set
=
get
return

t.hrs = 10;
t.setHrs(10);

Types of Member Functions
1. Constructor
2. Destructor
3. Mutator
4. Inspector
5. Facilitator

Constructor
- To initialize the object with the default values
- It is a special member function of the class
    - The name of ctor is same as that of class name
    - It does not have any return type
    - It gets automatically called when object is created.
- Types of Ctors
1. Default/Parameterless Ctor
2. Parameterized Ctor
3. Copy Ctor

Constant

```
const int num = 10;
int const num = 10;
```

```
const int num;
num = 10;
// NOT OK
```

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

int num2 = 100;
ptr = &num2;
```

```
const int num1 = 10;
const int *const ptr = &num1;
int num2 = 20;
//ptr = &num2; // NOT OK
```

int const num; // variable is constant
int *const ptr; // pointer becomes constant

const int *const ptr; // constant pointer pointing to the constant variable

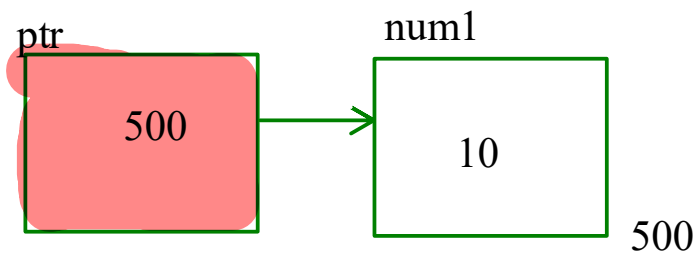
In C++ we can make
1. Variable as a constant
2. Data memebrs as constant
3. Member Functions as constant
4. Object as constant

```
const int num1 = 10;
```

num1
10

num1=20;//NOT OK

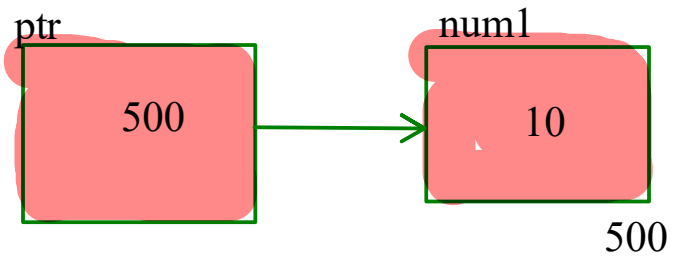
```
int num1 = 10;
int num2 = 20;
int *const ptr = &num1;
```



ptr=&num2;
// NOT OK

num2
20
600

```
const int num1 = 10;
const int num2 = 20;
const int *const ptr = &num1;
```



&ptr = &num2; // NOT OK
num1 = 40; // NOT OK

```
int num1 = 10;
int num2 = 20;
const int *ptr = &num1;
```


num1 = 40; // OK
*ptr = 40;// NOT OK

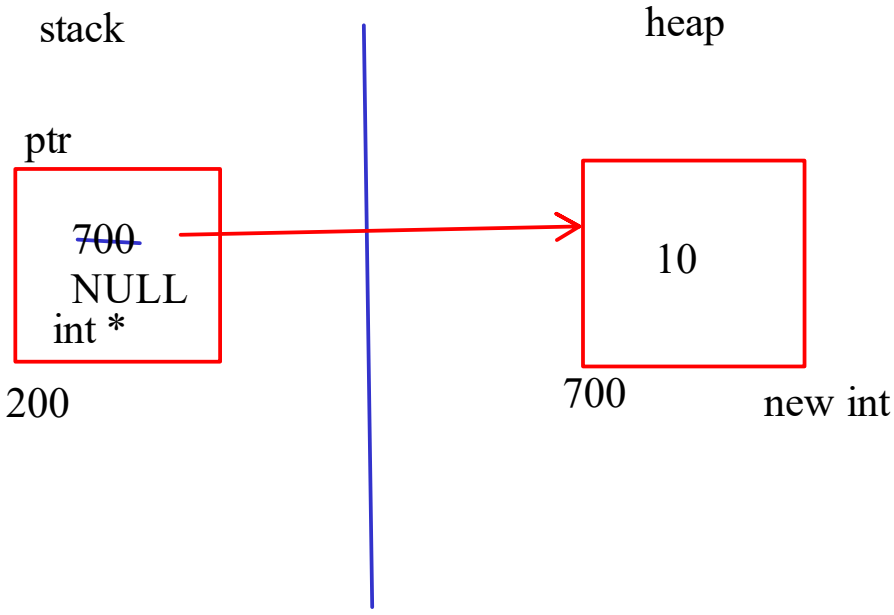
ptr = &num2;

