C++

OOP AND MAJOR PILLARS

Different roles

End User : Who is not a programmer. Who executes your programs. Who is non-technical.

Programmer User : Who writes main( ) and does programming for interaction with end user, who uses class library, call functions, create menus, thinks about end user

Developer User : Who designs classes, functionalities with classes and thinks about programmer user. He is senior and experienced.

Procedural Oriented Programming(POP)	Object Oriented Programming(OOP)
Problem is divided in to small logically independent tasks called as procedures or functions.	Problem is divided in to small real time entities called as objects
Importance is given to different functions and the sequence in which these functions are called	Importance is given to data and not to functions, actually functions are called on real time entities i.e. objects
We use only global and local scope. Does not have any access specifiers as such.	There are 3 access specifiers in C++ public, private and protected
Data is not hidden.	Data may be kept hidden by making it private
Examples : C, COBOL, PASCAL, FORTRAN etc.	Examples: C++, Java, C#, SmallTalk etc.

class

1. Class is classification of an entity
2. It is user-defined data type
3. It is collection of data members and member functions
4. It is blue-print for an object.
5. It is template for an object.
6. We declare class once in our program and create many objects of that class
7. Class is best example of encapsulation
8. At the time defining a class we do data level abstraction.
9. It is a logical entity. (concept)

object

1. It is a variable of type class
2. It is representation of an entity which has state, behavior and identity
3. object's life cycle - object is born, it undergoes many changes, it dies.
4. It is physical entity. (memory is allocated for object when created and memory is released when object goes out of scope).
5. We can create multiple objects of a class. Each object will have its own life-cycle.
6. Value of member variable gives us state of object at any given point of time.
7. We call functions of class with object and state of object is available in function.

Abstraction

Abstraction means showing only essential information while hiding the implementation details.

For example, a person driving a car. The driver knows that pressing the accelerator increases the car’s speed, and applying brakes stops it. However, the driver doesn’t need to understand the complicated inner workings of the car’s mechanisms.

Encapsulation

To keep data and functionalities together in one single unit.

Encapsulation is achieved through private and public access specifiers.

Data members (variables) are kept private and member functions are kept public.

my_laptop.on() - When the function / method is called on a particular object, the state of that object is going to change.

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users.

setter and getter methods

These methods are written for all possible member variables.

setter methods are used to set values of member variables.

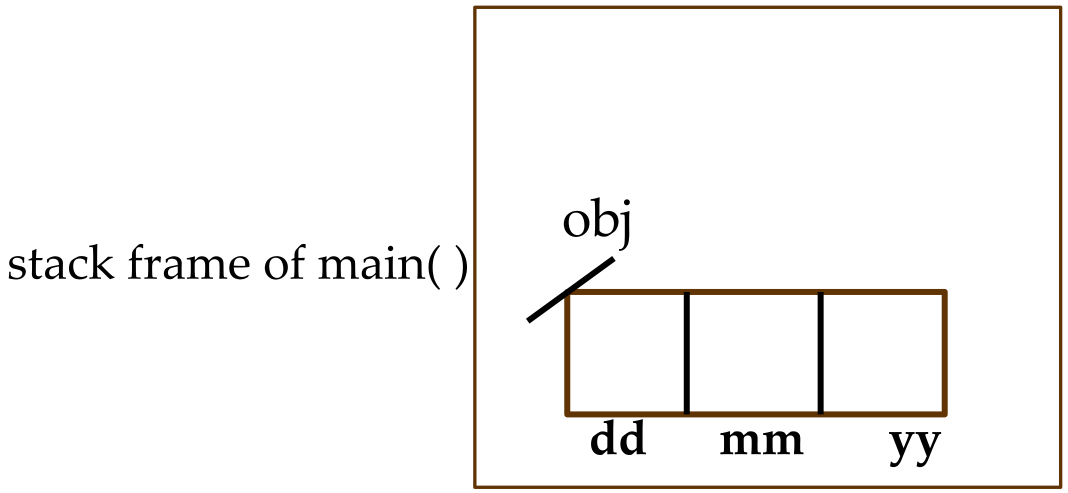
getter methods are used to get values of member variables.

Sample code to understand and write class **date**

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

class date {
   int dd, mm, yy;

public:
   void set_date(int d, int m, int  y)
    {
        dd = d;
        mm = m;
        yy = y;
    }
   void display( )
    {
        cout << “\n”<< dd << “/” << mm <<“ / “<<yy;
    }
};  //end of class
```

