```
Student
{
}
Date
{
}
```

```
using namespace NStudent;
int main(){
Student s1;
s1.init();

Date d1;
d1.init();
Date d2;
d2.init();
}
```

Types Of Member Functions

1. Constructor
2. Destructor
3. Mutator
4. Inspector
5. Facilitator

```
class Time{
int hr;
int min;

void accept();
void display();

}

Time t1;
t1.init();
Time t2;
t2.init();
t1.hr = 12;
t1.setHr(12);
cout<<t1.hr<<endl;
cout<<t1.getHr()<<endl;
```

Constructor
- It is special Member Function of the class
- Why so special-
    1. Its name of ctor is same as that of class name
    2. It do not have any return type
    3. It gets automatically called when object is created

Types of Constructor
1. Default/Parameterless Ctor
2. Paramterzied Ctor
3. Copy ctor
    - We will see after the reference topic

```
Point p1; // Parameterless Ctor
Point (2,3); // Parameterized Ctor
```

Ctor Delegation
- This feature is available from C++ 11 Version
- To call another ctor from one ctor is called as ctor delegation
- This is used to avoid writing redundent code.

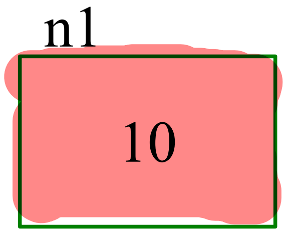
Destructor
- It is special Member Function of the class
- Why so special-
    1. Its name of Dtor is same as that of class name with tild(~) symbol
    2. It do not have any return type
    3. It gets automatically called when object goes out of scope
- Dtor calling sequence is opposite to the ctor calling sequence

```
const int n1 = 10;
n1 = 20;// NOT OK
```

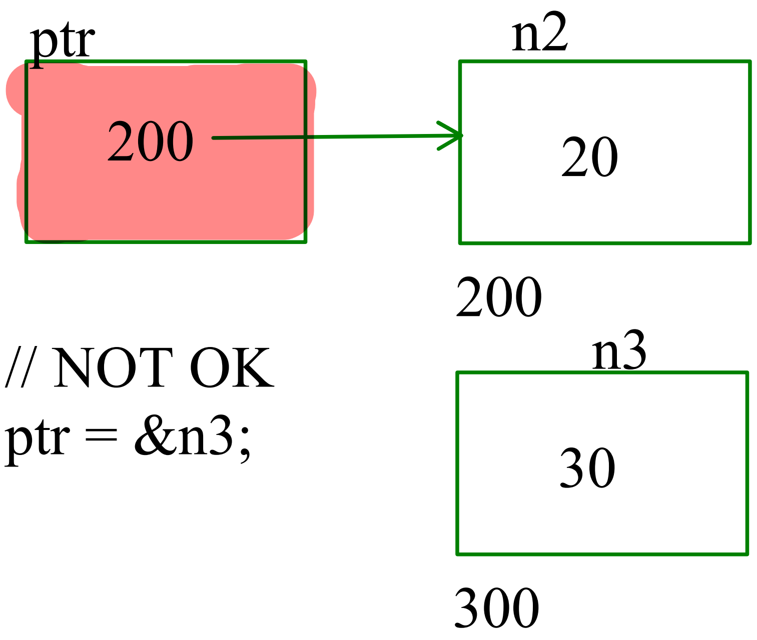
```
int n2 = 20;
int n3 = 30;
int *const ptr = &n2;
*ptr = 200;
ptr = &n3; // NOT OK
```

```
const int n1 = 10;
const int *ptr = &n1;
*ptr = 20; // NOT OK
```

```
int const n1 =10;
```

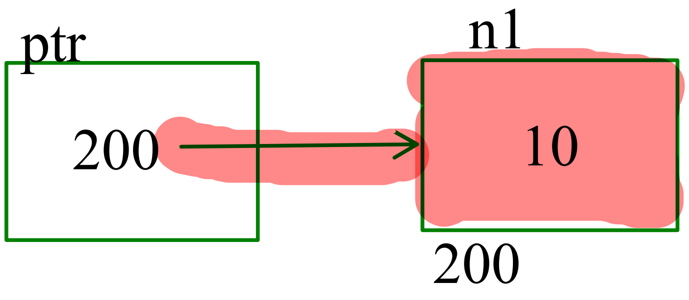


```
int n2 = 20;
int n3 = 30;
int *const ptr = &n2;
```



