Object Oriented Programming Using C++

Ketan Kore
Ketan.Kore@sunbeaminfo.com

Inline Function

- Process:
    1. Program in execution is called process.
    2. Running instance of a program is called process.
- Thread:
    1. Sub process / lightweight process is called thread.
    2. Thread is a separate path of execution which runs independently.
- Operating system maintains one stack(environment stack) per thread which contains stack frames.
- Stack frame contains information of called function(Function Activation Record[FAR]):
    1. Function parameter, Local Variable
    2. Other function call
    3. Temporary storage
    4. Return address.

Inline Function

- In simple words, if we call function then compiler generates function activation record.
- FAR get generated per function call.
- Due to FAR, giving call to the function is considered as overhead to the compiler.
- If we want to avoid this overhead then we should declare function inline.
- inline is keyword in C++.
- macro is command to the pre-processor but inline is request to the compiler.

Inline Function

- In following conditions, function can not be considered as inline.
    1. Use of loop(for/while) inside function
    2. Use of recursion.
    3. Use of jump statements.
- Excessive use of inline function increases code size.
- Except main function, we can declare any global function inline.
- We can not separate inline function code into multiple files.

Exception Handling

- Types of Error

  1. **Compiler error**
     - ➢ It gets generated due to syntactical mistake.
  2. **Linker error**
     - ➢ If we try to use any variable/function without definition then we get linker error.
  3. **Bug**
     - ➢ Logical error is also called bug.
  4. **Runtime error**
     - ➢ It gets generated due to wrong input.

Exception Handling

- **Operating System Resources**
  1. Memory
  2. File
  3. Thread
  4. Socket
  5. Connection
  6. Hardware resources(CPU, Keyboard, Mouse, Monitor)
  7. System call
- Since operating system resources are limited, we should use it carefully. In other words, we should avoid their leakage.

Exception Handling

- **Definition**
  1. Runtime error is also called as exception.
  2. Exception is an object which is used to send notification to the end user of the system if any exceptional situation occurs in the program.
- **Need of exception handling**
  1. If we want to manage OS resources carefully in the program we should handle exception.
  2. If we want to handle all the runtime errors at single place so that we can reduce maintenance of the application.
- **How to handle exception**
  - In C++, we can handle exception using 3 keywords:
    1. Try
    2. Catch
    3. throw

Exception Handling

- **try**

  1. try is a keyword in C++.
  2. If we want to keep watch on group of statements then we should use try block.
  3. try block is also called as try handler.
  4. We can not define try block after catch block/handler.
  5. try block must have at least one catch block.

- **throw**

  1. throw is keyword in C++.
  2. Exception can be raised implicitly or we can generate it explicitly.
  3. If we want to generate exception explicitly then we should use throw keyword.
  4. throw statement is a jump statement
  5. Jump statements in C/C++ : break, return, goto, continue, throw