```
int main( )
{
     date obj;
     obj.set_date(20,10,2023);
     obj.display( );
     return 0;
}
```

object allocation on stack in RAM



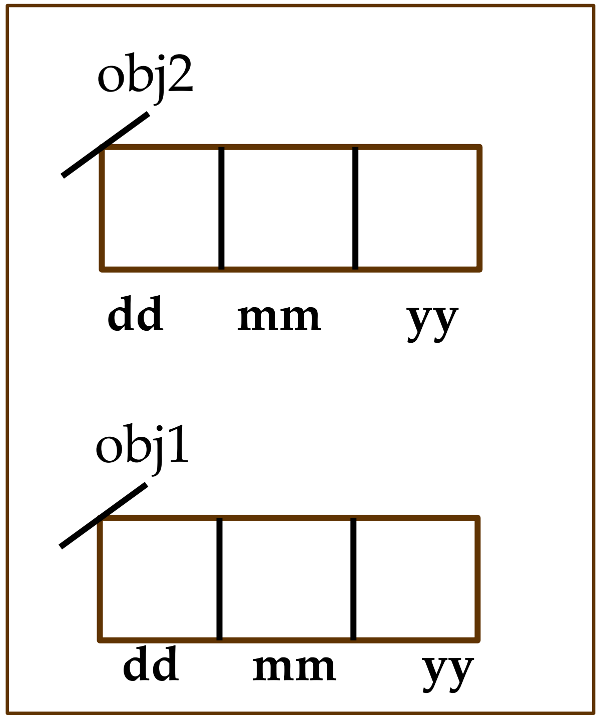
Sample code to understand and write class **date**

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

class date {
   int dd, mm, yy;

public:
   void set_date(int d, int m, int  y)
    {
        dd = d;
        mm = m;
        yy = y;
    }
   void display( )
    {
        cout << "\n"<< dd << "/" << mm <<" / "<<yy;
    }
};  //end of class
```

```
int main( )
{
     date obj1, obj2;
     obj1.set_date(20,10,2023);
     obj1.display( );
     obj2.set_date(23,10,2023);
     obj2.display( );
     return 0;
}
```

object allocation on stack in RAM

stack frame of main( )



Assignment: Do it yourself.

1. Declare and define class time having hh,mm,ss. Write accept_time( ) and display_time( ) in format “ hh : mm : ss “
2. Declare and define class point having x, y. Write set_point(int, int) and display_point( ) in format (x , y)

OOP Terminologies

Members variables / data members / fields: variables declared in the class

Member functions / methods: functions declared in the class

```
class date {
   int dd, mm, yy;                    Member variables

public:
   void set_date(int dd, int mm, int  yy)
    {
        ...
                                          Member function
        ...
    }
   void display( )
    {
                                          Member function
    }
};  //end of class
```

We can declare as many member variables and member functions in a class.

this pointer

- **this pointer** is passed implicitly in every function of class and it contains address of calling object.

- Each object gets its own copy of the data member inside member function through this pointer.

- The ‘this’ pointer is passed as a hidden argument to all member function calls and is available as a local variable within the body of all functions.

- Following are the situations where ‘this’ pointer is used:
  1) When local variable’s name is same as member’s name
  2) To return reference of the calling object from function.

Sample code to understand this Pointers

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

class date {
   int dd, mm, yy;

public:
   void set_date(int dd, int mm, int  yy)
    {
        this->dd = dd;
        this->mm = mm;
        this->yy = yy;
    }
   void display( )
    {
        cout << "\n"<< dd << "/" << mm <<" / "<<yy;
    }
};  //end of class
```

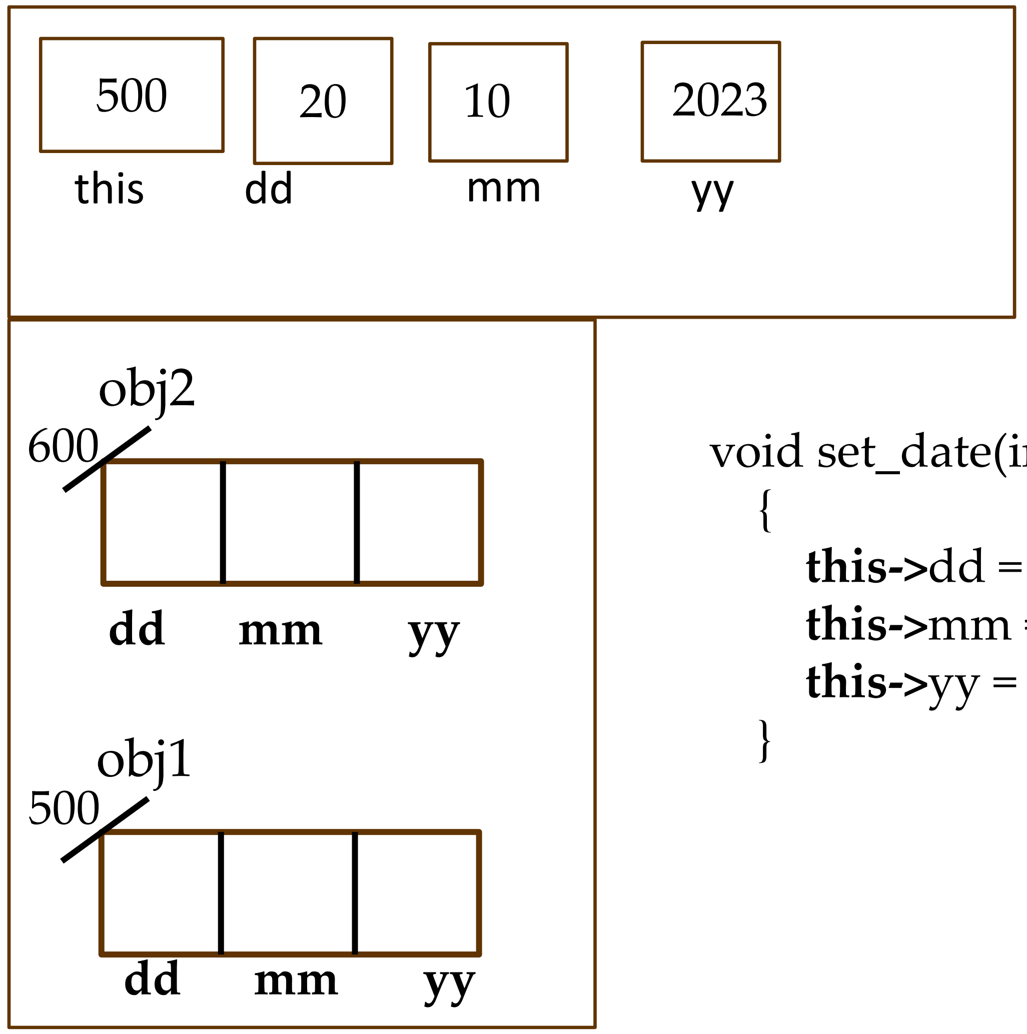
```
int main( )
{
     date obj1, obj2;
     obj1.set_date(20,10,2023);
     obj1.display( );
     obj2.set_date(23,10,2023);
     obj2.display( );
     return 0;
}
```

Stack frames in RAM on function call

obj1.set_date(20,10,2023);

stack frame of set_data( )

stack frame of main( )



