Dt 30/12/2022
Pg

# Module - Pointers

```
int i=10;          700 i [10]
i+=5;                  4bytes
cout<<i<<endl;
```

System has to maintain the address of variables & type

Symbol Table

$i \xrightarrow{int} 700$    700 i $\boxed{10}$

1. Memory Allocation
2. Entry into symbol table at compile time
3. Runtime issues

```
int i=10;
cout<<&i<<endl;  → 700 → hexadecimal (0x)
```

Some variable are used to store the address of other variable called pointers.

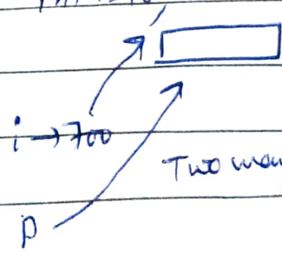
```
pointer p=&i;
int *p=&i;  → syntax
```

pointer is p to an integer

```
cout<<p<<endl;  → 700 ✓  } same.
cout<<&i<<endl; → 700 ✓
```

```
float f=12.32;
float *pf=&f;
double d=1.22;
double *df=&d;
```

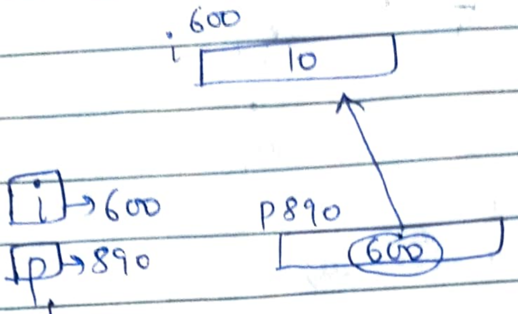
int i=10;



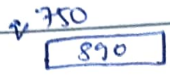
Two ways to reach the same address

To get the value through p → Dereference opr → * → *p →

```
int i=10;
int *p=&i;
cout<<&i; → 70
cout<<p; → 
cout<<*p; →
```

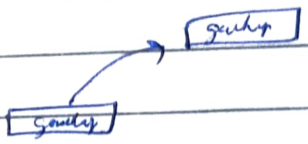


8 bytes for pointer depends on compiler specific



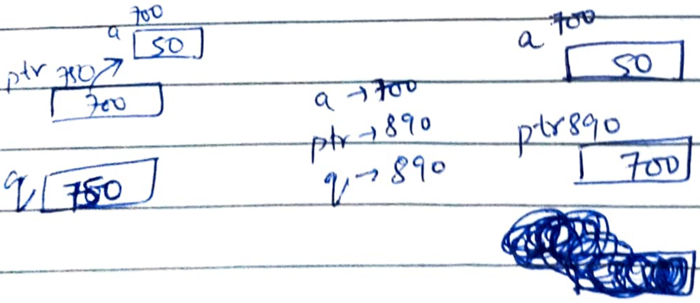
```
int a=i; → *p
a++;
cout<<a<<endl;
i=12;
*p=23;
(*p)++;
int *q=p;
```

```
int *p;
cout<<p<<endl;
cout<<*p<<endl; → memory leak or system may crash.
(*p)++;
```

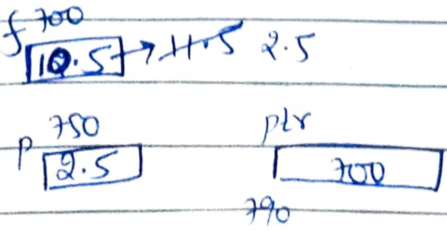


int *p=0; → Null pointer → Segmentation fault

①



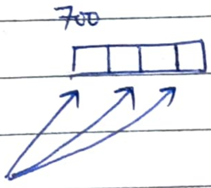
②



Dt________
Pg________

## Pointer Arithmetic

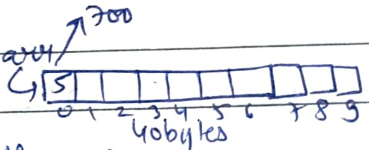
sizeof(p); → 8 bytes only for all datatypes
int *p;
char *p;



