C, C++, DSA in depth

Application of Pointers

Saurabh Shukla (MySirG)

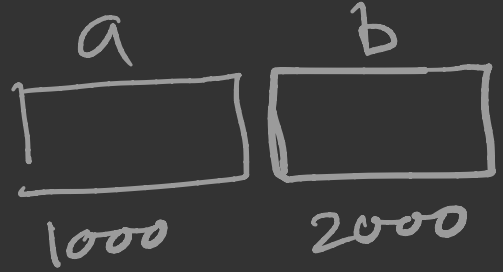
Agenda

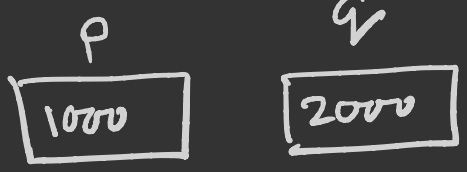
1. Pointer's Arithmetic
2. Call by reference
3. Pointers and arrays
4. Pointers and strings
5. Array of pointers
6. Pointer to array
7. Wild pointer
8. NULL pointer
9. Dangling pointer
10. Void pointer

- pointer is a variable
- It contains address of another variable
- size of pointer is not dependent on it type.
- Pointer jis type ka hota usi type ke variable ko point karta hai
- int *p;
- *p ⇝ variable pointed by p

Pointer's Arithmetic

```
int a, b, *p, *q;
p = &a;
q = &b;
```





q - p 250
p - q -250

p + 1 = 1000
p + 1 = 1001
p + 1 = 1004

p + q
p * q
p / q
} Error

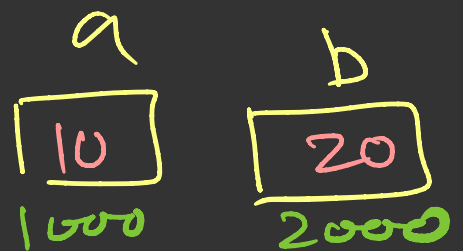
p * 5
p / 3
} Error

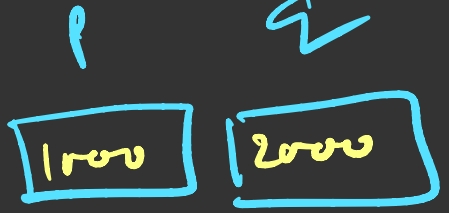
p + 1 = 1004
p + 2 = 1008
p + 5 = 1020
p + i = base address + i * size of type of (p)

p - 3 = 1000 - 3 * 4
= 988

You can subtract address from an address provided they are of same type.

```
int main()
{
    int a, b;
    printf("Enter two numbers");
    scanf("%d %d", &a, &b);
    swap(&a, &b);
    printf("%d %d", a, b);
    return 0;
}
void swap(int *p, int *q)
{
    int c;
    c = *p;
    *p = *q;
    *q = c;
}
```





*p = a;

Call by Reference

```
void swap(int *, int*);
int main()
{
    int a,b;
    printf("Enter two numbers");
    scanf("%d %d",&a,&b);
    swap(&a,&b);
    printf("%d %d", a, b);
    return 0;
}
void swap(int *p, int *q)
{
    int t;
    t = *p;
   *p = *q;
   *q = t;
}
```

call by reference

Formal argument

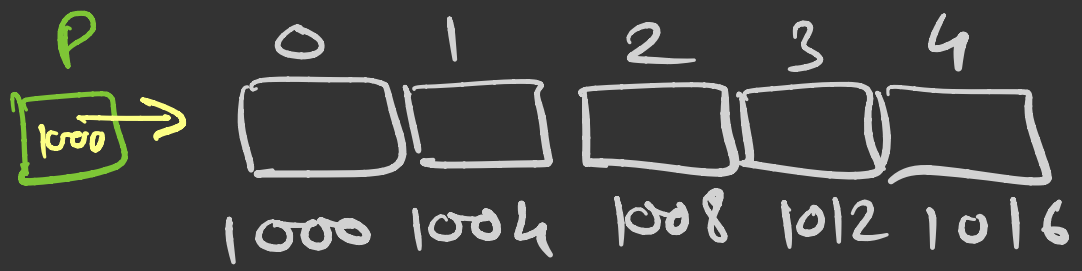
1. ordinary variable
2. pointer variable

