Object Oriented Programming Using C++

Ketan Kore

Ketan.Kore@sunbeaminfo.com

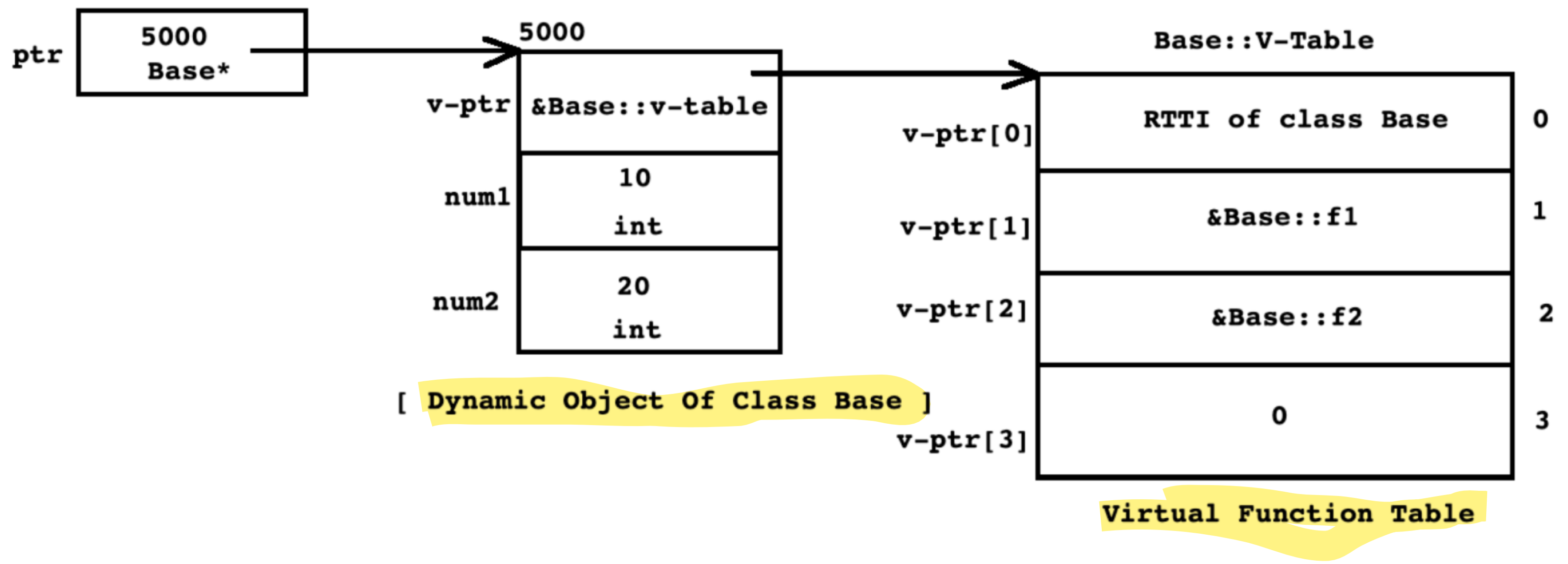
Virtual Function Table and Pointer

- **Virtual Function Table**
  - If we declare function virtual then compiler implicitly creates one table(array/structure) to store address of that virtual function. Such table is called virtual function table.
  - It is also called as vf-table / v-table.
  - In short, **a table which stores address of virtual functions declared inside class is called v-table.**
  - Compiler generates v-table once per class.
- **Virtual Function Pointer**
  - To store address of virtual function table, compiler implicitly declare one pointer as a data member inside class. Such data member is called virtual function pointer/vf-pointer/v-ptr.
  - In short, **a pointer which stores address of virtual function table is called v-ptr.**
  - Compiler generates v-ptr once per object.