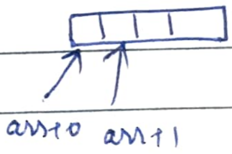
```
int i=10;
int *p= &i;
cout<<p <<endl; → 700
  p=p+1;          → next Integer → After 4bytes
  cout<<p <<endl; → 704
  p=p+2;
  cout<<p <<endl; → 708
  p=p-2;
  cout<<p <<endl;
  p1<p2;
```

## Arrays and Pointers

int arr[10] ;



cout<< arr <<endl; → 700
↓
~~Address of~~ 0th element

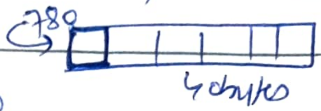


```
cout<< &arr[0] <<endl; → 700
arr[0]=5;
cout<< *arr <<endl;   → 5 → cout<<*(arr+1)<<endl;
int *p=
```

*(arr+1) = arr[1]
*(arr+2) = arr[2]
arr[i]
i[arr]
*(arr+i)

①  sizeof(arr) → 40bytes
int *p; → sizeof(p) → 8bytes

int arr[10];



p → 890
arr → 780
No extra 8 bytes

② & operator

cout << arr << endl;    arr → 790
cout << &arr << endl;

③ Array cannot be assigned.
a ≠ p

p=p+1; — ✓
arr=arr+1; — ✗
p=arr+1; → ✓

# Characters and Pointers → Behave differently

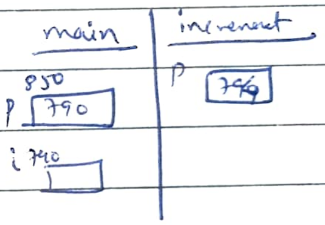
int arr[]={1,2,3};
char arr1[]="abc";
cout << arr <<endl; → 700
cout << arr1<<endl; → abc
char *c = &b[0];
cout << c <<endl; → abc
char c1='a';
char *pc=&c1;
cout <<c1 <<endl;
cout <<*pc <<endl;

```
char  str[] = "abcde"; ✓
char  *ptr = "abcde";  ✗
```

## # Pointers and Functions

① 
```
void print (int *p){
        cout<<*p<<endl; }  ⟶ 10
int main(){
    int i=10;
    int&p=&i;
    print(p);
}
```

② 
```
void increment(int *p){
        p=p+1; }
int main(){ int i=10; int *p=&i;
    cout<< p<<endl;
    increment(p); cout<<p <<endl;
}
```



Same value as it is passby value function.

③ 
```
                  *arr → sizeof(arr) → 8bytes
int sum(int arr[], int size){  int sum=0;
    for(int i=0;i<size;i++){
            sum+=arr[i]; }
    return sum;
}
int main(){
    int arr[10];
    cout<<sum(arr,10)<<endl;
    }
```

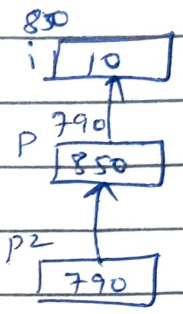
④ Can also pass part of array → f(arr+3)

# Double Pointer

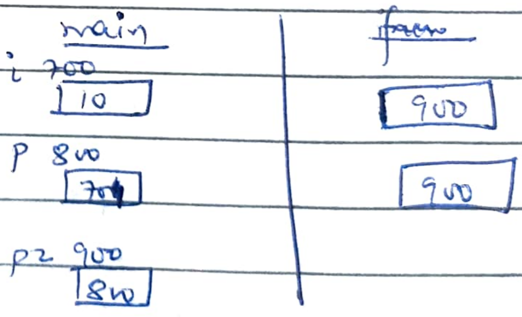
how to intepret → how much byte to read

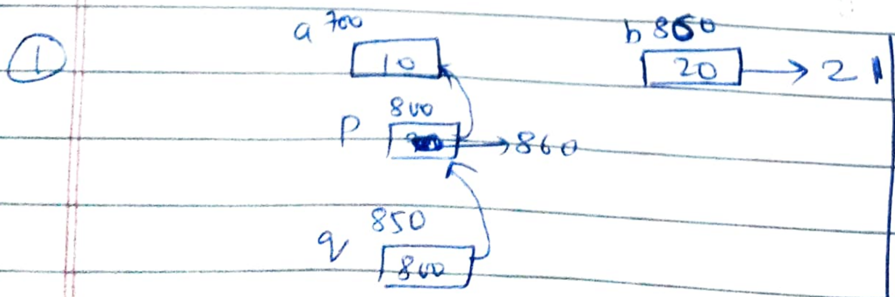
```
int *p= &i;
pointer p=&i;
*p

{
int i=10;
int *p=&i;
int**p2=&p;
cout<<p2<<endl; → 790
cout<<&p<<endl; → 790
cout<<*p2 <<endl; → 850
cout<<p<<endl; → 850
cout<<&i<<endl; → 850
cout<<i<<endl;
cout<<*p<<" "<<**p2<<endl; }
```