Why we use & in scanf() ?

```
scanf("%d", &x);
```

Pointers and Arrays

```
int a[5];
int *p;
P=&a[0];
```



P+0	1000	&a[0]	*(P+0)
P+1	1004	&a[1]	*(P+1)
P+2	1008	&a[2]	*(P+2)
P+3	1012	&a[3]	*(P+3)
P+4	1016	&a[4]	
P+i		&a[i]	*(P+i)

a[0]
a[1]
a[2]
a[3]
a[i]

a[2] → *(a+2) → *(2+a) → 2[a]

p[2] → *(p+2) → *(2+p) → 2[p]

what is the difference between a and p.

p is a variable

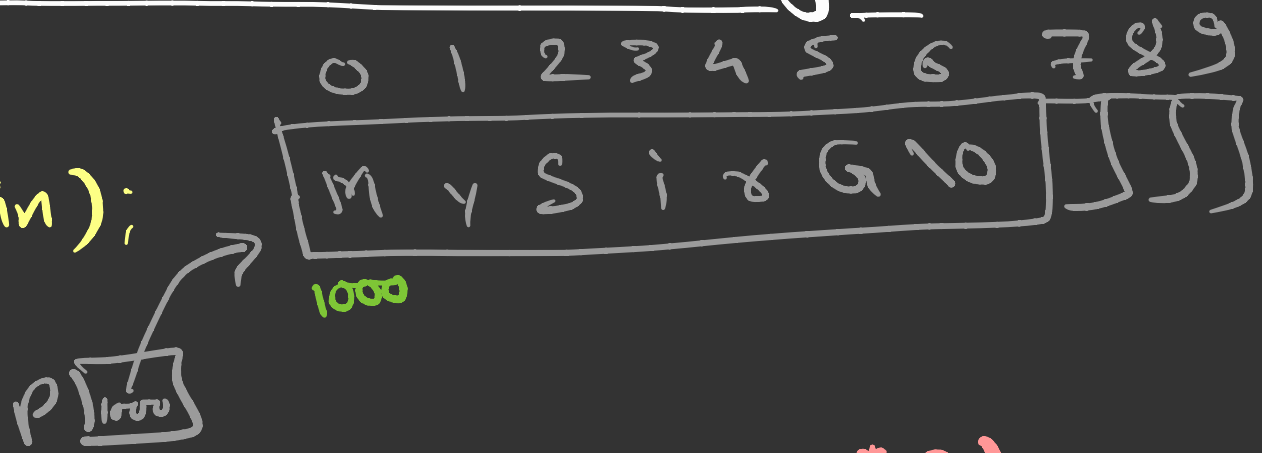
a is constant

p++ ✓

a++ ✗

Pointers and Strings

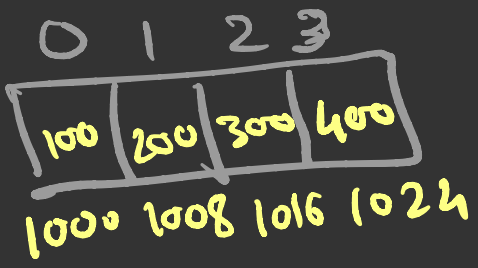
```
int l;
char str[10];
fgets(str, 10, stdin);
l = length(str);
```



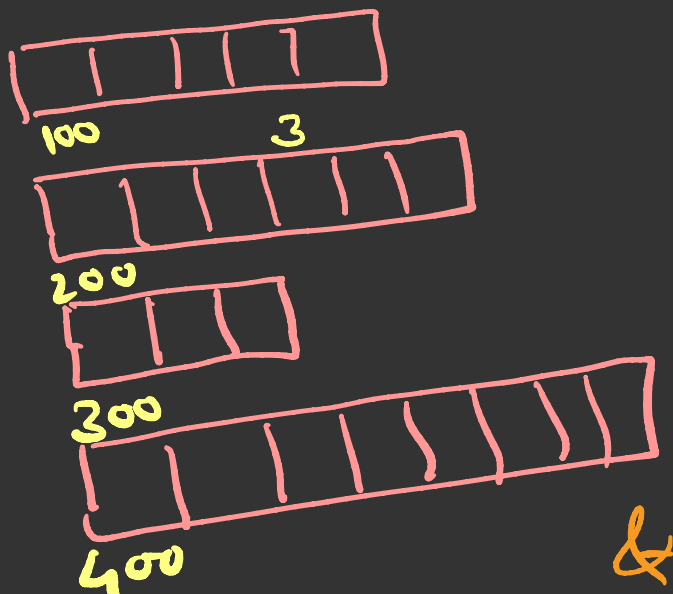

```
int length(char *p)
{
    int i;
    for(i=0; *(p+i); i++);
    return i;
}
```