```
void set_date(int dd, int mm, int  yy)
   {
       this->dd = dd;
       this->mm = mm;
       this->yy = yy;
   }
```

Access Specifiers in C++

Private :

- We can access the private members from within the class only.
- We can not access the private members from outside the class.
- We keep data members as private, so that user of class can not modify the data from out side the class.
- We can declare some of the functions as private if needed. These functions will not be accessible from outside the class.

Access Specifiers in C++

Public :

- We keep member functions as public, because we want that the user of the class to be able to call functions using object of the class.

- Public means we can access the members from within the class as well from outside class.

- Public is used to provide interface of class. User of the class can talk with the object by calling public functions.

setter and getter methods

Data is kept private in the class. To access or modify the data members we write methods for all possible member variables. These methods are getters and setters. We can implement all possible data validations in these methods.

- **setter methods are used to set values of member variables.**

- **getter methods are used to get values of member variables.**

Setters and getters in date class

```
void setDay(int d)
 {
   if( d <= 31)  dd = d;
   else cout<<"\n Invalid day.. can not be set as value of dd...";
 }

 void setMonth(int m)
 {
   if( m <= 12)  mm = m;
   else cout<<"\n Invalid month.. can not be set as value of mm...";
 }

 void setYear(int y)
 {
   yy = y;
 }
```

```
int getDay()
 {
   return dd;
 }

 int getMonth()
 {
   return mm;
 }

 int getYear()
 {
   return yy;
 }
```

Assignment:

Add getters and setters in **time** and **point** class, and call these methods from main( ).

Constructor

- It is a public member function of a class
- Name of constructor function is same as name of class
- It is called automatically when object of the class created / object is born
- We can overload constructors – default constructor and parameterized constructor
- Constructors are used to initialize values of data members of class and we can allocate resources for objects in constructor. (resources means memory / any device)

- Constructors do not have any return data type, not even void
- If no constructor is written in class definition, compiler provides default constructor.
- If user is writing any constructor, compiler will not provide its default copy.

Destructor

- It is a member function of a class
- Name of destructor is ~followed by name of class
- It is called automatically when object goes out of scope / object dies
- We cannot overload destructor. It is one per class
- We release resource allocated in constructor.
- Destructor do not have any return data type.
- If no destructor is written, compiler will provide default destructor.

Assignment: Add constructor and destructor for date, time and point classes.

Assignment: Declare a class student having

data members : int rno, string name, int mk1, mk2, mk3, total, char grade
member functions: constructor, parameterized constructor, destructor, display, getters and setters, calculate_grade

Operator overloading

Need to overload operator-

- Operator Overloading is a feature of C++ because of which additional meanings can be given to existing operators for user-defined datatypes.
- **operator is keyword in C++ which is used to implement** operator overloading.
- Operator overloading feature makes user defined data type (class) more natural & closer to built in data types.

Rules of Overloading Operator

You cannot create new operators, only can overload existing ones.

precedence or associatively of an operator can not be changed

Meaning of operator should be similar as for built-in data types

ways to overload operators in C++

There are various ways to overload Operators in C++ by implementing any of the following types of functions:

1) Member Function

2) Non-Member Function or **Friend Function**

List of operators that cannot be overloaded

1) Scope Resolution Operator (::)

2) Ternary or Conditional Operator (?:)

3) Member Access or Dot operator (.)

4) Pointer-to-member Operator (.*)

5) Object size Operator (sizeof)

6) Object type Operator(typeid)

7) static_cast (casting operator)

8) const_cast (casting operator)

9) reinterpret_cast (casting operator)

10) dynamic_cast (casting operator)

Copy constructor

- When an object is created using another object, copy constructor is called implicitly.
- Compiler provides default copy constructor in case the programmer is not providing.
- Default constructor does bit by bit copy of one object to another.
- It is called when we are passing object of any class as call by value parameter
- It is also called when any function is returning object
- We can create copy of object at the time of declaration. Syntax is