Exception Handling

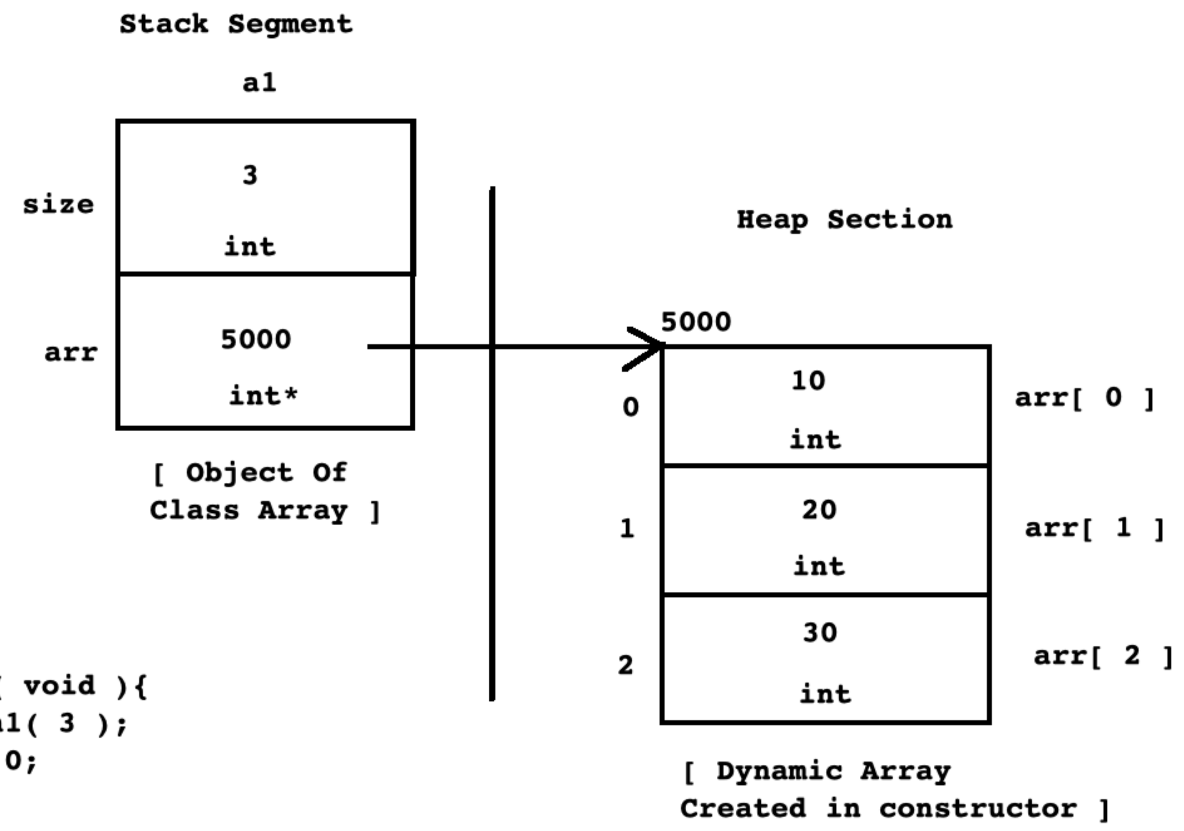
- **catch**

  1. catch is keyword in C++.
  2. If we want to handle exception thrown from try block then we should use catch block.
  3. Catch block is also called as catch handler.
  4. Single try block may have multiple catch blocks.
  5. We can not define catch block before try block/handler.
  6. For thrown exception, if matching catch block is not available then C++ runtime environment implicitly give call to the std::terminate() function which implicitly calls std::abort( )function.
  7. A catch block, which can handle all type of exception is called default/generic catch block. E.g. **catch(. . .){ }**
  8. We must define generic catch block after all specific catch block.

Destructor

- Destructor is a member function of a class which is used to release resources hold by the object.
- If we do not define destructor inside class then compiler provide one destructor for that class by default. It is called default destructor.
- Default destructor do not perform any clean up operation on data member declared by the programmer.
- **Destructor calling sequence is exactly opposite of constructor calling sequence.**
- **We can not declare destructor static, constant or volatile but we can declare destructor inline and virtual.**

Destructor



```
int main( void ){
  Array a1( 3 );
  return 0;
}
```

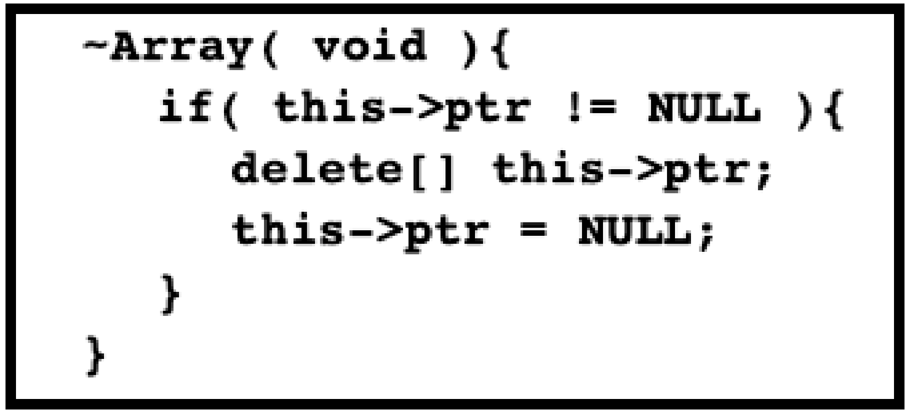
Destructor

- Due to following reasons, destructor is considered as special member function of the class:
    1. Its name is same as class name and always precedes with ~ operator.
    2. It doesn't have a return type.
    3. It doesn't have any parameters.
    4. We can call destructor on object/pointer/reference explicitly. But it is designed to call implicitly.
- We can overload constructor but we can not overload destructor.
    - ➢ To overload any function either number of parameters must be different or if number of parameters are same then type of parameters must be different or order of type of parameters must be different.
    - ➢ Since destructor do not take any parameter, we can not overload destructor.

