Dynamic Memory Management

- In C language, if we want to manage memory dynamically then we should use functions declared in header file.
- Following are the functions declared in header file:
  - void* malloc(size_t size);
  - void* calloc(size_t count, size_t size);
  - void* realloca(void *ptr, size_t newSize);
  - void* free(void *ptr);
- In C++, to manage memory dynamically, we should use **new** and **delete** operator.
- new operator is used to allocate memory and delete operator is used to deallocate memory.

```
Memory allocation and deallocation for single integer variable:
```

```
    int *ptr = new int;
        //int *ptr = ( int* )::operator new(sizeof( int ) );

        *ptr = 125;     //Dereferencing
        cout<<"Value    :       "<<*ptr<<endl;//Dereferencing

        delete ptr;
        //::operator delete(ptr);

        ptr = NULL;
```

```
int *p1 = new int;
```

In above statement, we are allocating space for single variable. Memory will be initialized to garbage value.

```
int *p1 = new int();
```

In above statement, we are allocating space for single variable. Memory will be initialized to zero.

```
int *p1 = new int(3);
```

In above statement, we are allocating space for single variable. Memory will be initialized to 3.

```
Memory allocation and deallocation for single dimensional array:
    int *ptr = new int[ 3 ];
        //int *ptr = ( int * )::operator new[](3 * sizeof( int ) );

        for( int index = 0; index < 3; ++ index )
        {
                cout<<"Enter element    :       ";
                cin>>ptr[ index ];
        }

        for( int index = 0; index < 3; ++ index )
                cout<<ptr[ index ] <<endl;

        delete[] ptr;
        //::operator delete[](ptr);

        ptr = NULL;
```

```
Memory allocation and deallocation for multi dimensional array:
```

```
    int **ptr = new int*[ 2 ];
        for( int index = 0; index < 2; ++ index )
                ptr[ index ] = new int[ 3 ];

        //TODO I/P and O/P

        for( int index = 0; index < 2; ++ index )
                delete[] ptr[ index ];
        delete[] ptr;
        ptr = NULL;
```

- If malloc function faiils to allocate memory then return NULL. If new operator fails to allocate memory then it throws std::bad_alloc exception.
- If we create dynamic object using malloc function then constructor do not call but if we create dynamic object using new operator then constructor gets called.

> Interview Question : What is the difference between malloc and new.

Array Of Objects

Static memory allocation

```
Complex arr[ 3 ];
        for( int index = 0; index < 3; ++ index )
                arr[ index ].acceptRecord();
        for( int index = 0; index < 3; ++ index )
                arr[ index ].printRecord();
```

Dynamic memory allocation

```
Complex *arr = new Complex[ 3 ];
        for( int index = 0; index < 3; ++ index )
                arr[ index ].acceptRecord();

        for( int index = 0; index < 3; ++ index )
                arr[ index ].printRecord();

        delete[] arr;
        arr = nullptr;
```

Destructor

- It is a member function of the class that is used to deinitialize object.
- Constructor calling sequence is depends on order of object declaration but destructor calling sequence is exactly opposite of constructor calling sequence.
- Due to following reasons, destructor is considered as special member function of a class:
  - Its name is same as class name and always precedes with tild operator(~).
  - It doesn't take any parameter and doesn't return any value.
  - It is designed to call implicitly.
- We can not call constructor on object, pointer and reference explicitly. It is designed to call implicitly.
- Destructor is designed to call implicitly but we can call it on object, pointer and reference explicitly.
- We can declare destructor **inline** and **virtual** only.

> Interview Question: Can we overload destructor?

- If we do not define destructor inside class, then compiler provides one destructor for the class by default. It is called default destructor.
- For the programmer, default destructor is empty destructor:

```
~Array( void )
{
    //Empty body
}
```

- Default destructor do not take any action on data member declared by the programmer. If we want to release resources(memory, file, socket etc.) hold by the object then we should define destructor inside class.

> Interview Question: Can we declare destructor private ** Destructor do not deallocate memory of object rather it is used to relase the resources hold by the object. **

Object Copy Semantics

1. Shallow Copy
2. Deep Copy
3. Lazy Copy

Shallow Copy

- It is also called as bitwise copy / bit-by-bit Copy.

```
int num1 = 10;
int num2 = num1;     //Shallow Copy
```

```
Complex c1(10,20);
Complex c2;
c2 = c1;     //Shallow Copy
```

- Process of creating copy of the object from existing object is called shallow Copy.

Deep Copy

- It is also called as Member-wise copy.
- Process of creating copy by modifying some state is called deep copy.
- Conditions to create deep copy
  1. Class must contain at least one pointer type data member.
  2. Class must contain user defined destructor
  3. We must create copy of the object.
- Steps to create deep copy:
  1. Copy the required size from source object into destination object.
  2. Allocate new resource for destination object.
  3. Copy the contents from resource of source object into destination object.
- Location to create deep copy:
  1. In case of assignment, we should create deep copy inside assignment operator function.
  2. and in rest of the condition, we should create deep copy inside copy constructor.

Copy Constructor

- It is parameterized constructor of the class, which take only one parameter of same type but as a reference.
- General Syntax of Copy constructor is:

```
//ClassName *const this = address of dest. object
//const ClassName &other = reference of source object
ClassName( const ClassName &other )
{
    //TODO : Create copy of the object
}
```

- Job of copy constructor is to initialize object from existing object of same class.
- Copy constructor gets called in following condition:

1. If we pass object of user defined type as a argument to the function by value then on function parameter copy constructor gets called.
2. If we return object from function by value then compiler implicitly create one annonymous object. On that annonymous object compiler invoke copy constructor.
3. If we initialize object from existing object of same class then on newly created object copy constructor gets called.
4. If we throw object then its copy gets created on runtime stack. On that object, compiler invoke copy constructor.
5. If we catch object by value then on catching object, copy constructor gets called.

> Interview Question: In which conditions copy constructor gets called.

- If we do not define copy constructor inside class then compiler provides one copy constructor for that class by default. It is called **Default Copy Constructor**.
- By default, it creates shallow copy.

> If we want to create shallow copy of the of the object then we should not define copy constructor inside class.