KHUSHBOO SHAKYA

Pointers

Pointer is address of another variable. (a variable which stores)

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

p is a pointer to integer

p = address

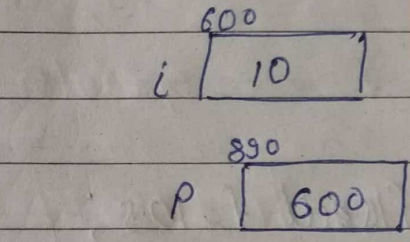
*p = value

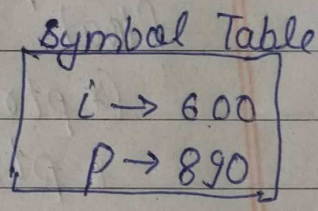
dereference operator

gives me the value pointed by p

Example

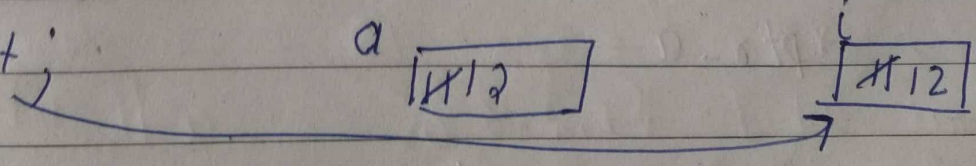
```
int i=10
int* p=&i;

cout<<i;    // O/P 10
cout<<*p;   // O/P 10
i++;
cout<<i;    // O/P 11
cout<<*p;   // O/P 11
```






```
int a = *p;
int a = i;   } will not affect   // a=12, i=11
a++;
```

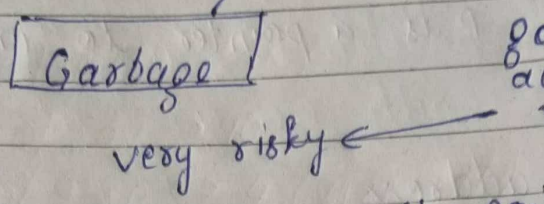
```
(*p)++;
```




```
int *q = p;
```

→ q is now also pointing i

```
int *p;
cout << p << endl;   → garbage address
cout << *p << endl;
```



very risky ←

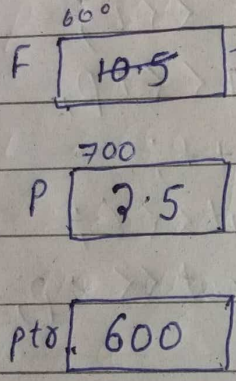
So, we will initilize a null pointer

`int *p = 0` → Not pointing to anything
↓ Null ptr

cout << p << endl → will give segment fault error

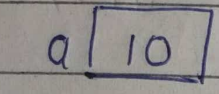
↓
aisi mem access kr rhe ho jo apki hai

Q
```
float f = 10.5;
float p = 2.5;
float *ptr = &f
(*ptr)++;
*ptr = p
```



O/P — *ptr = 2.5, f = 2.5, p = 2.5

Q
```
int *ptr = 0
a = 10
*ptr = a
```




```
cout << *ptr << endl
```

O/P — 0

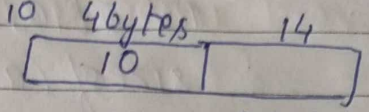
Note :- Pointer size can be varry. If 32 bit computer the pointer size can be 4 bytes for a 64 bit computer the pointer size can be 8 byte

32 bit computer → 4 byte pointer size
64 bit compute → 8 byte pointer size

Pointer Arithmetic

Any type of pointer is of size 4 byte.

```
int i = 10;
int *p = &i;
```




```
cout << p << endl; — 10
p = p+1;
cout<<p << endl; — 14
```

next four bytes

p++; next intege
--p; previous inte

usage in array

- Otherwise we don't know what is stored at prev or next p location
- we can also do comparison $p_1 < p_2$

Arrays and Pointers

```
int a[10];
cout << a;      → address of a[0]
cout << &a[0] → same ↑
```

a is behaving like a pointer

```
a[0] = 5;
cout << *a; → 5
a[1] = 10;
cout << *(a+1); → 10
```

*(a+i) → a[i]

*(a+1): dereferencing; address of 1st element

i[a] ⟶ *(i+a)

This is also true because compiler do this.