Destructor

- If constructor and destructor are public then we can create object of the class inside member function as well as non member function.

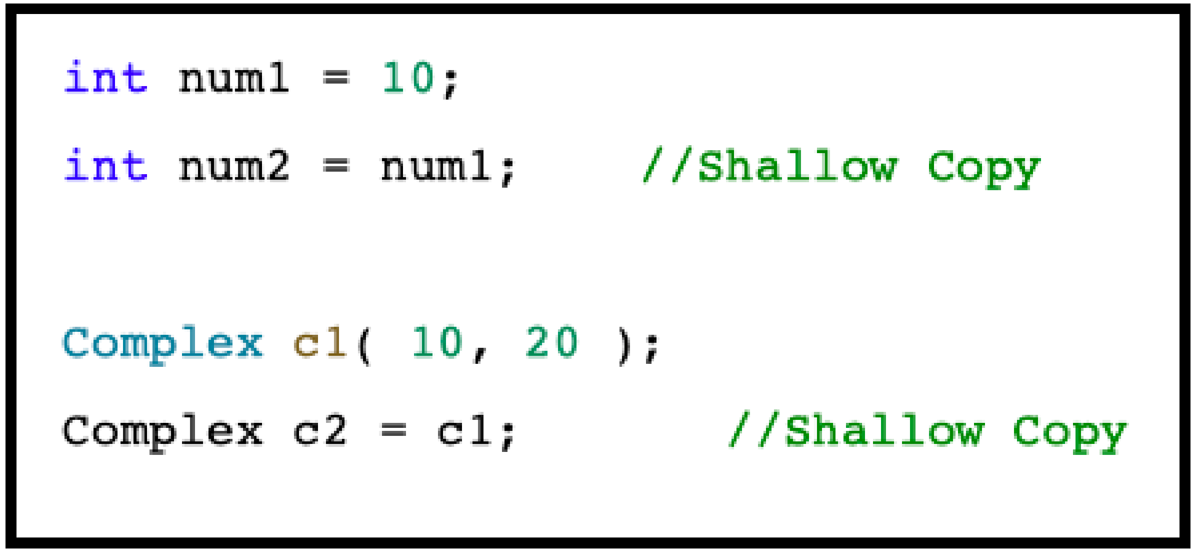
- If constructor or destructor is private then we can create object of the class inside member function only.



```
~Array( void ){
   if( this->ptr != NULL ){
      delete[] this->ptr;
      this->ptr = NULL;
   }
}
```

Shallow Copy

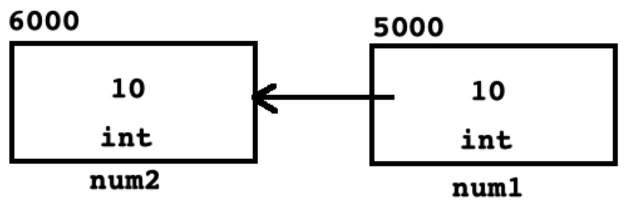
- Process of copying contents of object into another object as it is, is called shallow copy.
- Shallow copy is also called as bitwise copy / bit-by-bit copy.
- Compiler by default creates shallow copy.



