Memory Management in C++

Definition

- Dynamic memory allocation in C/C++ refers to performing memory allocation manually by a programmer. Dynamically allocated memory is allocated on **Heap,** and non-static and local variables get memory allocated on **Stack.**
- C++ allows us to allocate the memory of a variable or an array in run time. This is known as dynamic memory allocation.
- In C++, we need to deallocate the dynamically allocated memory manually after we have no use for the variable.

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 for variable-length arrays.
- The most important use is the flexibility provided to programmers. We are free to allocate and deallocate memory whenever we need it and whenever we don't need it anymore. There are many cases where this flexibility helps. Examples of such cases are Linked List, Tree, etc.

There are two operators we use for this purpose –

1. new operator
2. delete operator

The new operator allocates memory to a variable. For example,

```
int* pointVar;
pointVar = new int;
*pointVar = 45;
```

In the above code, the new operator returns the address of the variable's memory location.

But in the case of an array, the new operator returns the address of the first element of the array.

delete Operator

- Once we no longer need to use a variable that we have declared dynamically, we can deallocate the memory occupied by the variable.
- For this, the delete operator is used. It returns the memory to the operating system. This is known as memory deallocation.

The syntax for delete operator is –

```
delete pointerVariable;
```

Example

```
int* pointVar;
pointVar = new int;
*pointVar = 45;
cout << *pointVar;
delete pointVar;
```

Example

```
#include <iostream>
 using namespace std;
 int main() {
 // declare an int pointer
 int* pointInt;
// declare a float pointer
 float* pointFloat;
// dynamically allocate memory
 pointInt = new int;
pointFloat = new float;
 // assigning value to the memory
 *pointInt = 45;
*pointFloat = 45.45f;
cout << *pointInt << endl;
cout << *pointFloat << endl;
 // deallocate the memory
delete pointInt;
delete pointFloat;
return 0;
}
```

Example

```
#include <iostream>
 using namespace std;
 int main() {
 int num;
cout << "Enter total number of students: ";
 cin >> num;
 float* ptr;
 ptr = new float[num];
 cout << "Enter GPA of students." << endl;
 for (int i = 0; i < num; ++i) {
 cout << "Student" << i + 1 << ": ";
 cin >> *(ptr + i);
 }
cout << "\n Displaying GPA of students." << endl;
 for (int i = 0; i < num; ++i) {
 cout << "Student" << i + 1 << ": " << *(ptr + i) << endl;
 }
delete[] ptr;
return 0;
}
```

C++ String Class

- C++ string class internally uses char array to store character but all memory management, allocation, and null termination is handled by string class itself that is why it is easy to use.
- The length of the C++ string can be changed at runtime because of dynamic allocation of memory similar to vectors.
- As string class is a container class, we can iterate over all its characters using an iterator similar to other containers like vector, set and maps, but generally, we use a simple for loop for iterating over the characters and index them using the [] operator.
  C++ string class has a lot of functions to handle string easily. Most useful of them are demonstrated in below code.

Example

```
int main()
{
   // various constructor of string class

   // initialization by raw string
   string str1("first string");

   // initialization by another string
   string str2(str1);

   // initialization by character with number of occurrence
   string str3(5, '#');

   // initialization by part of another string
   string str4(str1, 6, 6); //    from 6th index (second parameter)
                        // 6 characters (third parameter)

   // initialization by part of another string : iterator version
   string str5(str2.begin(), str2.begin() + 5);
   cout << str1 << endl;
   cout << str2 << endl;
   cout << str3 << endl;
   cout << str4 << endl;
   cout << str5 << endl;
   //  assignment operator
   string str6 = str4;
// clear function deletes all character from string
   str4.clear();
   //  both size() and length() return length of string and
```

Continue

```
 //  they work as synonyms
int len = str6.length(); // Same as "len = str6.size();"
cout << "Length of string is : " << len << endl;
// a particular character can be accessed using at /
// [] operator
char ch = str6.at(2); //  Same as "ch = str6[2];"
cout << "third character of string is : " << ch << endl;
//  front return first character and back returns last character
//  of string
char ch_f = str6.front();  // Same as "ch_f = str6[0];"
char ch_b = str6.back();   // Same as below
                    // "ch_b = str6[str6.length() - 1];"

cout << "First char is : " << ch_f << ", Last char is : "
    << ch_b << endl;

// c_str returns null terminated char array version of string
const char* charstr = str6.c_str();
printf("%s\n", charstr);
// append add the argument string at the end
str6.append(" extension");
//  same as str6 += " extension"
// another version of append, which appends part of other
// string
str4.append(str6, 0, 6);  // at 0th position 6 character

cout << str6 << endl;
cout << str4 << endl;
```

Continue

```
if (str6.find(str4) != string::npos)
        cout << "str4 found in str6 at " << str6.find(str4)
             << " pos" << endl;
    else
        cout << "str4 not found in str6" << endl;
    //  substr(a, b) function returns a substring of b length
    //  starting from index a
    cout << str6.substr(7, 3) << endl;

    //  if second argument is not passed, string till end is
    // taken as substring
    cout << str6.substr(7) << endl;
    //  erase(a, b) deletes b characters at index a
    str6.erase(7, 4);
    cout << str6 << endl;
    //  iterator version of erase
    str6.erase(str6.begin() + 5, str6.end() - 3);
    cout << str6 << endl;

    str6 = "This is a examples";

    //  replace(a, b, str)  replaces b characters from a index by str
    str6.replace(2, 7, "ese are test");

    cout << str6 << endl;

    return 0;
}
```

Output

```
first string
 first string
 #####
string first Length of string is : 6
 third character of string is : r
 First char is : s, Last char is : g
string
string extension
string
str4 found in str6 at 0 pos
Ext
 extension
string nsion strinion These are test examples
```