→ But there is a diff b/w 'a' and *ptr

1) Sizeof :-

int a[10];

sizeof(a) → 40

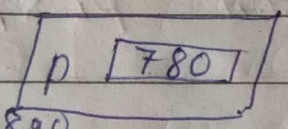
- But int *p → 4 (size of)

The basic diff is for p there is diff mem. of 4 bytes for storing address. but for a there is no such thing.

cout << a << endl; ⟶ 780 (In symbol table)

2) & operator :-

*p = &a



&p → 890

&a → 780

a → 780

bcz no alg se mem. for a

3) *ptr = &a[3]; ⟶ 780+12

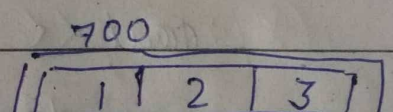
a = a+3 ✗

a → 780 → symbol table me value change ni hogi

Char Array :-

int a[] = {1,2,3};

char b[] = "abc";



cout << a; ⟶ 700

cout << b; ⟶ abc (diff behaviour)
It will print the whole array until it find null ptr

```
char * c = &b[0];
   cout << c;        // still abc
```

- char c1 = 'a'
  char * pc = &c1;
  cout << c1 —— a
  cout << pc —— a- - - - - until it finds any nullptr

- char str[] = "abcde";
  char *ptr = "abcde";

  It make a temp mem and then stores it in str
  str [abcde]

  temp [abcde]
  Here it is pointing to temp mem.
  (which is very wrong)

Pointers and Functions

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

```
void increment Pointer (int *p)
{  p = p+1;
}
```

↳ call by value no change in original pointer

```
void increment (int *p)
{
   (*p)++;
}
```

change in original value as it goes to *p and changes value as increment only has copy of p not *p (i.e i)

Double Pointer

These are pointers which stores address of other pointers

```
i = 10
int *p = &i;
int **p2 = &p
```

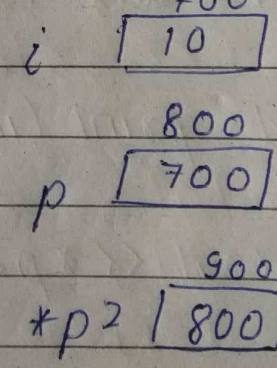
pointer bnao pr kis data type ka int*

Example

// p = *p2 = 700

// *p2 = p = &i = 700

// i = *p = **p2 = 10



DYNAMIC ALLOCATION

Address Typecasting

```
int i = 65;
char c = i;        // A
```
Implict Typecasting

```
int *p = &i;
char *pc = p;
```
Is It possible

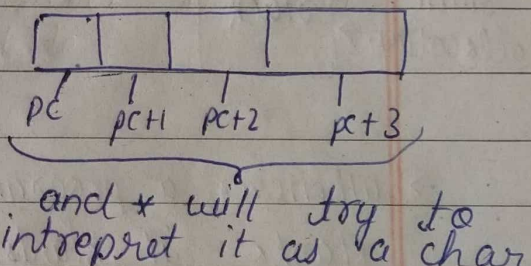
p - [int] → char*pc

→ Cannot be implicity typecasted (will give error)

means pc is also pointing to the thing where p is pointing

```
char *pc = (char*)p;
```
→ explicity Typecasting

* p → 66
* pc → A
* (pc+1) → null
* (pc+3) → null



and * will try to intrepret it as a char

Reference and Pass by reference

```
int i = 10;
int j = i;
```

i++; → no effects on j

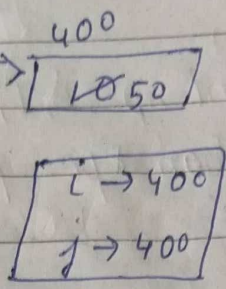
i: 10

j: 10

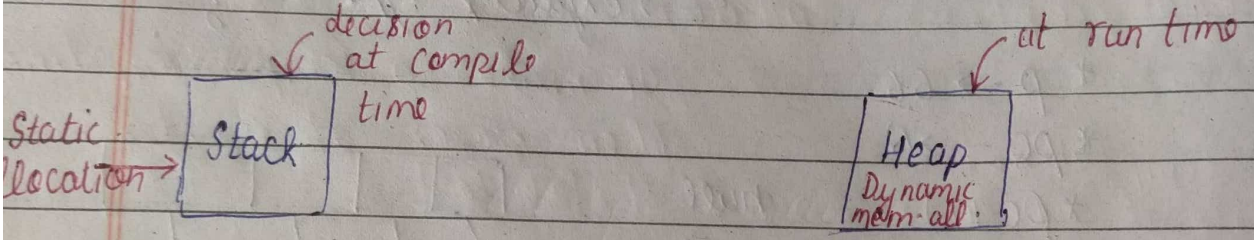
They are at different different memory so no changes will happen.