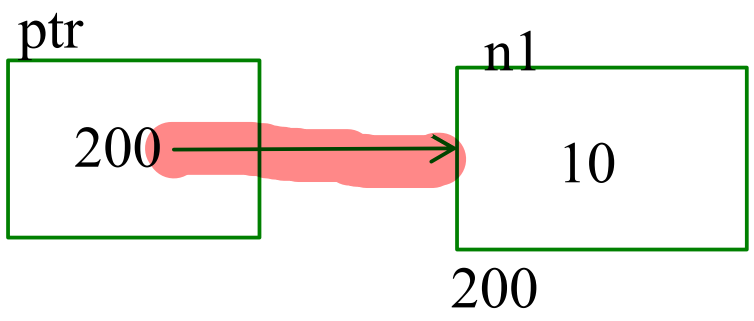
```
// NOT OK
ptr = &n3;
```

```
const int n1 = 10;
const int *ptr = &n1;
```



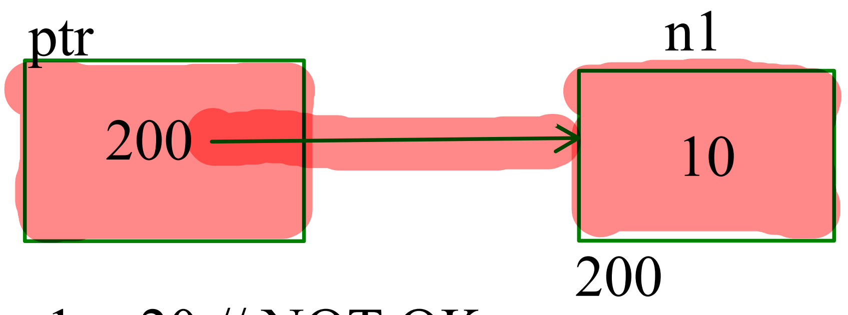
```
n1 = 20; // NOT OK
*ptr = 20;// NOT OK
```

```
int n1 = 10;
const int *ptr = &n1;
```



```
n1 = 20; // OK
*ptr = 30; // NOT OK
ptr = &n2;// OK
```

```
const int n1 = 10;
const int *const ptr = &n1;
```



```
n1 = 20;// NOT OK
*ptr = 30;// NOT OK
ptr = &n2; // NOT OK
```

In C++ we can make
1. Data memebrs as constant
2. Member functions as constant
3. Object as constant

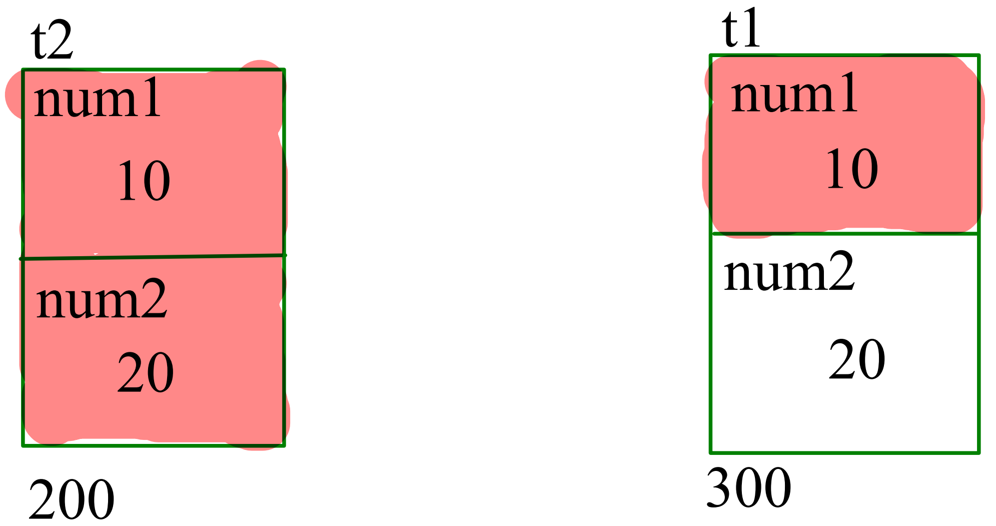
```
class Time{
int hr;
int min;

void accept() // Time *const this
{
Time t1;
this = &t1;
}

}
```

```
const int n1 = 10;
int n2 = 20;
int *const ptr = &n2;
const int *ptr2 = &n2;
```

database_name
connection

# Constant Data member
- We can make the data members as constant
- Constant data memebers must be initialized inside ctor members initializer list