```
date obj2(obj1)
```

Here date is the name of class, obj2 is the new object to be created and it will be copy of obj1

Copy constructor

- When an object is created using another object, copy constructor is called implicitly.
- Compiler provides default copy constructor in case the programmer is not providing.
- Default constructor does bit by bit copy of one object to another.
- It is called when we are passing object of any class as call by value parameter
- It is also called when any function is returning object
- We can create copy of object at the time of declaration. Syntax is

  date obj2(obj1)

  Here date is the name of class, obj2 is the new object to be created and it will be copy of obj1

```
class myArray{
  int size;
  int *ptr;

  public:

     myArray( ) {
        size = 5;
        ptr = new int[size];
        for(int i=0 ;i<size ; i++)
           ptr[i] = 0;
     }

    ~myArray( ) {
        delete []ptr;
    }
};
```

Scenario 1:

```
int main( ) {
   myArray obj1;
   return 0;
}
```

Scenario 2:

```
int main( ) {
   myArray obj1;
   myArray obj2;
   return 0;
}
```

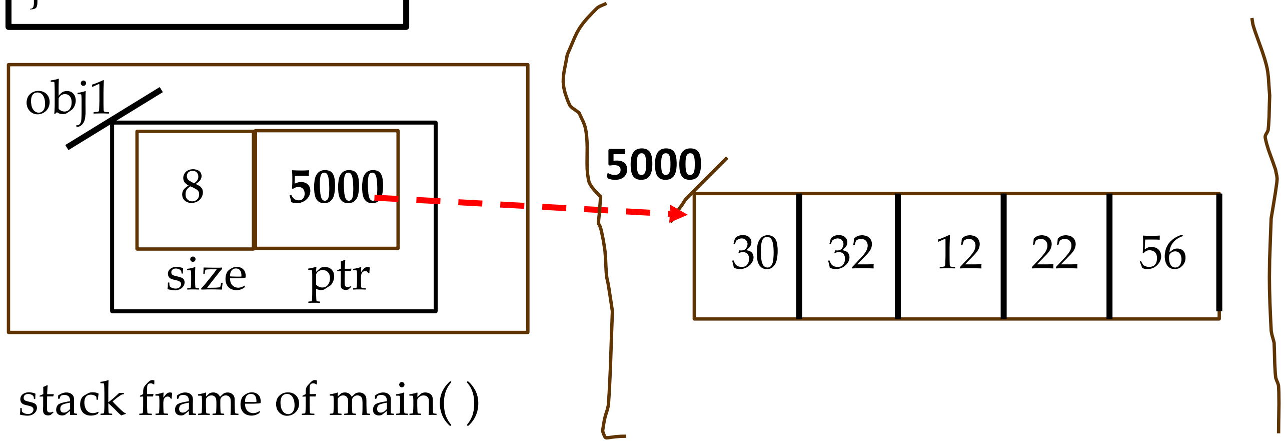
Scenario 3:

```
int main( ) {
   myArray obj1;
   myArray obj2(obj1);
   return 0;
}
```

```
class myArray{
 int size;
 int *ptr;

 public:

     myArray( ) {
         size = 5;
         ptr = new int[size];
         for(int i=0 ;i<size ; i++)
             ptr[i] = 0;
     }

   ~myArray( ) {
        delete []ptr;
    }
};
```

Scenario 1:

```
int main( ) {
    myArray obj1;
    return 0;
}
```



stack frame of main( )

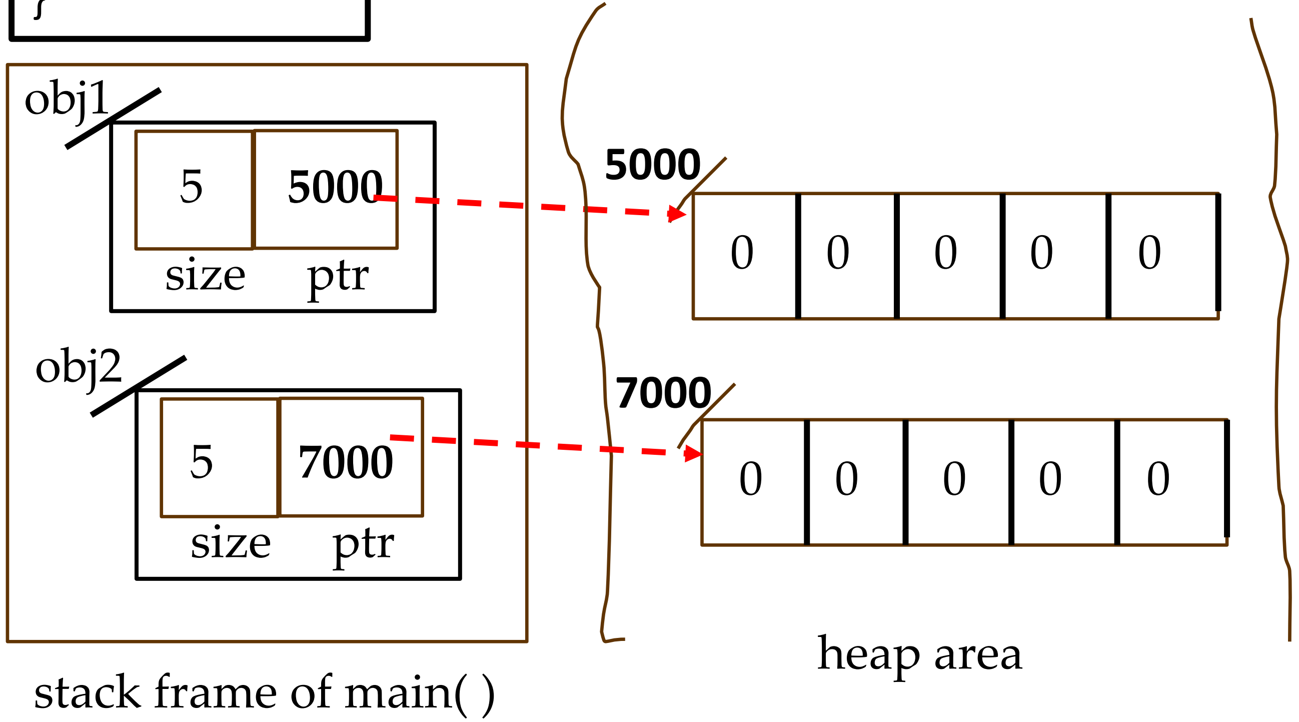
heap area

heap area

```
class myArray{
 int size;
 int *ptr;

 public:

    myArray( ) {
        size = 5;
        ptr = new int[size];
        for(int i=0 ;i<size ; i++)
            ptr[i] = 0;
    }

  ~myArray( ) {
        delete []ptr;
    }
};
```

Scenario 2:

