
```

Dynamic Memory Allocation for Objects :

Objects are no different from simple data types. For example, consider the following code where

we are going to use an array of objects to clarify the concept −

```
#include <iostream>
using namespace std;
class Box
{
public:
Box()
{
cout << "Constructor called!" << endl;
}
~Box()
{
cout << "Destructor called!" << endl;
}
};
int main()
{
Box* myBoxArray = new Box[4];
delete[] myBoxArray; // Delete array
return 0;
}
```

If you were to allocate an array of four Box objects, the Simple constructor would be called four times and similarly while deleting these objects, destructor will also be called same number of times. If we compile and run above code, this would produce the following result −

```
Constructor called!
Constructor called!
Constructor called!
Constructor called!
Destructor called!
Destructor called!
Destructor called!
Destructor called!
```