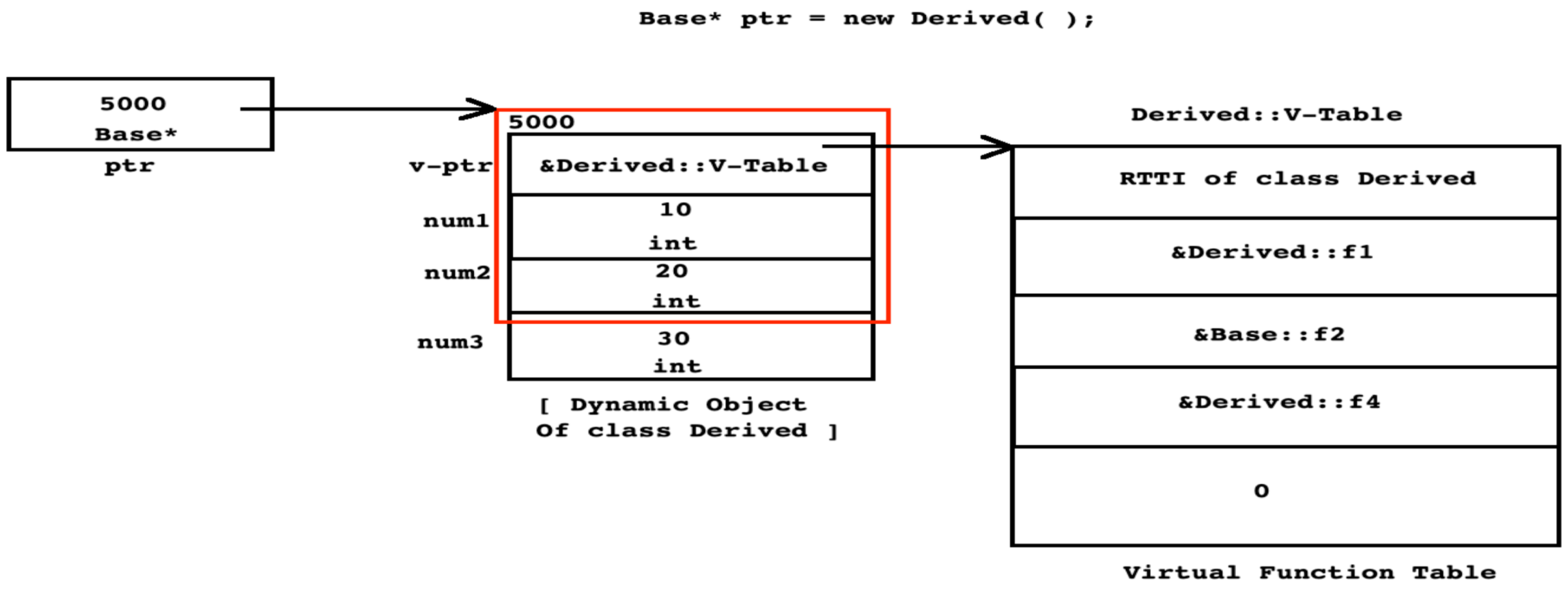
- **Size of object of derived class = size of all the non static data member declared in base class + size of all the non static data member declared in derived class + 2/4/8 bytes (If base class / derived class contains at least one virtual function).**

Virtual Function Table and Pointer

```
Base *ptr = new Base( );
```



Virtual Function Table and Pointer



Virtual Function

- **Why we can not declare constructor virtual?**
  - ➢ Virtual function is designed to call on base class pointer/reference
  - ➢ We can not call constructor on object/pointer/reference explicitly. It is designed to call implicitly.
  - ➢ Since we can not call constructor on pointer/reference, we can not declare it virtual.
- **In case of upcasting, to call destructor of derived class, it is necessary to declare destructor in base class virtual.**
- **Points to remember**
  1. According to problem statement / client's requirement, if implementation of base class member function is logically 100% complete(no need to override) then we should declare it non virtual.
  2. According to problem statement / client's requirement, if implementation of base class member function is logically incomplete / partially complete (need to override) then we should declare it virtual.
  3. According to problem statement / client's requirement, if implementation of base class member function is logically 100% incomplete(must override) then we should declare it pure virtual.

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.