```
int main( ) {
    myArray obj1;
    myArray obj2;
    return 0;
}
```



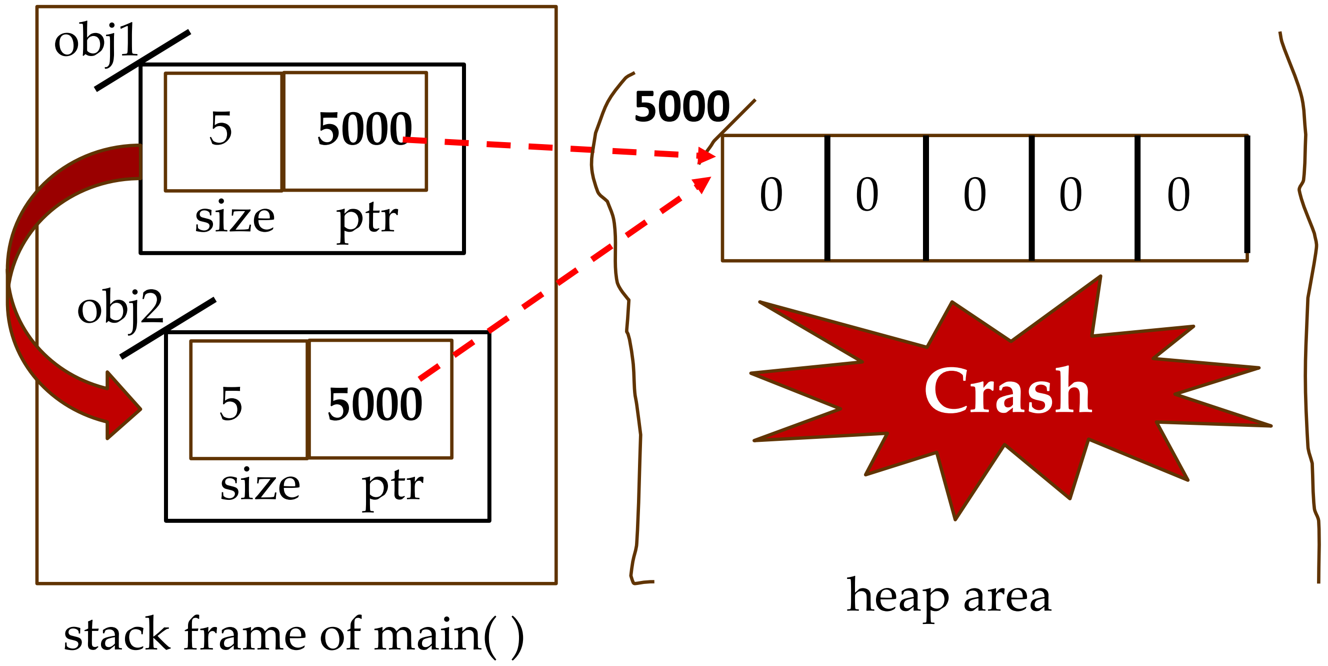
heap area

Shallow copy
Bit by bit copy

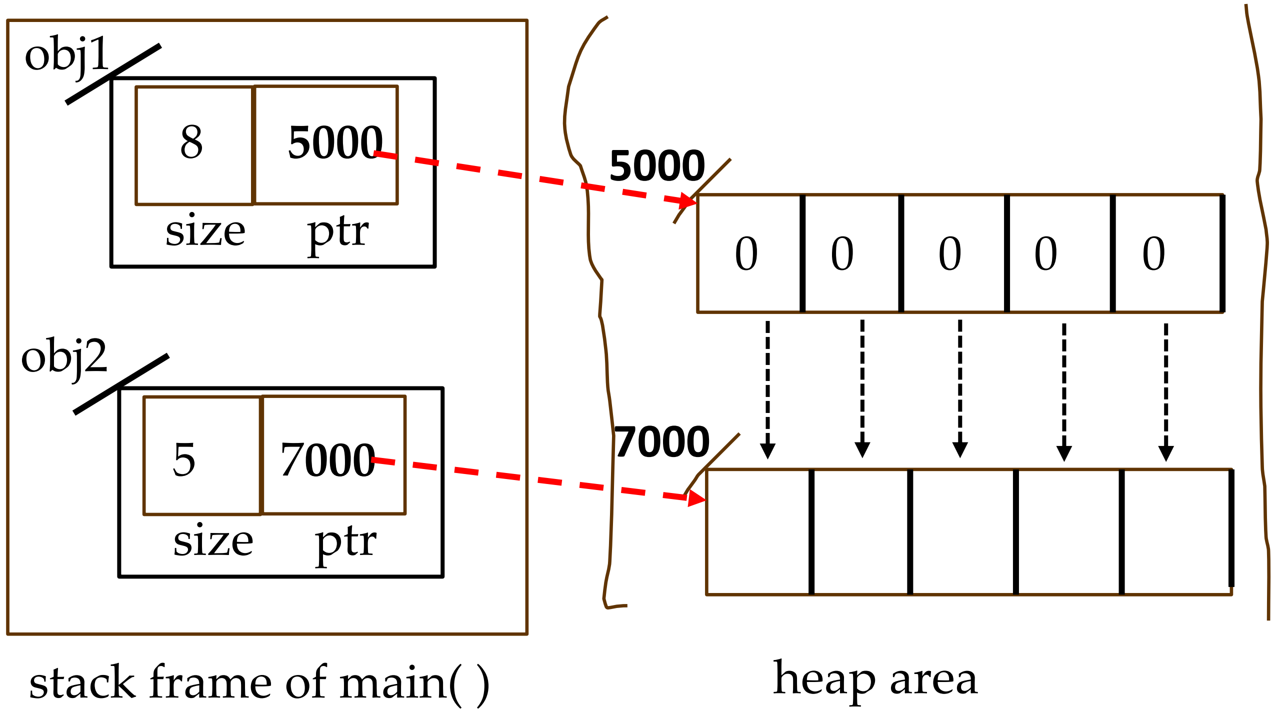
```
class myArray{
 int size;
 int *ptr;

 public:

    myArray( ) {
        size = 5;
        ptr = new int[size];
        for(int i=0 ;i<size ; i++)
            ptr[i] = 0;
    }

   ~myArray( ) {
        delete []ptr;
   }
};
```

Scenario 3:

```
int main( ) {
    myArray obj1;
    myArray obj2(obj1);
    return 0;
}
```



Deep copy by copy constructor