Array of Pointers

```
int *p[4];
```




```
int a[5], b[6], c[3], d[8];
p[0] = &a[0];
p[1] = &b[0];
p[2] = &c[0];
p[3] = &d[0]
input(p)
```



p[0] == a
p[1] == b
p[2] == c
p[3] == d

&a[j]
& p[0][j]
& *(p+i)[j]

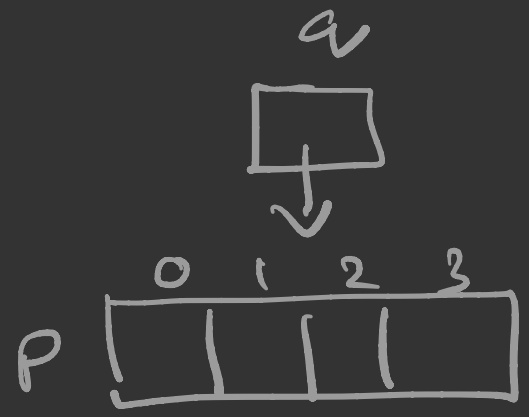
q ≒ & p[0]
*q ≒ p[0]

p[0] ≒ & a[0]
*(p[0]) ≒ a[0]

*q p[0]
*(q+1) p[1]
*(q+2) p[2]

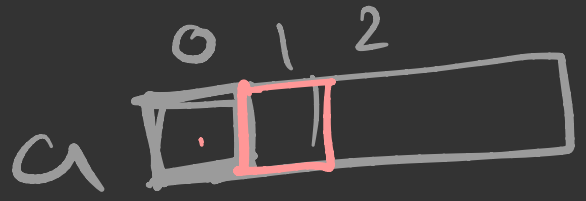
$*(a+i) \simeq p[i]$

i=0	a
i=1	b
i=2	c
i=3	d



$p[0] == a$

✓ p[0]+0	&a[0]
p[1]	&b[0]
p[2]	&c[0]



✓ p[0]+1	&a[1]
✓ p[0]+2	&a[2]
✓ p[0]+3	&a[3]
p[0]+j	&a[j]

p[0][1] ✗

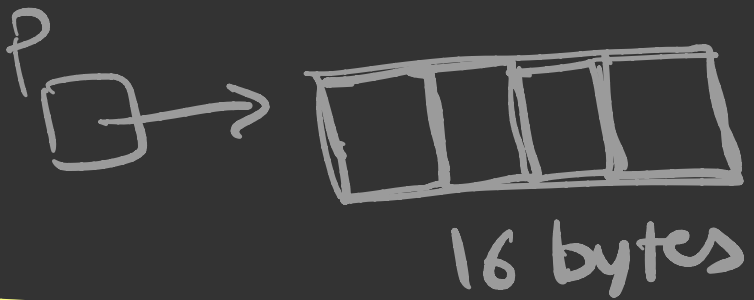
*(p[0]+1) ✗

i=0 *(a+i)+j

Pointer to Array

```
int (*p)[4];
```

P is a pointer to an array of int type with 4 blocks.

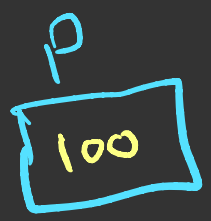


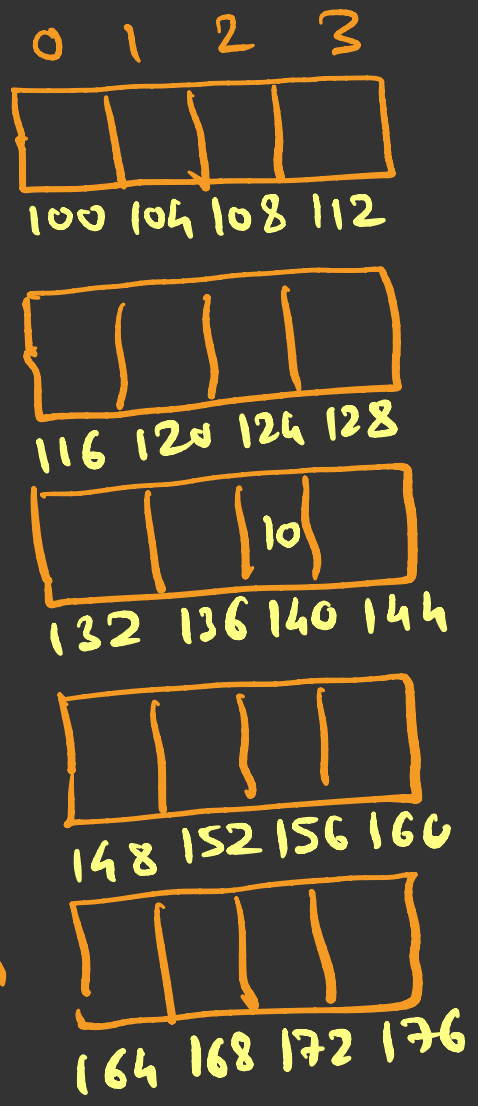
p+1 = 1016
p+2 = 1032
p+3 = 1048
p+4 = 1064

```
int *p;
```