```
const int num1 = 10;
int *ptr = &num1; // NOT OK
```

```
void addTime(Time t){
// changes in t should not be allowed in original object
}

        addTime(t1);
```

```
void addTime(const Time *t){
// changes in *t itself is not allowed
// hence their are no changes in original object
}

int main(){
Time t1;
t1.accept();
}

        addTime(&t1);
```

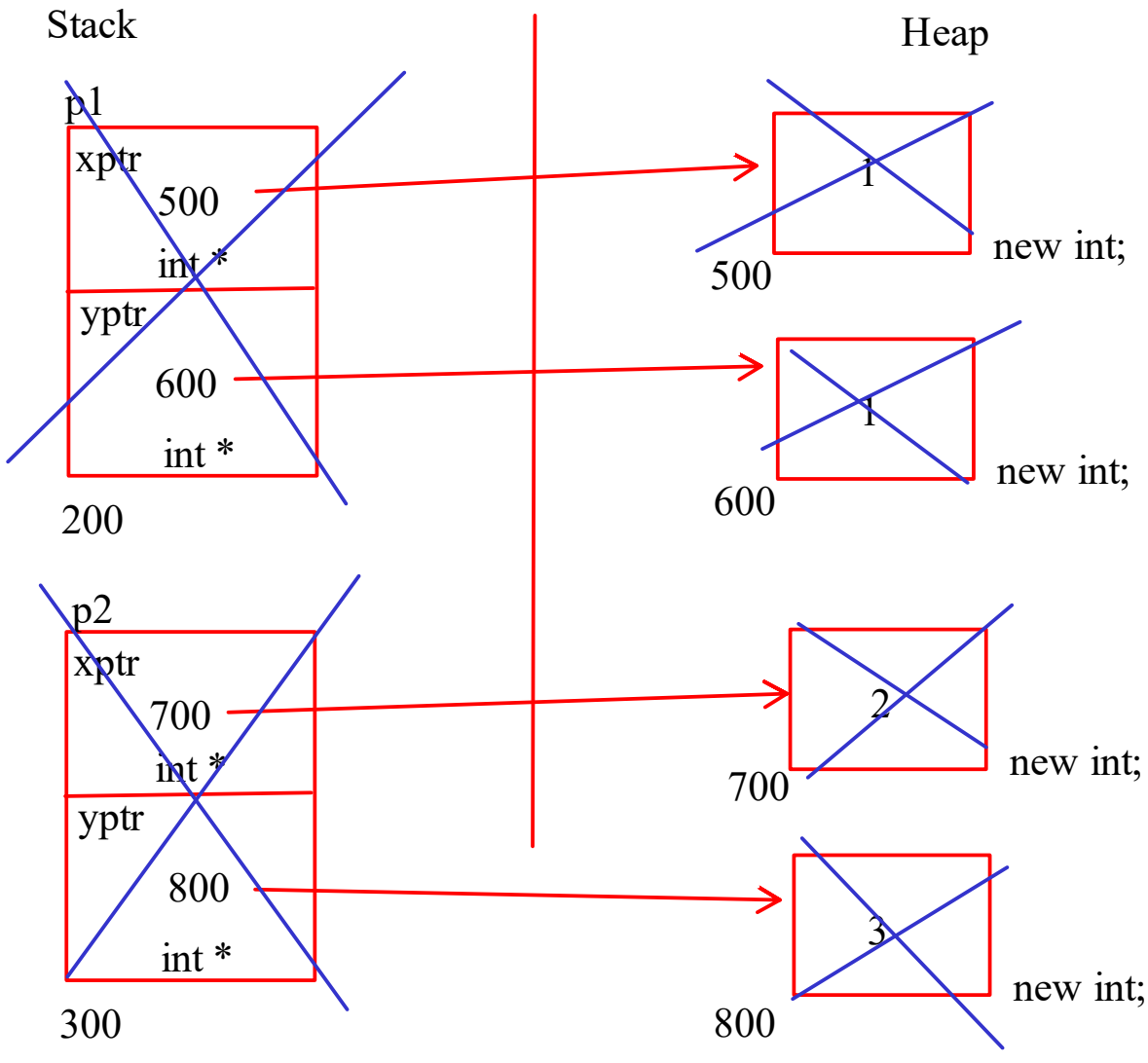
Dynamic Memory Allocation

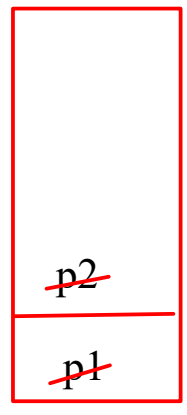