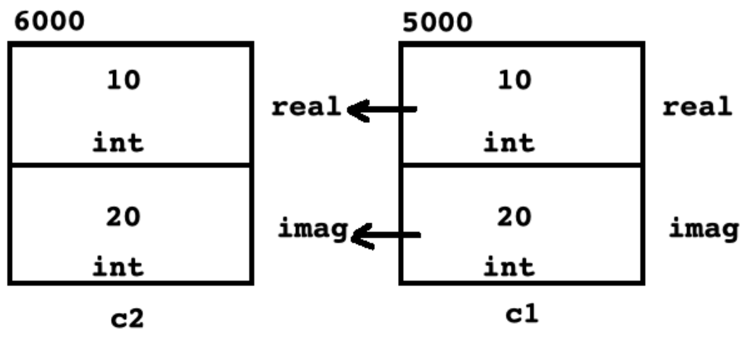
```
int num1 = 10;
int num2 = num1;     //Shallow Copy


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

Shallow Copy

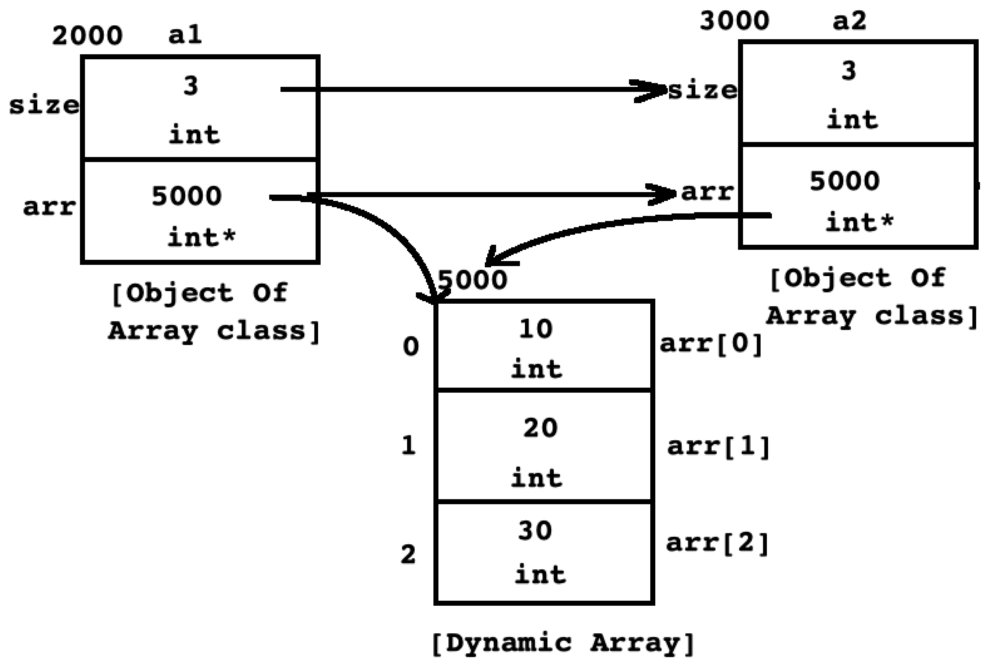