Template

- If we want to write generic code in C++, then we should use template.
- Objective

  1. Not to reduce code size

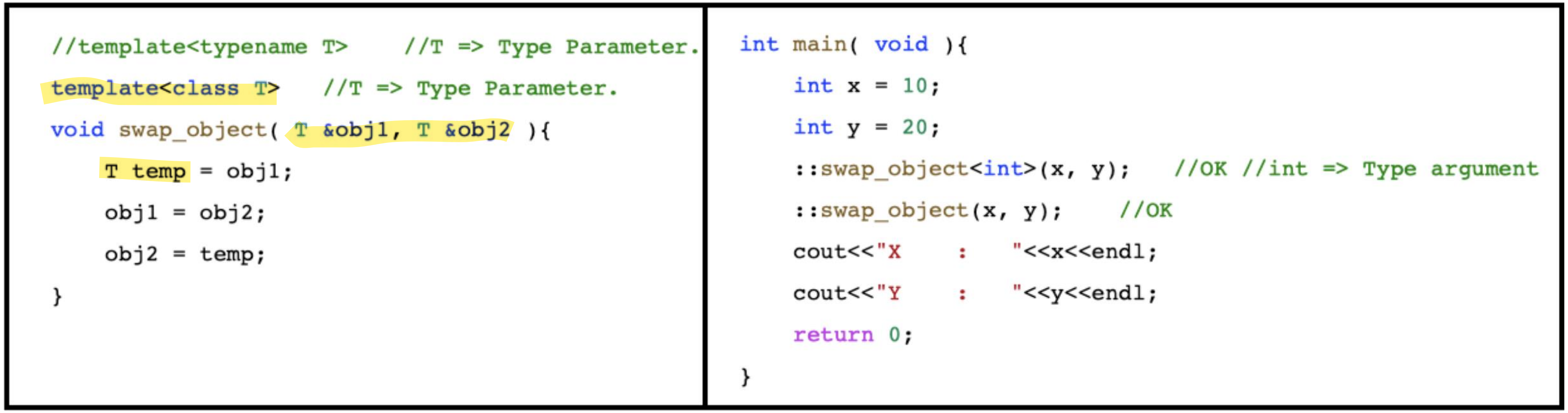
  2. Not to reduce execution time

  3. To reduce developers effort.
- It is possible by passing data type as a argument.
- Types of template:

  1. Function template

  2. Class Template

Function Template



```
//template<typename T>    //T => Type Parameter.
template<class T>    //T => Type Parameter.
void swap_object( T &obj1, T &obj2 ){
    T temp = obj1;
    obj1 = obj2;
    obj2 = temp;
}
```

```
int main( void ){
    int x = 10;
    int y = 20;
    ::swap_object<int>(x, y);    //OK //int => Type argument
    ::swap_object(x, y);    //OK
    cout<<"X    :    "<<x<<endl;
    cout<<"Y    :    "<<y<<endl;
    return 0;
}
```

Template

- An ability of compiler to detect and pass type of argument implicitly to the function is called type inference.
- Template is mainly designed for data structure and algorithm.
- By passing data type as a argument, we can define generic code in C++. Hence parametrized type is called template.
- We can not divide template code into multiple files.

string s1= "pune", s2="Kara
swap_objct ______ (s1,s2);