```
myArray(myArray &m ) {
      size = m.size;
      ptr = new int[size];
      for(int i=0 ;i<size ; i++)
         ptr[i] = m.ptr[i];
   }
```

- *Copy constructor has to copy data on stack from source object to target object*
- *Copy constructor has to allocate new memory for pointer variables in target object, then copy contents on heap area from source object to target object.*

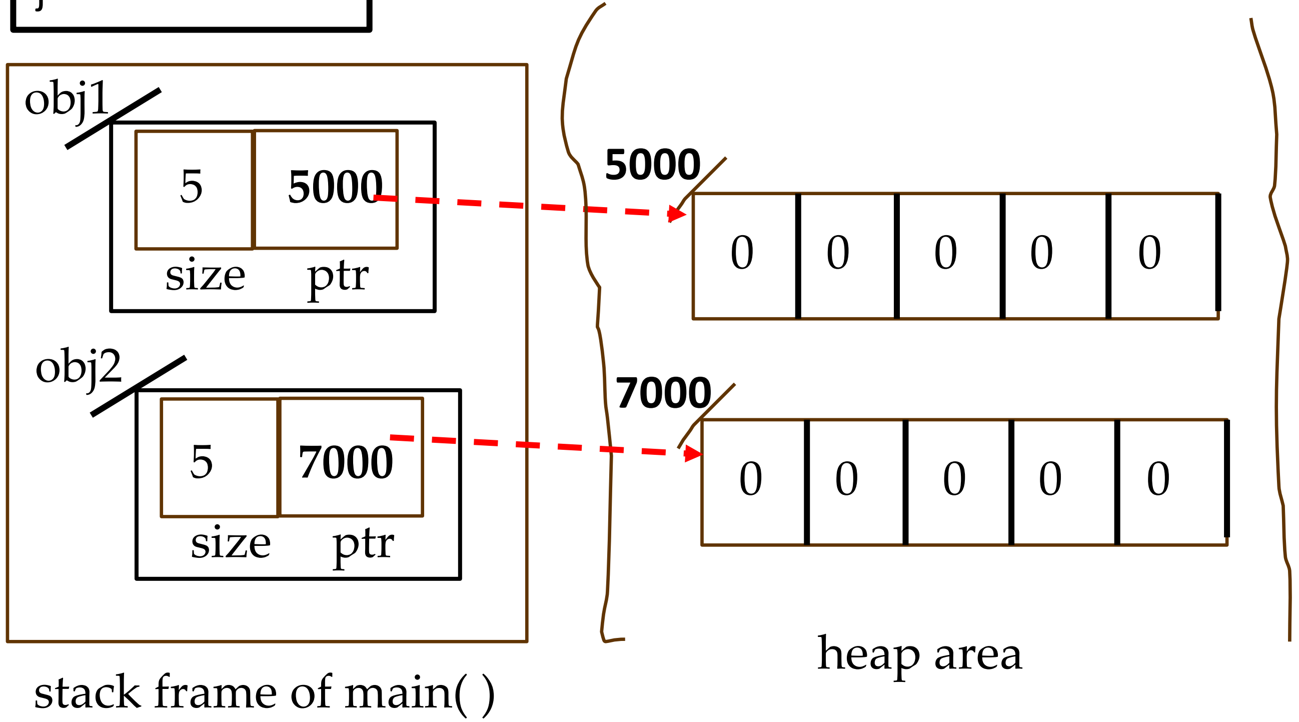
Assignment operator

- When an object is assigned value of another object, assignment operator functions is called implicitly.
- Compiler provides default assignment operator implementation (=) in case the programmer is not providing.
- Default assignment operator function does bit by bit copy of one object to another.
- obj2 = obj1, when obj1 and obj2 are already declared objects of any class.

Consider scenario

```
class myArray{
 int size;
 int *ptr;

 public:

     myArray( ) {
        size = 5;
        ptr = new int[size];
        for(int i=0 ;i<size ; i++)
            ptr[i] = 0;
    }