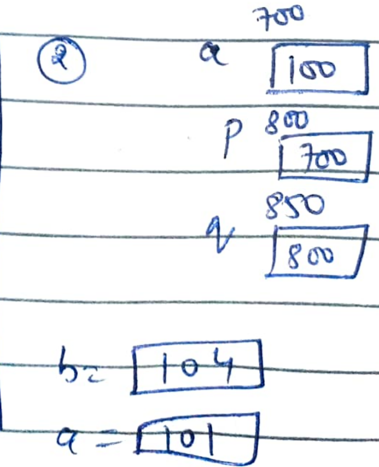
i → 850
p → 790
~~p2→890~~



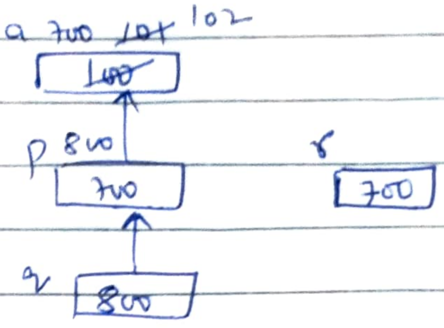
```
void increment(int **p){
        p=p+1; }   ✗
void increment(int **p){
        *p=*p+1; } → ✓
void increment(int **p){
        **p=**p+1; }
```







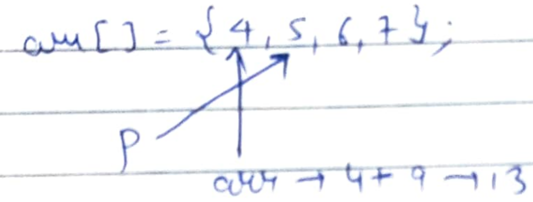
(3)



b = 100

# Void Pointer → Generic pointer → can store data of any type

(1)



(2)

| 10 | 30 | 20 | 40 | 50 |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

(3)

str[] = "abcdefg";

ptr = ptr + 5; → fg

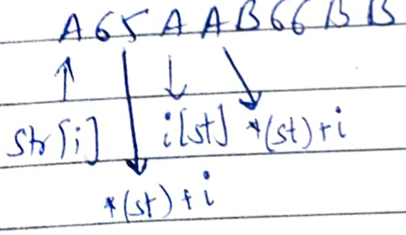
(4) ABCDEFGHIJ

(5) ninjasquiz codingninjas

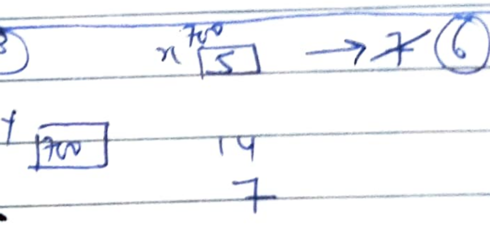
codingninjas ninjasquiz

(6) 90.5 3

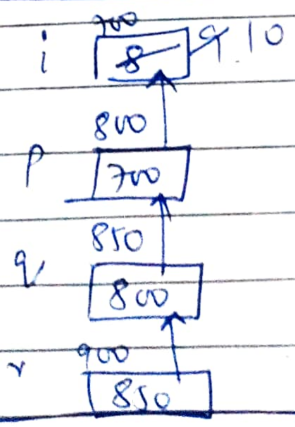
(7)



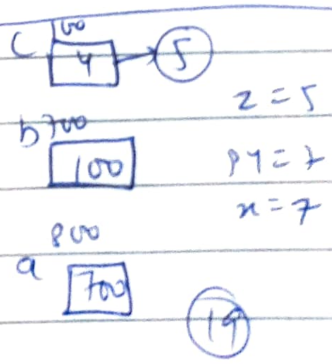
(8) (6)



(9)



(10)



END OF LECTURE

WOW!