```
int num1 = 10;
int num2 = num1; //Shallow Copy
```



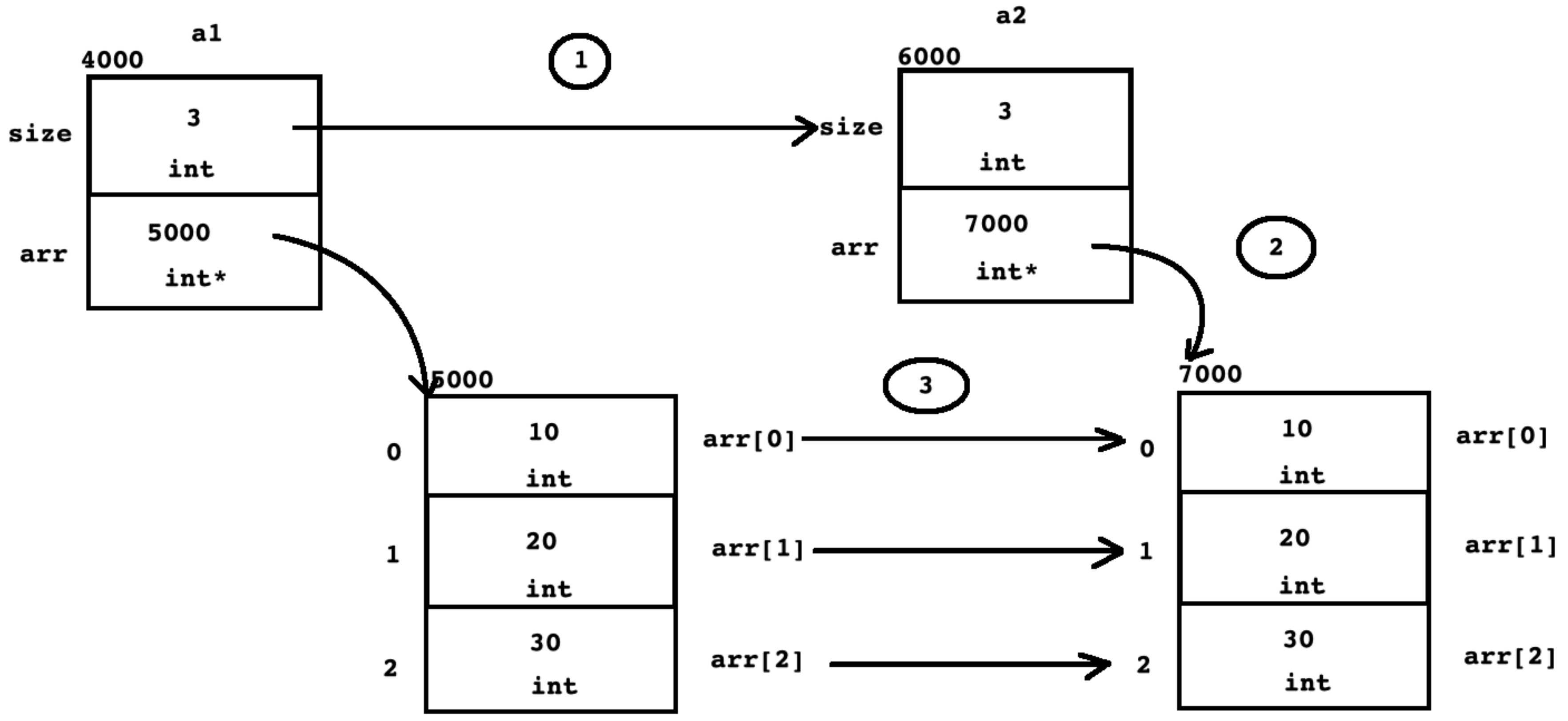
```
Complex c1( 10, 20 );
Complex c2 = c1;  //Shallow Copy
```

```
Array a1( 3 );
a1.acceptRecord( );
Array a2 = a1;
a2.printRecord( );
```



Shallow Copy / Bitwise / Bit-By-Bit Copy

Deep Copy



Deep Copy / Memberwise Copy

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(length, row/col/ array size etc) from source object into destination object.
    2. Allocate new resource for destination object.
    3. Copy the contents from resource of source object into resource of destination object.
- **Location to create deep copy**
    1. In case of assignment, we should create deep copy inside assignment operator function.
    2. In rest of the conditions, we should create deep copy inside copy constructor.

Copy Constructor

- It is parameterized constructor of a class which take single parameter of same type as a reference.
- Job of copy constructor is to initialize object from existing object.

Syntax

```
//ClassName &other = source object;
//ClassName *const this = address of destination object;
ClassName( const ClassName &other ){
    //TODO : Shallow / Deep Copy
}
```

Example

```
//Complex &other = c1;
//Complex *const this = &c2;
Complex( const Complex &other ){
    //Shallow Copy
    this->real = other.real;
    this->imag = other.imag;
}
```

- Since copy constructor take single parameter it is considered as parameterized constructor.

Copy Constructor

- If we do not define copy constructor inside class then compiler generates default copy constructor for the class. By default it creates shallow copy.

- **Copy constructor gets called in five conditions**
  1. If we pass object as a argument to the function by value then its copy gets created in function parameter. On function parameter copy constructor gets called.
  2. If we return object from function by value then its copy gets created inside in memory(anonymous object). On anonymous object, copy constructor gets called.
  3. If we initialize object from another object of same class then on newly created object copy constructor gets called.
  4. If we throw object then its copy gets created on environmental stack. Compiler invoke copy constructor on object created on stack.
  5. If we catch object by value then on catching object copy constructor gets called.

Thank you