```
int *ptr = new int;
delete ptr;
ptr = NULL;


cout<<ptr; // 700
cout<<&ptr; // 200
cout<<*ptr; // 10
```

Destructor

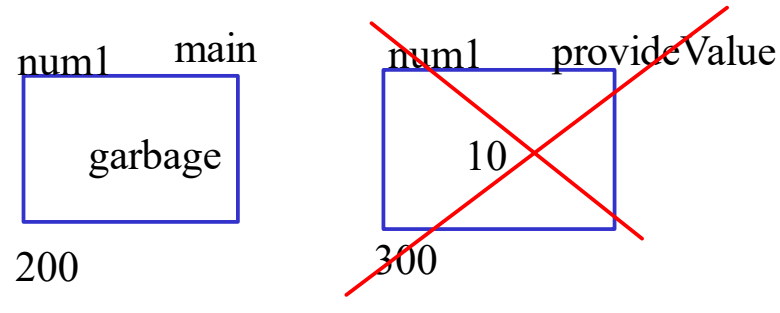
- It is a special member function of the class
    1. Its name is same as that of teh class with tild(~) sign
    2. It does not have any return type
    3. It is automatically called when the object goes out of scope.
- If we do not provide a destructoor inside a class then compiler adds a dtor called as Default Destructor
- No any concept of Dtor Overloading
- Dtor are required if we are allocating the memory dynamically for the class data memebrs or using any resources.
- Dtor calling sequence is exactly opposite to ctor calling sequence.



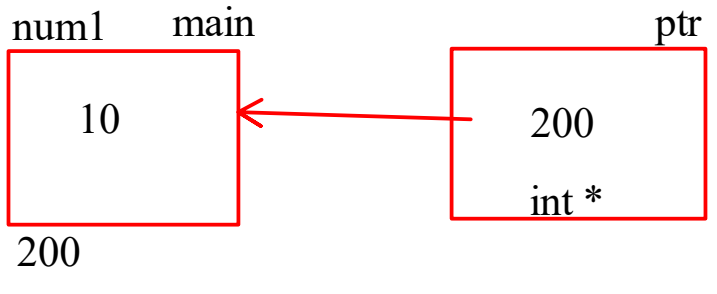


Reference

```
int provideValue(int num1){
        num1=10;
}

int main(){
        int num1;
        provideValue(num1); // pass by value
        cout<<num1<<endl;
return 0;
}
```




```
int provideValue(int *ptr){
        *ptr = 10;
}

int main(){
        int num1;
        provideValue(&num1); // pass by address
        cout<<num1<<endl;
return 0;
}
```




```
int provideValue(int &ref){
        ref=10;
}

int main(){
        int num1;
        provideValue(num1); // pass by reference
        cout<<num1<<endl;
return 0;
}
```