

C Language

Application of Pointers



Saurabh Shukla (MySirG)

Agenda

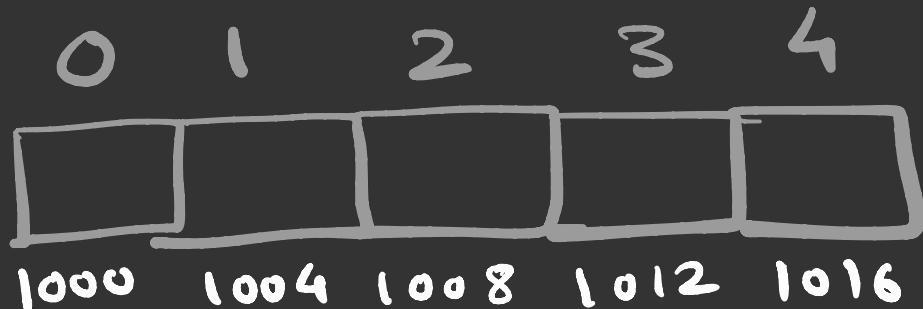
- ① Pointers' Arithmetic ✓
- ② Call by reference ✓
- ③ Pointers and arrays
- ④ Pointers and strings
- ⑤ Array of pointers
- ⑥ Pointer to array
- ⑦ Wild pointer
- ⑧ NULL pointer
- ⑨ Dangling pointer
- ⑩ Void pointer

Pointers and Arrays

```
int a[5];
```

```
int *p;
```

```
p = &a[0];
```



index

0	$p + 0$	1000	$\&a[0]$
1	$p + 1$	1004	$\&a[1]$
2	$p + 2$	1008	$\&a[2]$
3	$p + 3$	1012	$\&a[3]$
4	$p + 4$	1016	$\&a[4]$

$i=0 \rightarrow 4 \quad p+i$

$*(\mathbf{p} + 0)$	$a[0]$
$*(\mathbf{p} + 1)$	$a[1]$
$*(\mathbf{p} + 2)$	$a[2]$
$*(\mathbf{p} + 3)$	$a[3]$
$*(\mathbf{p} + 4)$	$a[4]$
$*(\mathbf{p} + i)$	$a[i]$

$$a[2] \rightarrow *(\&a + 2) \rightarrow *(2 + a) \rightarrow 2[a]$$

$$p[2] \rightarrow *(\&p + 2) \rightarrow *(2 + p) \rightarrow 2[p]$$

what is the difference between a and p .

p is a variable

a is a constant (address)

$$p = \checkmark$$

$$a = \times$$

Pointers and Strings

```
char *p;
```

```
char str[] = "Bhopal";
```

```
p = str; // p = &str[0]
```

```
printf(">.s", p);
```

```
int length(char s[])
{
    int i;
    for(i=0; s[i]; i++);
    return i;
}
```

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

Array of Pointers

```
int *p;
```

```
int *ptr[4];
```

```
int a[5], b[3], c[6], d[7];
```

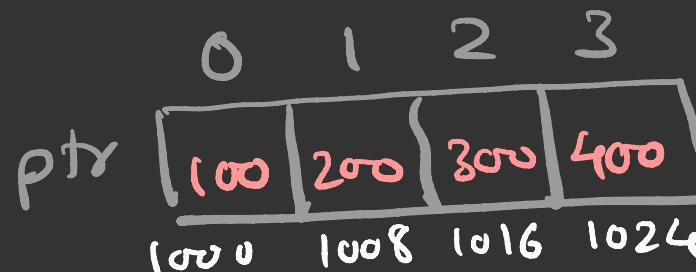
```
ptr[0] = a;
```

```
ptr[1] = b;
```

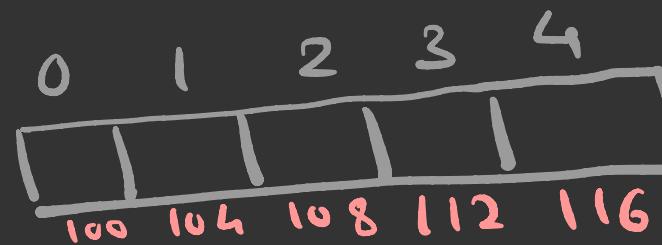
```
ptr[2] = c;
```

```
ptr[3] = d;
```

