Practical Lecture : Dynamic memory management

Quick Recap

Let’s take a quick recap of previous lecture –

- Order of execution of constructors and destructors
- Resolving ambiguities in inheritance
- Virtual base class.

Today’s Agenda

Today we are going to cover –

- Dynamic memory allocation using new and delete operators
- Memory leak and allocation failures
- Dangling, void, null , Wild pointer

Let’s Get Started-

Memory allocation

It is the process where memory for named variables is allocated by the compiler.

There are two ways to allocate-

Compile time allocation or static allocation of memory: where the memory for named variables is allocated by the compiler. Exact size and storage must be known at compile time and for array declaration, the size has to be constant.

Runtime allocation or dynamic allocation of memory: where the memory is allocated at runtime and the allocation of memory space is done dynamically within the program run . In this case, the exact space or number of the item does not have to be known by the compiler in advance. Pointers play a major role in this case.

Why dynamic Memory allocation

Often some situation arises in programming where data or input is dynamic in nature, i.e. the number of data item keeps changing during program execution.

For example : we are developing a program to process lists of employees of an organization. The list grows as the names are added and shrink as the names get deleted.

We cannot use arrays to store employee data as arrays cannot grow and shrink as we want.

Such situations in programming require dynamic memory management techniques

dynamic memory Allocation refers to performing memory management for dynamic memory allocation manually.

Dynamic memory allocation using new and delete operator

To allocate space dynamically, use the unary operator new, followed by the type being allocated.

new int; //dynamically allocates an integer type
new double; // dynamically allocates an double type
new int[60];

But the above-declared statements are not so useful as the allocated space has no names. But the lines written below are useful:

int * p; // declares a pointer p which points an int type data
p = new int; // dynamically allocate memory to contain one single element of type int and store the address in p

double * d; // declares a pointer d which points to double type data
d = new double; // dynamically allocate a double and loading the address in p

Practice question

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

int main()
{
    double* val = NULL;
    val = new double;
    *val = 38184.26;
    cout << "Value is : " << *val << endl;
    delete val;
}
```

Dynamic memory allocation for arrays

If you as a programmer; wants to allocate memory for an array of characters, i.e., a string of 40 characters. Using that same syntax, programmers can allocate memory dynamically as shown below.

```
char* val  = NULL;      // Pointer initialized with NULL value
val = new char[40];     // Request memory for the variable
```

Dynamic memory allocation for arrays

```
int * arr;
arr= new int [5];
```

The system dynamically allocates space for five elements of type int and returns a pointer to the first element of the sequence, which is assigned to arr (a pointer). Therefore, arr now points to a valid block of memory with space for five elements of type int.

Here, arr is a pointer, and thus, the first element pointed to by arr can be accessed either with the expression arr[0] or the expression *arr (both are equivalent). The second element can be accessed either with arr1] or *(arr+1), and so on...

Dynamic memory allocation for arrays

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 using constructors

```
class stud {
public:
    stud()
    {
        cout << "Constructor Used" << endl;
    }
    ~stud()
    {
        cout << "Destructor Used" << endl;
    }
};
int main()
{
    stud* S = new stud[6];
    delete[] S;
}
```

Delete operator

In most cases, memory allocated dynamically is only needed during specific periods of time within a program; once it is no longer needed, it can be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of operator delete, whose syntax is:

```
delete p;  //releases memory allocated using int *p;
delete[] p;  //releases memory allocated using  int *p=new int[5];
```

The first statement releases the memory of a single element allocated using new, and the second one releases the memory allocated for arrays of elements using new and a size in brackets ([]).

It is same as free() function in c which frees dynamically allocated memory using malloc() and calloc() functions.

Memory leak

- 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, So that place is reserved for no reason.
- It causes memory leak (memory is not deallocated until program terminates).
- Memory leak occurs when programmers create a memory in heap and forget to delete it.
- Memory leaks are particularly serious issues for programs like daemons and servers which by definition never terminate. In such cases programs will never terminate and memory will never be freed.
- To avoid memory leaks, memory allocated on heap should always be freed when no longer needed.

Memory allocation failure

If memory allocation using new is failed in C++ then how it should be handled?

When an object of a class is created dynamically using new operator, the object occupies memory in the heap.

Below are the major thing that must be kept in mind:

1. What if sufficient memory is not available in the heap memory, and how it should be handled? - using try and catch block
2. If memory is not allocated then how to avoid the project crash? – prevent memory crash by throwing an exception

Memory allocation failure

