Edit & Run

CL-217 OBJECT ORIENTED PROGRAMMING LAB

LAB MANUAL 4

INSTRUCTOR: MUHAMMAD HAMZA

SEMESTER SPRING 2020

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. There are many cases where this flexibility helps. Examples of such cases are Linked List, Tree, etc.

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).

How is memory allocated/deallocated in C++?

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 also has two operators new and delete that perform the task of allocating and freeing the memory in a better and easier way.

This article is all about new and delete operators.

new operator

The new operator denotes a request for memory allocation on the Heap. 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 to use new operator: To allocate memory of any data type, the syntax is:

  ```
  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;
  ```

- Initialize memory: We can also initialize the memory using new operator:

  ```
  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.

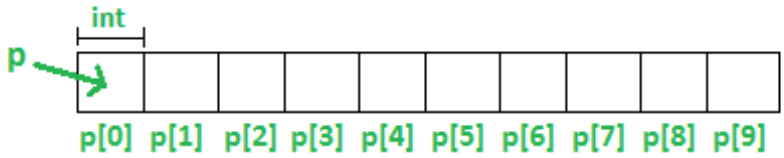
  ```
  pointer-variable = new data-type[size];
  ```

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

```
Example:

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

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 (scroll to section “Exception handling of new operator” in this article). Therefore, it may be good idea to check for the pointer variable produced by new before using it program.

```
int *p = new(nothrow) int;

if (!p)

{

   cout << "Memory allocation failed\n";
```

```
}
```

delete operator

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

Syntax:

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

delete pointer-variable;
```

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

Examples:

```
  delete p;

  delete q;
```

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:

   // It will free the entire array

   // pointed by p.

   delete[] p;
```

filter_none

edit

play_arrow

brightness_4

```
// C++ program to illustrate dynamic allocation
// and deallocation of memory using new and delete
#include <iostream>
```

```cpp
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;

    // freed the block of allocated memory
    delete[] q;

    return 0;
```

}
Output:

```
Value of p: 29

Value of r: 75.25

Value store in block of memory: 1 2 3 4 5
```

POINTER TO POINTER:

A pointer to a pointer is a form of multiple indirection or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.

A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, following is the declaration to declare a pointer to a pointer of type int −

```
int **var;
```

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example −

Live Demo

```
#include <iostream>

using namespace std;

int main () {
   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   // take the address of var
   ptr = &var;

   // take the address of ptr using address of operator &
```

```
   pptr = &ptr;

   // take the value using pptr
   cout << "Value of var :" << var << endl;
   cout << "Value available at *ptr :" << *ptr << endl;
   cout << "Value available at **pptr :" << **pptr << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Value of var :3000
Value available at *ptr :3000
Value available at **pptr :3000
```