p+1 1004
p+2 1008

```
int (*p)[4];
int a[5][4];
p = a;
```





a[2][2]=10 1

*(P+2)[2]=10 2

((P+2)+2)=10 3

*(P[2]+2)=10 4

P[2][2]=10

P+1 116
P+2 132
P+3 148
P+4 164

*140 = 10

a[2][2] = 10

*(a[2]+2)=10

((a+2)+2)=10

((P+2)+2)=10

a[i][i]
P[i][j]

Wild Pointer

An uninitialized pointer is a wild pointer.

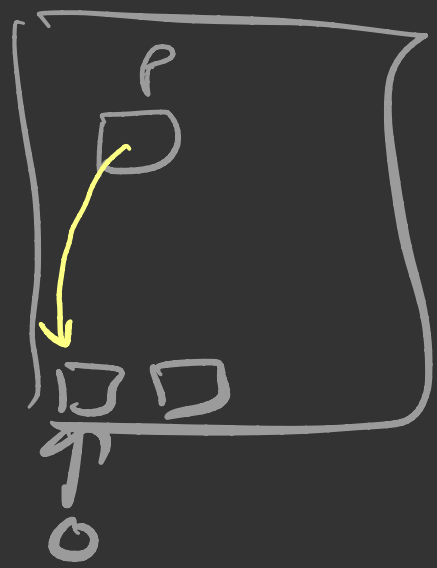
```
void f1()
{
        int *p;   <- wild pointer
        *p = 5;   <- illegal use of pointer
        ....

}
```

NULL Pointer

- A pointer containing NULL (special address) is known as NULL pointer.
- If a pointer containing NULL, we consider it as if it is not pointing to any location.
- As a safe guard to illegal use of pointers you can check for NULL before accessing pointer variable

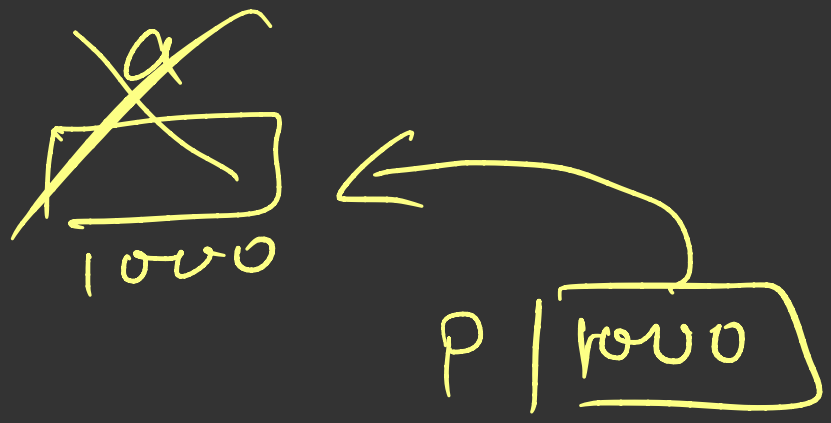
```
int  *p = NULL;
...
if (p != NULL)
{
      *p = 5;
}
```



Dangling Pointer

A pointer pointing to a memory location that has been deleted is called dangling pointer.

```
int * f1()
{
    int a;
    .....
    return &a;
}
void fun()
{
    int *p;
    p = f1();
    *p = 5;
}
```



p is dangling pointer

illegal use of pointer

```
void f1()
{
    int *p;
    {
        int x;
        p=&x;
        ....
    }
    *p=5;
}
```

Scope and life of x is limited to the block

After this line p becomes dangling pointer

illegal use of pointer

```
void f1()
{
    int *p;
    p = (int *) malloc (sizeof(int));
    ...
    ...
    free(p);
    *p = 5;
}
```

← p becomes a dangling pointer

← illegal use of pointer.

Void Pointer

- void pointer is a generic pointer that has no associated data type with it.
- void pointer can hold address of any type.

```
void *p;
int x;
p=&x;
```

```
void *p;
float y;
p=&y;
```

void pointers can not be dereferenced

```
*p=5;
```

```
*p=3.5f;
```

Error

- However void pointer can be dereferenced using typecasting.

```
*(int*)p = 5;
```

```
*(float*)p = 3.5f;
```