C Language

Application of Pointers

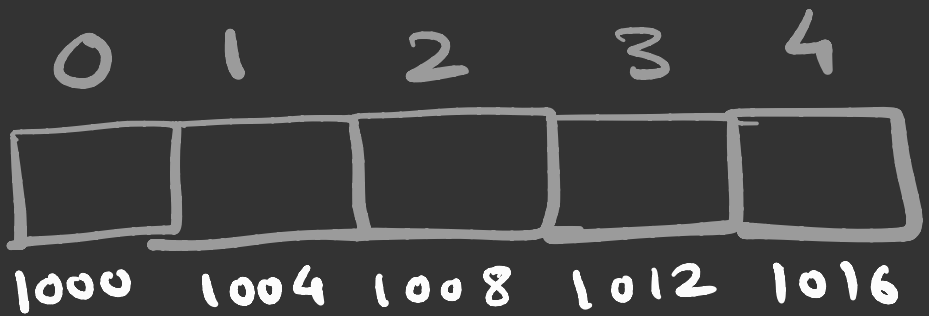
Saurabh Shukla (MySirG)

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

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=0to4	p+i		&a[i]

*(p+0)	a[0]
*(p+1)	a[1]
*(p+2)	a[2]
*(p+3)	a[3]
*(p+4)	a[4]

*(p+i) ≅ a[i]

a[i] → *(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.

a ≅ 1000
p ≅ 1000

p is a variable
a is a constant

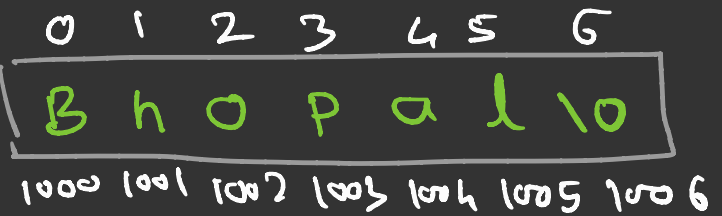
p = ✓

a = ✗

a[i] ≅ p[i]
*(a+i) ≅ *(p+i)

Pointers and Strings

```
char *p;
char str[] = "Bhopal";
p = str;  // p = &str[0]
printf("%s", p);
```

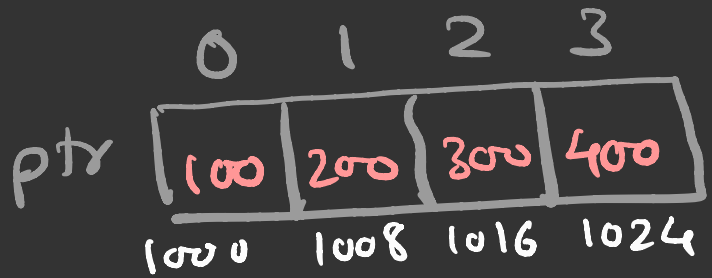


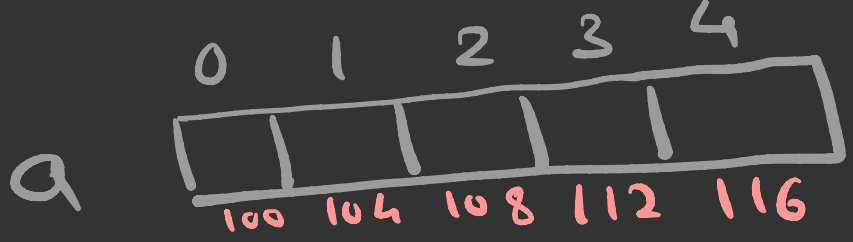
```
int length(char s[])
{
    int i;
    for(i=0; s[i]; i++);
    return i;
}
```

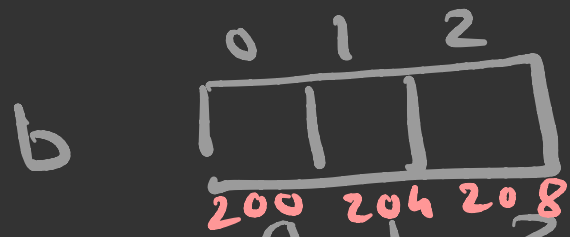
```
int length(char *p)
{
    int i;
    for(i=0; p[i]; i++);
    return i;
}
```