→ But to make effects we will make j as reference variable.
(we are not making new mem. for j it is just refering the other one)

```
int i=10;
int &j=i;
int k=50;
j=k;
```



DYNAMIC MEMORY ALLOCATION



→ Whenever a program starts it starts with some stack mem.

```
int i=10;
int a[20];
```

The problem comes when

```
int n
cin>>n
int a[n];
```

↓

at run time it is deciding the size not at compile time so ye kuch memory stack ki lekar start ho jayega par n if ek bhi bda to problem so make it in heap!

How

int i=10; ⟶ Stack

new int ⟶ makes a 4 byte mem. in heap

↓

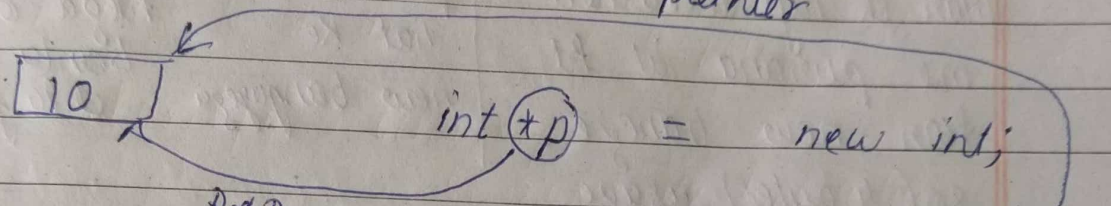
akeli ye statement useless hai

Now

↓

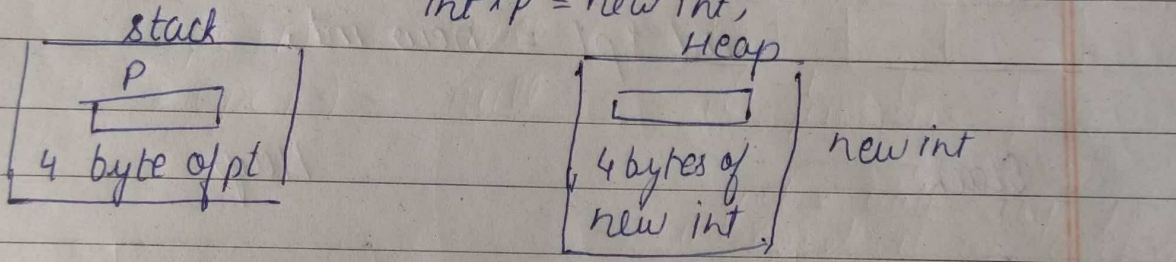
How to access it as it has no name.

actually it gives address of that allocated mem. so we can use pointer



Dereferencing to a memory

int *p = new int;



→ Now If I make array in Heap then it will give address of first index of array.

int *a = new int [n];

↓

Now we can use a as we normally us in array

a[0] = 10;

↓

Bcz *(a+0)

Now we can do this

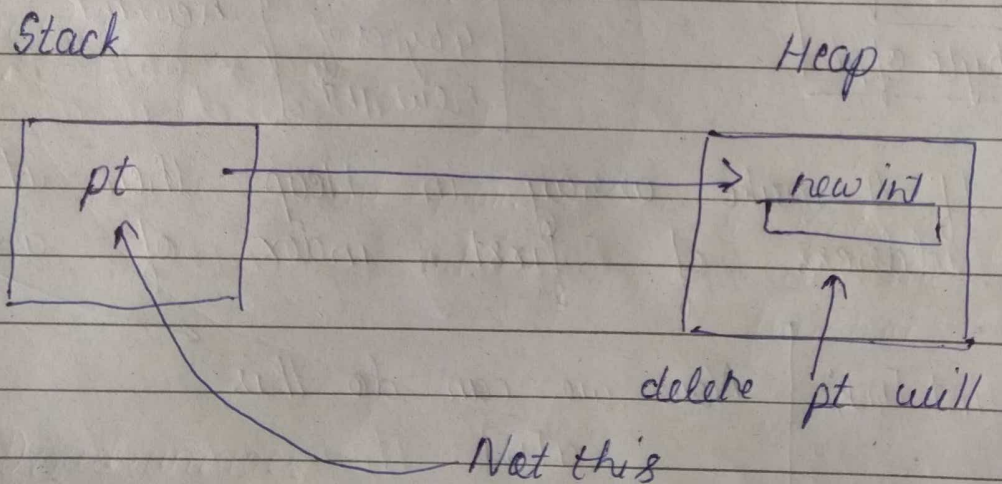
a[n]