   ~myArray( ) {
       delete []ptr;
   }
};
```

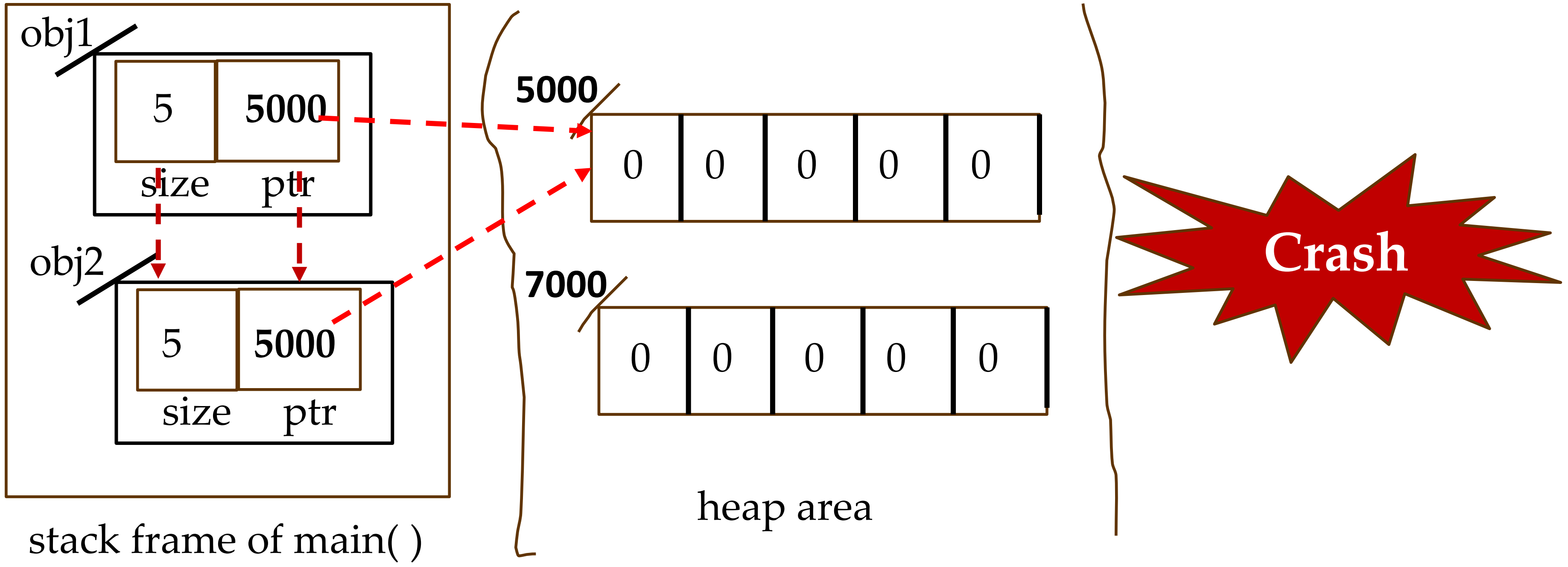
```
int main( ) {
   myArray obj1;
   myArray obj2;
   return 0;
}
```



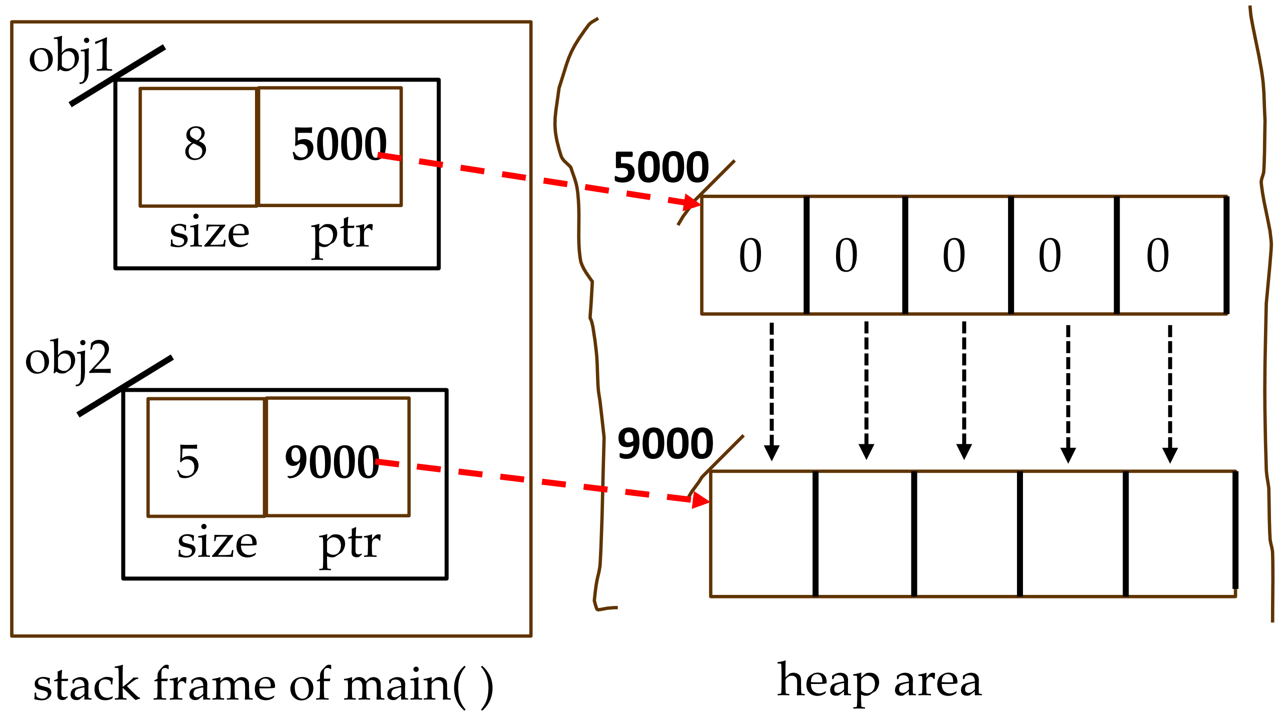
heap area

Default behaviour of assignment operator

```
int main( ) {
   myArray obj1;
   myArray obj2;

   obj2 = obj1; //call to assignment operator
   return 0;
}
```



Assignment Operator



```
myArray operator = (myArray &m ) {
        delete [ ] ptr;
        size = m.size;
        ptr = new int[size];
        for(int i=0 ;i<size ; i++)
            ptr[i] = m.ptr[i];
    }
```

- *Assignment operator has to release memory in hold for ptr of target object.*
- *Assignment operator has to allocate new memory for ptr of target object.*
- *Assignment operator has to copy values of source object to target object.*

Assignment operator

- Some compiler make it compulsory to define signature of assignment operator as

  const &myArray operator = (const myArray &m) { ……. }

  This means the function returns reference to constant and takes constant reference as parameter.