```
#include <iostream>
using namespace std;
int main()
{
   // Allocate huge amount of memory
   long MEMORY_SIZE = 0x7fffffff;
    // Put memory allocation statement
   // in the try catch block
   try {
      char* ptr = new char[MEMORY_SIZE];
       // When memory allocation fails, below line is not be executed
      // & control will go in catch block
      cout << "Memory is allocated“ << " Successfully" << endl;
   }
```

Memory allocation failure

```
// Catch Block handle error
   catch (const bad_alloc& e) {

      cout << "Memory Allocation“ << " is failed: “    << e.what()   << endl;
   }

   return 0;
}
```

Output:

Memory Allocation is failed: std::bad_alloc

The above memory failure issue can be resolved without using the try-catch block. It can be fixed by using nothrow version of the new operator.

Memory allocation failure

The nothrow constant value is used as an argument for operator new and operator new[] to indicate that these functions shall not throw an exception on failure but return a null pointer instead.

By default, when the new operator is used to attempt to allocate memory and the handling function is unable to do so, a bad_alloc exception is thrown.

But when nothrow is used as an argument for new, and it returns a null pointer instead.

This constant (nothrow) is just a value of type nothrow_t, with the only purpose of triggering an overloaded version of the function operator new (or operator new[]) that takes an argument of this type.

Memory allocation failure

```
#include <iostream>
using namespace std;
 int main()
{
   // Allocate huge amount of memory
   long MEMORY_SIZE = 0x7fffffff;
   // Allocate memory dynamically using "new" with "nothrow“  version of new
   char* addr = new (std::nothrow) char[MEMORY_SIZE];
   // Check if addr is having  proper address or not
   if (addr) {
        cout << "Memory is allocated“ << " Successfully" << endl;
   }
   else {
  // This part will be executed if large memory is allocated and failure occurs
      cout << "Memory  allocation“ << " fails" << endl;
   }
   return 0;
}
```

Output: Memory allocation fails

MCQ

What are the ways to allocate memory to variables?

1. Using malloc
2. Using calloc
3. Using new

A. 1 ,2
B. 1,2,3
C. Only 3
D. None of the above

MCQ

What are the ways to allocate memory to variables?

1. Using malloc
2. Using calloc
3. Using new

A. 1 ,2
B. 1,2,3
C. Only 3
D. None of the above

Answer: option B

MCQ

Find the odd man out.
Dynamic allocation, run time allocation, pointer, array

1. Dynamic allocation
2. Run time allocation
3. Pointer
4. Array

MCQ

Find the odd man out.
Dynamic allocation, run time allocation, pointer, array

1. Dynamic allocation
2. Run time allocation
3. Pointer
4. Array

Answer: Array as it is static allocation

MCQ

Which of the following in not correct way to dynamically allocate memory?

1. int new *p;
2. int *p=new int;
3. int *p=new int[10];
4. classA objA=new classA();

MCQ

Which of the following in not correct way to dynamically allocate memory?

1. int new *p;
2. int *p=new int;
3. int *p=new int[10];
4. classA objA=new classA();

Answer: option A

MCQ

Which of the following in not correct about dynamically allocated memory?

1. It is necessary to free memory allocated dynamically to avoid memory leaks
2. To allocate memory dynamically we use new operator
3. We must use delete operator to de-allocate dynamically allocated memory
4. The dynamic memory requested by our program is allocated by the system from the memory stack

MCQ

Which of the following in not correct about dynamically allocated memory?

1. It is necessary to free memory allocated dynamically to avoid memory leaks
2. To allocate memory dynamically we use new operator
3. We must use delete operator to de-allocate dynamically allocated memory
4. The dynamic memory requested by our program is allocated by the system from the memory stack

Answer: option 4 . It is allocated from heap

MCQ

Choose an incorrect option.

How to handle memory allocation failure?

1. Using try and catch block
2. Using nothrow argument for new operator
3. By avoiding memory leaks
4. By not allocating memory dynamically

MCQ

Choose an incorrect option.

How to handle memory allocation failure?

1. Using try and catch block
2. Using nothrow argument for new operator
3. By avoiding memory leaks
4. By using overloaded version of new operator

Answer: Option C . Rest all are ways to handle memory allocation failure

Dangling pointer

Dangling pointer is a pointer pointing to a memory location that has been freed (or deleted) or it goes out of scope.