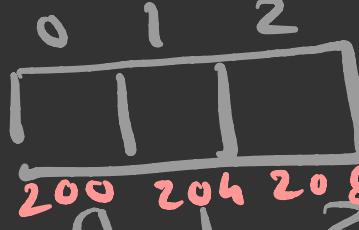
```
f1(ptr);
```



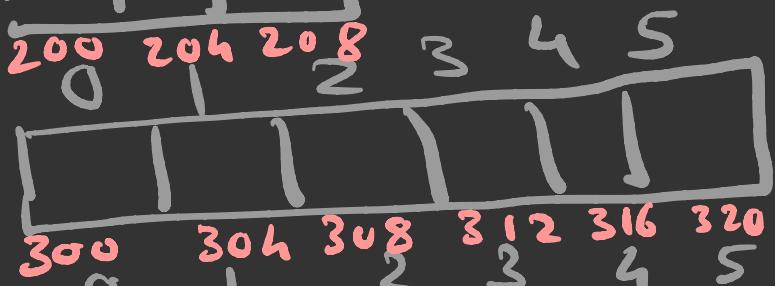
a



b



c



d



Pointer to Array

`int * p ;`

P is a pointer to an int

`int * p[4];`

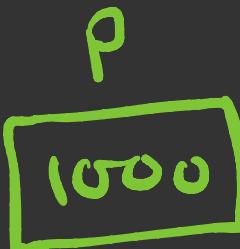
P is an array of int pointers

`int (* p)[4];`

P is a pointer to an int
array of 4 variables.



P + 1 1004



P + 1 1016

int a[4][5];

int *P;

P = &a[0][0];



P+1 → 1004

P+2 → 1008

int (*P)[5];

P = &a[0][0];



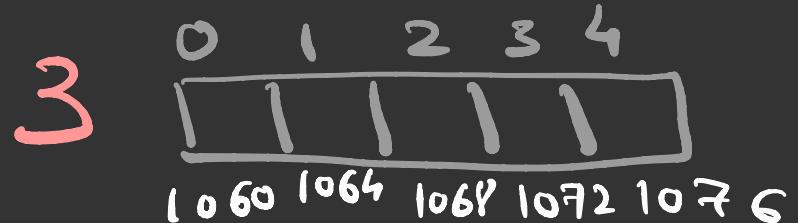
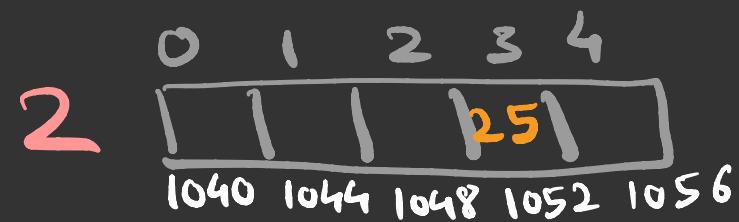
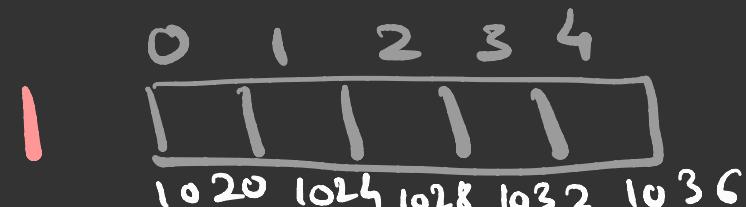
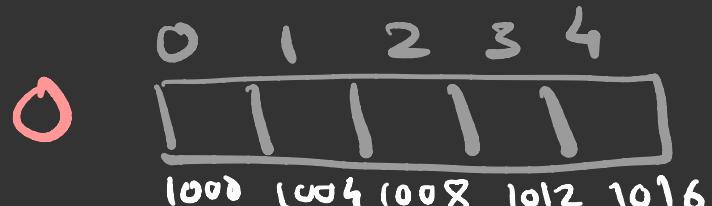
P+1 → 1020

P+2 → 1040

a[2][3] = 25;

*(a[2] + 3)

((&a+2) + 3) →



* (*(&P+2) + 3) → P[2][3]

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;           // illegal use of pointer
}
```

P is dangling pointer

```
void f1()
{
    int *p;
    {
        int x; // Scope and life of x is
        p=&x; // limited to the block
        ...
    } // After this line
    *p=5; // p becomes dangling
           // pointer
}

```

The handwritten annotations provide additional context:

- An arrow points from the brace of the inner block to the declaration of `x`, with the text "Scope and life of `x` is limited to the block".
- An arrow points from the brace of the inner block to the assignment `p = &x;`, with the text "After this line `p` becomes dangling pointer".
- An arrow points from the assignment `*p = 5;` to the text "illegal use of pointer".

```
void f1()
{
    int *p;
    p = (int *) malloc (sizeof(int));
    ...
    ...
    free(p);  $\leftarrow$  P becomes a dangling pointer
    *p = 5;  $\leftarrow$  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;		void *p;
int x;		float y;
p=&x;		p=&y;

void pointers can not be dereferenced

* p = 5;		* p = 3.5f;
		
Error		

• However void pointer can be dereferenced using typecasting.

```
void *P;  
int a;  
P = &a;
```

$*(\text{int}^*)P = 5;$



$*(\text{float}^*)P = 3.5f;$

$\star(\text{int}^*)P = 5; \checkmark$