$\uparrow$

type inference

Standard Template Library

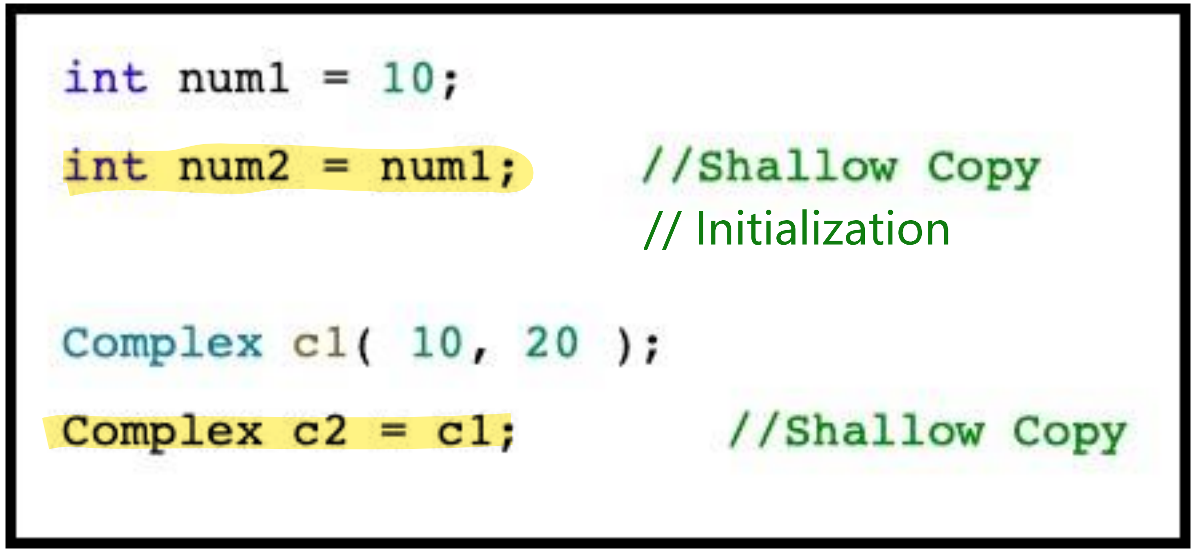
- Library of generic classes and algorithms.

- Components of STL:
    1. Container
    2. Iterator
    3. Algorithms
    4. Functions

- Book : C++ Complete reference (Part -3): Herbert Schildt

Shallow Copy

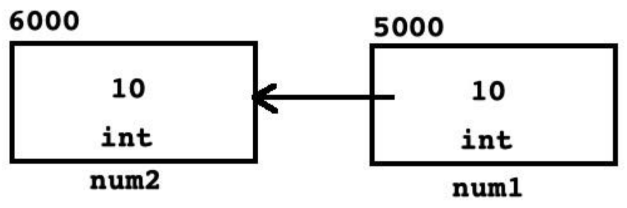
- Process of copying contents of object into another object as it is, is called shallow copy.
- Shallow copy is also called as bitwise copy / bit-by-bit copy.
- Compiler by default creates shallow copy.



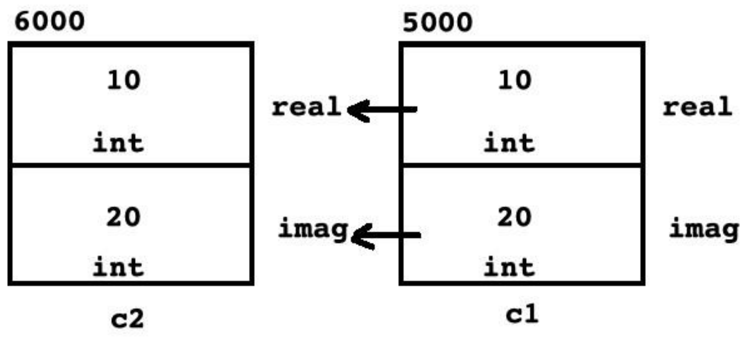
```
int num1 = 10;
int num2 = num1;      //Shallow Copy
                      // Initialization

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

Shallow Copy

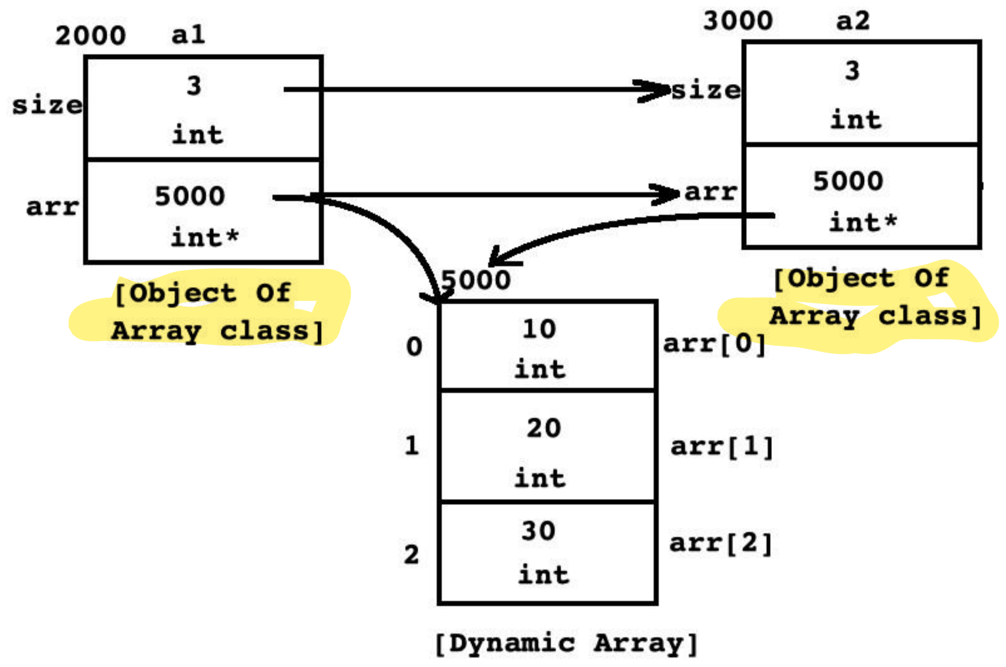