```
//when variable goes out of scope
int main() {
   int *p;
 //some code//
{
      int c; p=&c;
 }
   //some code//
   //p is dangling pointer here because variable c does not exist here, so p is now pointing to memory location that is freed.
}
```

Dangling pointer

Dangling pointer is a pointer pointing to a memory location that has been freed (or deleted) or it goes out of scope.
//when memory is freed or deleted

```
#include <iostream>
using namespace std;
int main()
{
    int *ptr = (int *)malloc(sizeof(int));
    // After below free call, ptr becomes a
    // dangling pointer
    free(ptr);
    // No more a dangling pointer
   //  ptr = NULL;
}
```

Void pointer

Void pointer in C is a pointer which is not associate with any data types. It points to some data location in storage means points to the address of variables. It is also called general purpose pointer.

It has some limitations

Pointer arithmetic is not possible of void pointer due to its concrete size.

It can’t be used as dereferenced.

Void pointer

```
#include<iostream>
using namespace std;
int main() {
   int a = 7;
   float b = 7.6;
   void *p;
   p = &a;
   cout<<*((int*) p)<<endl ;
   p = &b;
   cout<< *((float*) p) ;
   return 0;
}
```

Null pointer

Null pointer is a pointer which points nothing.

Some uses of null pointer are

- To initialize a pointer variable when that pointer variable isn't assigned any valid memory address yet.
- To pass a null pointer to a function argument if we don't want to pass any valid memory address.
- To check for null pointer before accessing any pointer variable. So that, we can perform error handling in pointer related code e.g. dereference pointer variable only if it's not NULL.

Null pointer

```
#include <iostream>
using namespace std;
int main() {
   int *p= NULL; //initialize the pointer as null.
   cout<<"The value of pointer is ";
   cout<<p;
   return 0;
}
```

Output:

The value of pointer is 0

Wild pointer

- Wild pointers are pointers those are point to some arbitrary memory location. (not even NULL)
- They may cause the programs to crash or misbehave.
- They point to some memory location even we don’t know

```
int main() {
   int *ptr; //wild pointer
   *ptr = 5;
}
```

How to avoid wild pointers?

by allocating memory explicitly using malloc or new functions like follows:

```
 int *ptr= (int * ) malloc(sizeof(int)); // avoid wild pointer
   *ptr = 5;
```

Wild pointer

How to avoid wild pointers?

1. by allocating memory explicitly using malloc or new functions like follows:

```
int *ptr= (int * ) malloc(sizeof(int)); // avoid wild pointer
  *ptr = 5;
```

2. By initializing the address

```
int main()
{
  int  *p; /* wild pointer */
  int a = 10;
  p = &a;  /* p is not a wild pointer now*/
  *p = 12; /* This is fine. Value of a is changed */
}
```

Assignment

Write a C++ program to create an array of five Student CLASS. You can store attributes of your interest in student class. Use dynamic way of memory allocation to objects.

QNA Time

Any Questions ??

Any Questions??

Thank You!

See you guys in next class.