↓

if I use new

Static	Dynamic
i) Auto release the mem. acc. to scope	i) we manually need to release the mem.
Eg.	Eg.
while (true)	while (true)
{	{
int i = 10;	int *p = new int; → Jb tak mem khtm hi hoti
}	
//Har ek nyi iteration me purana i kt kar new banega so (4 bytes) usage only	purana pointer kat ke new banjayega / purana nhi hoga or banta chla jayega

int *pt = new int;



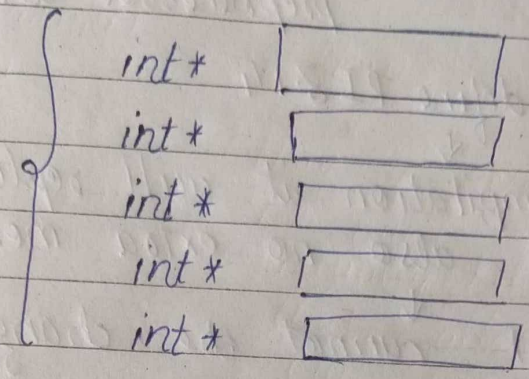
pointer jisko point kar raha vo delete ho rah hai pointer khud nhi

delete [] p; // for array.

Dynamic memory allocation for 2D array

int a[5][20]

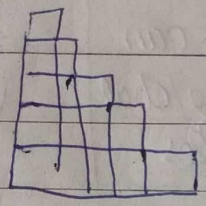
Now we have to store all these



for deleting within loop
delete [] p[i];

int **p = new int *[5]; — for rows

Now for col.
with in loop
p[0] = new int [20]; — for col



→ We can make 2D array like this also

#DEFINE

- It allows the programmer to give a name to a constant value before the program is compiled.
- Don't take any program memory space on the chip.

#include <iostream>

before compile include this already

↓

preprocessor directive

Same as above

define PI 3.14

↓

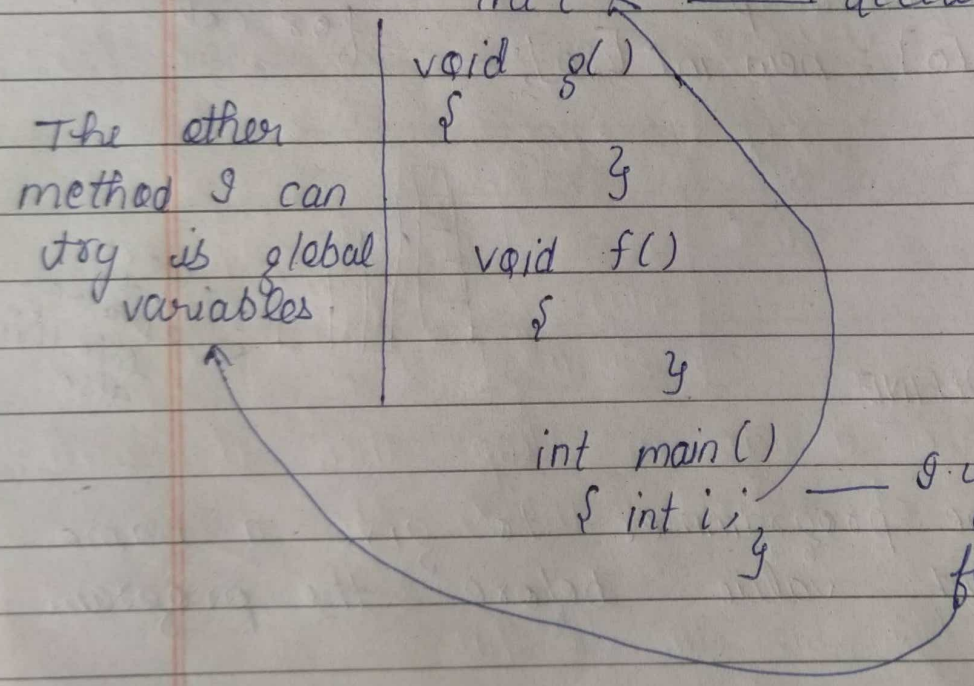
Before compilation it will replace all PI with 3.14 and also no extra mem. is allocated.

↓

also it cannot be changed

#Global Variable

int i —— declared globally



Disadvantages :-

Data can be modified by any function