```
int num1 = 10;
int num2 = num1; //Shallow Copy
```



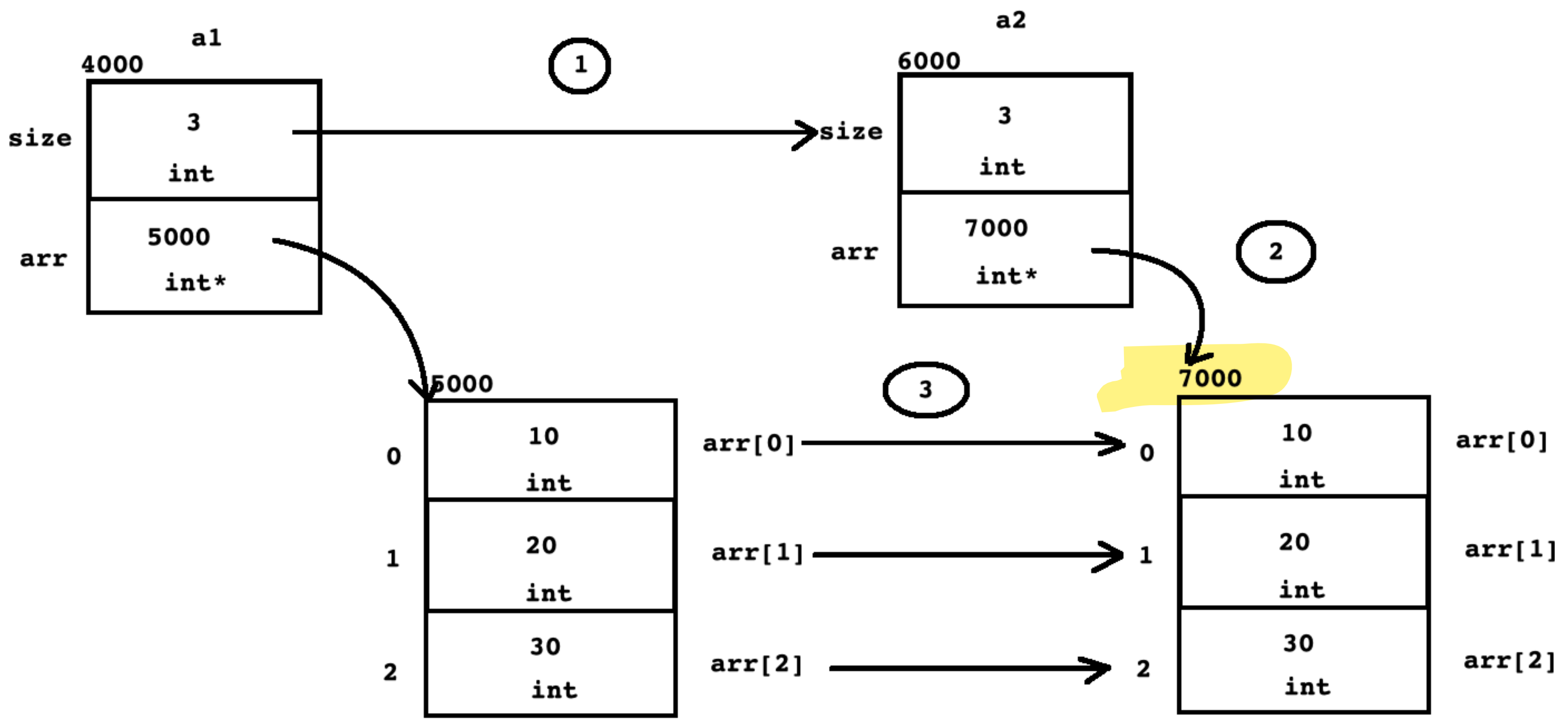
```
Complex c1( 10, 20 );
Complex c2 = c1;  //Shallow Copy
```

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