# Constant Member Functions
- memeber functions that do not modifiy the state of the current calling object should be made as constant
- Once the functions are constant we cannot change the value of non constant data members inside them
- We can make the display or all the Inspector functions as constant



```
Test t1;
t1.num2 =  40;
const Test t2;
t2.
```

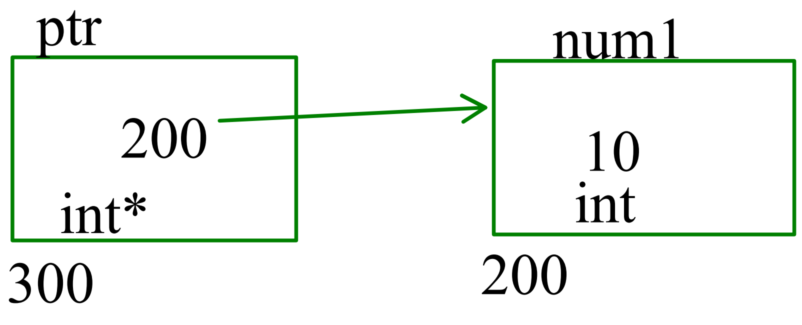
# Constant Object
- In C++ we can also make Object as constant.
- Object as constant means the state once initialized cannot be changed
- on constant objects we can call only constant member functions.
- we cannot call non constant member functions on constant object

```
BankAccount a1();
```

```
void changenum1(int *ptr){
ptr = 20;
}

int main(){
int num1=10;
 changenum1(&num1);
}
```