Shallow Copy / Bitwise / Bit-By-Bit Copy

Deep Copy



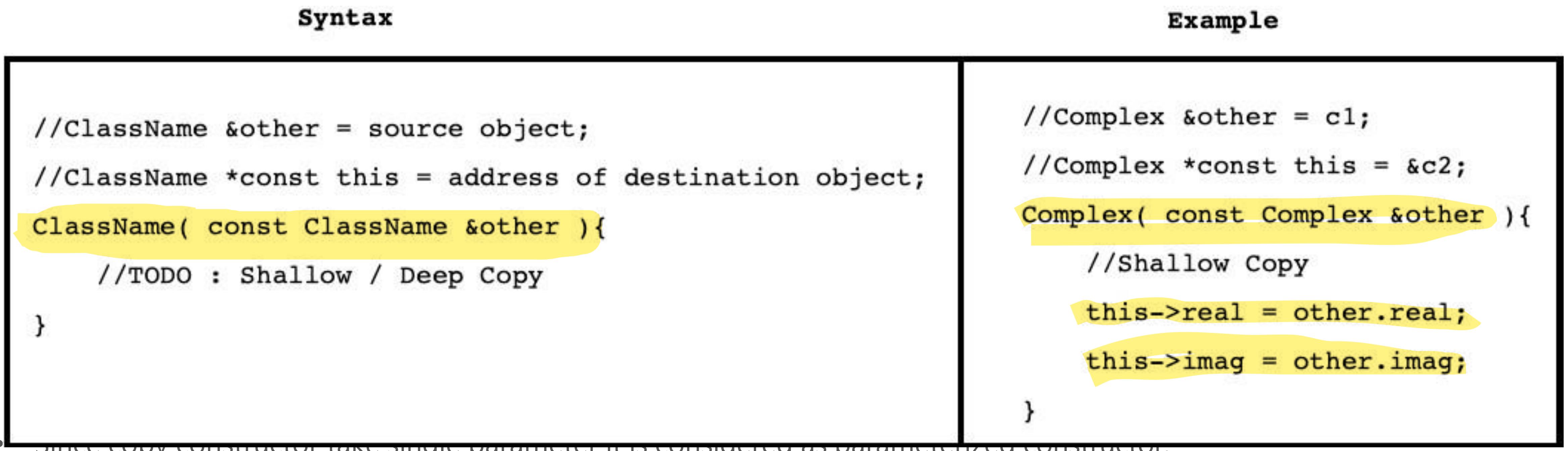
Deep Copy / Memberwise Copy

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