Stack

ptr-> 200
&ptr-> 300
*ptr -> 10

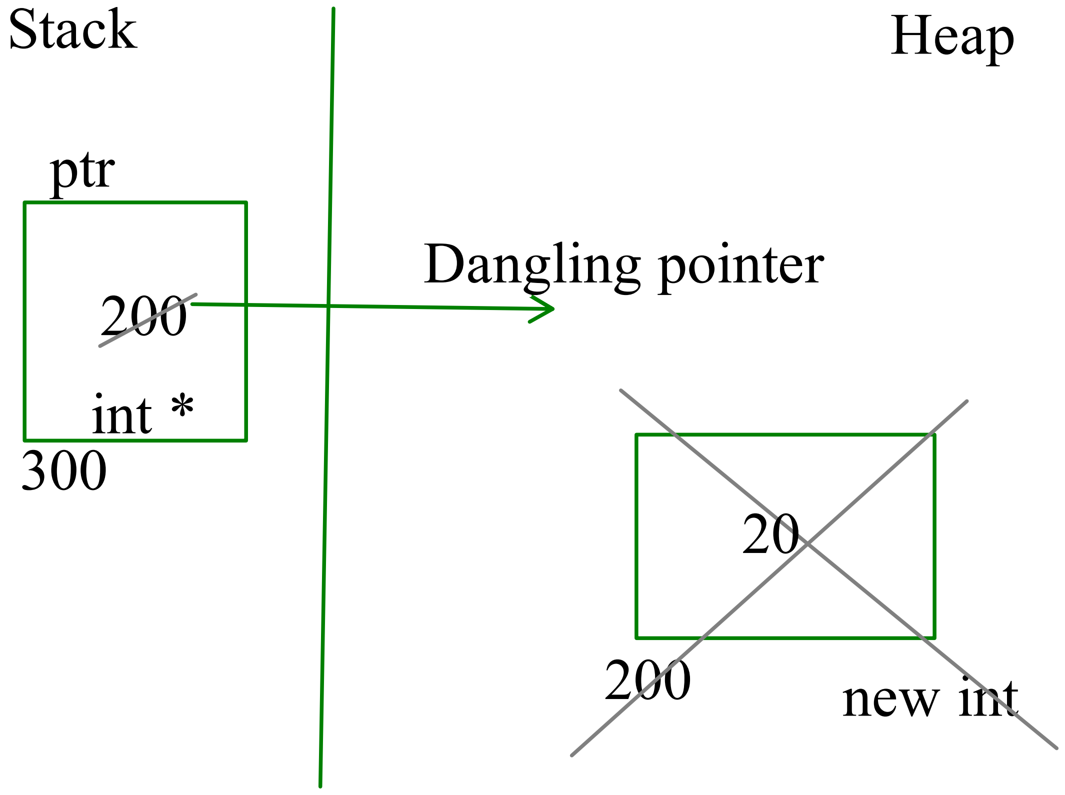


num1 -> 10
&num1 ->200

- Need
    -
  new

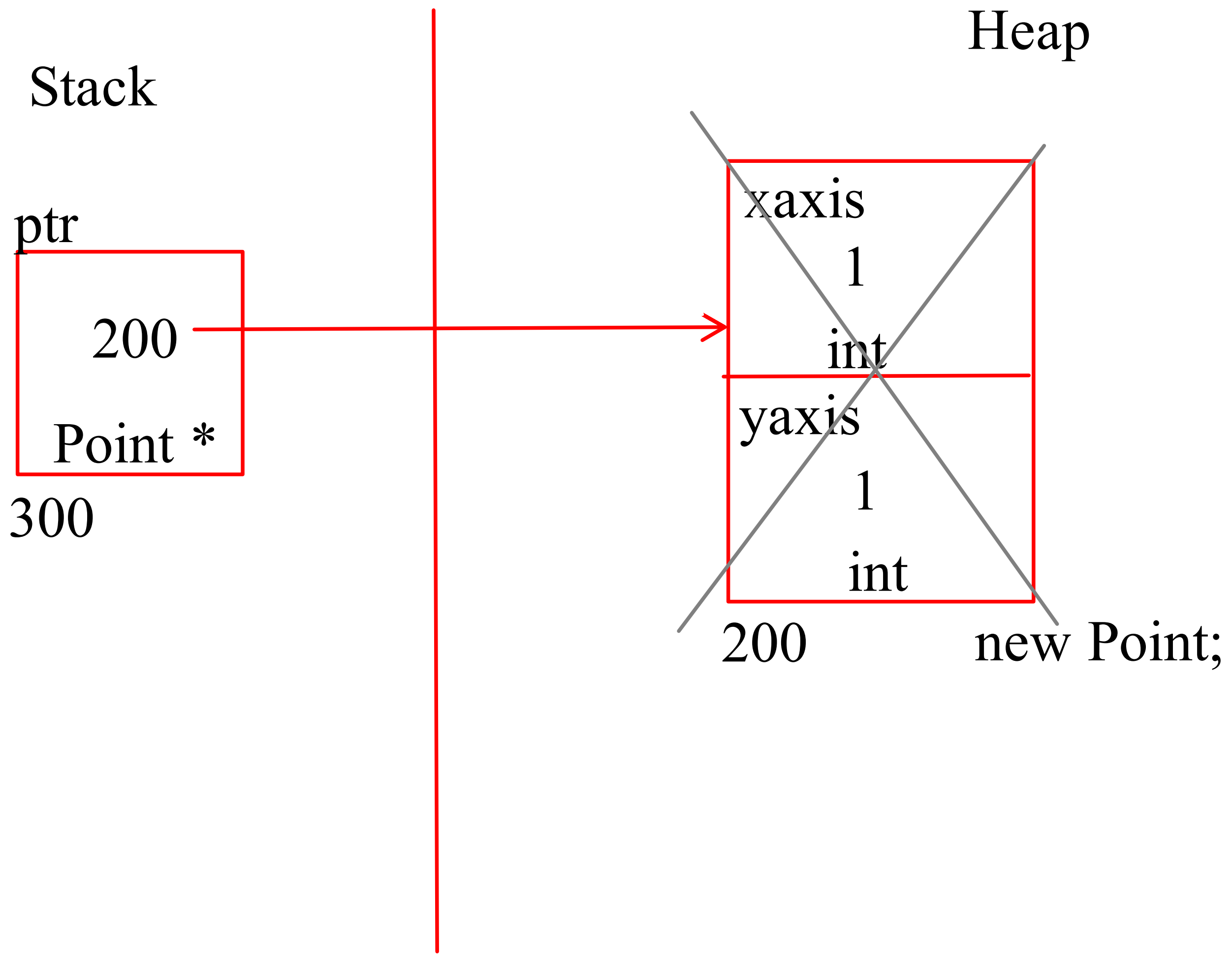
```
f1(){
int num1;
int num2;
int num3;
if(condition)
        num1
else if(condition)
        num2;
else
        num3;
}

int main(){
f1();
return 0;
}
```

```
calculateArea(int choice){
Circle c;
Rectangle r;
Square sq;

if(choice ==1)
        c.calculate()
else if(choice ==2)
else
}
```

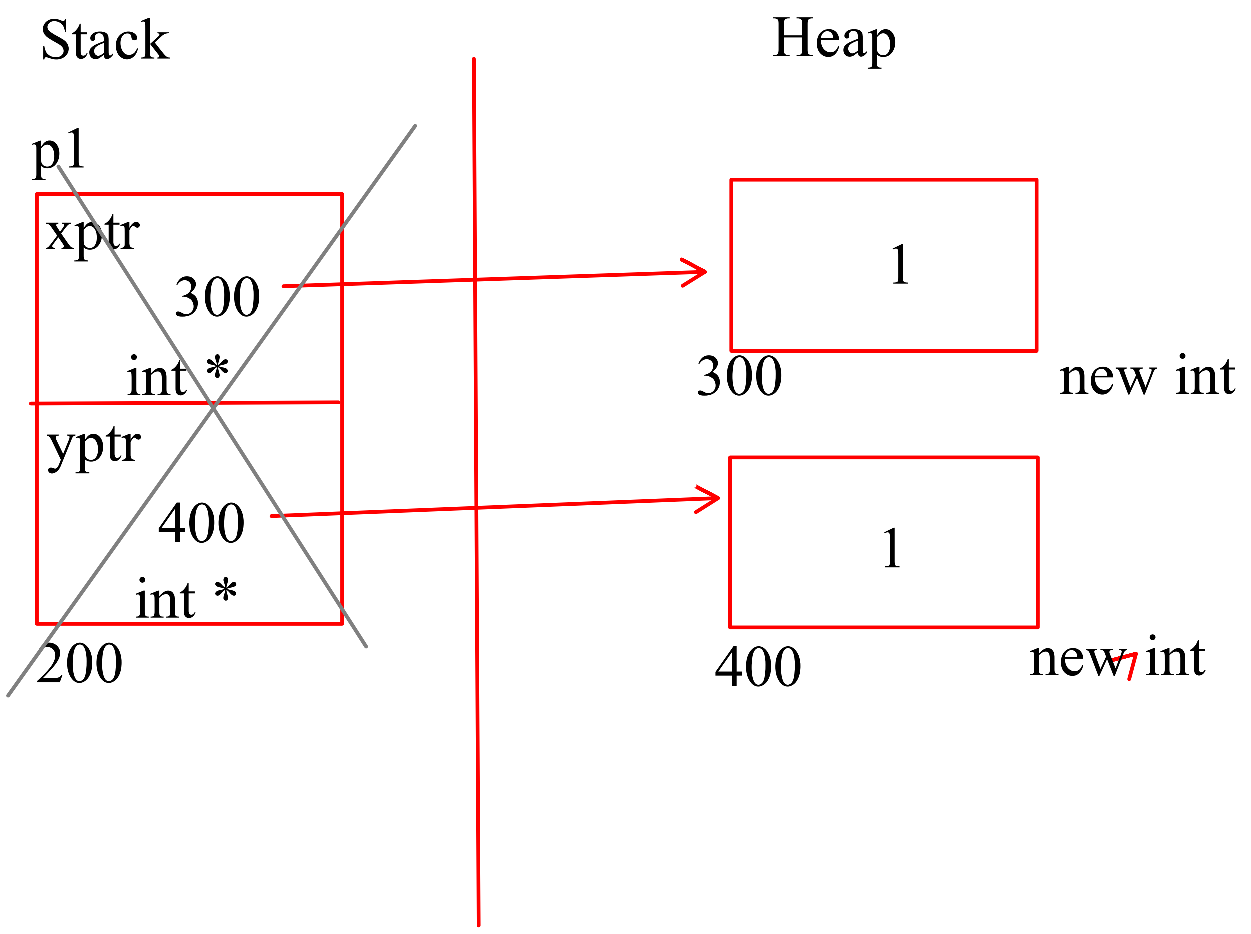


```
*ptr -> 0
*ptr = 10;
cout<<*ptr;
```

```
delete 200;
delete ptr;
ptr =NULL; // To avoid the dangling pointer
```



```
delete 200;
delete ptr;
```



Stack

Heap

p1

xptr

300

int *

yptr

400

int *

200

1

300

new int

1

400

new int

1,1

```
*xptr = 1;
*yptr = 1;
```

```
delete xptr;
delete yptr;
```

When dynamic memory is allocated it should be deallocated by the programmer itself.
If the dynamic memeory is allocated witin the class it should be deallocated inside the Dtor
If the dynamic memory allocation is done outside the class it should be deleted outside the class

```
case:
{
Volume v1
}
```

```
Toolbooth t1;
t1.
```