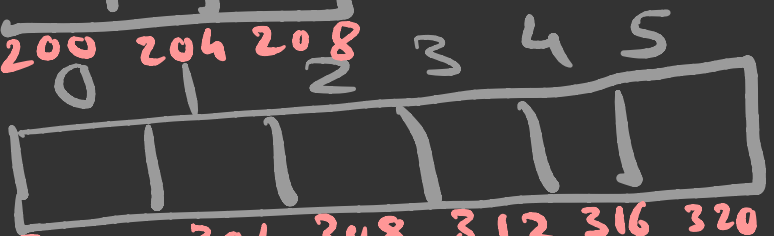
Array of Pointers

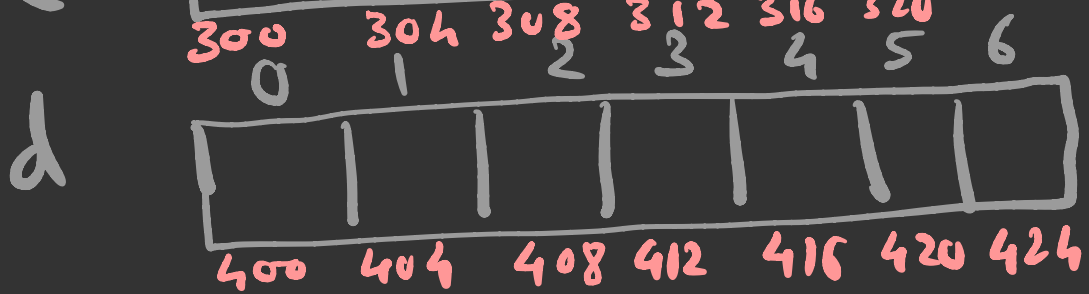
```
int *p;
int *ptr[4];
int a[5], b[3], c[6], d[7];
ptr[0] = a;
ptr[1] = b;
ptr[2] = c;
ptr[3] = d;
f1(ptr);
```









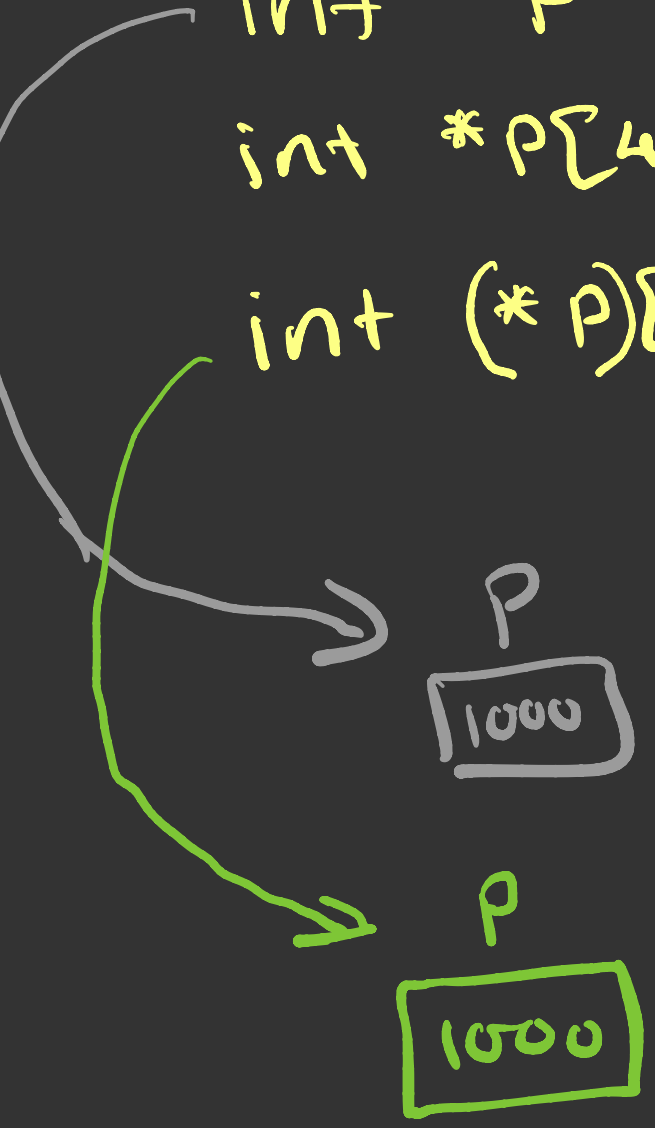


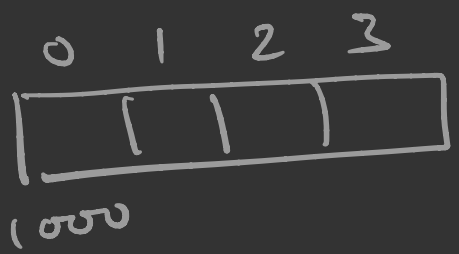
Pointer to Array

```
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.





int a[4][5];

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

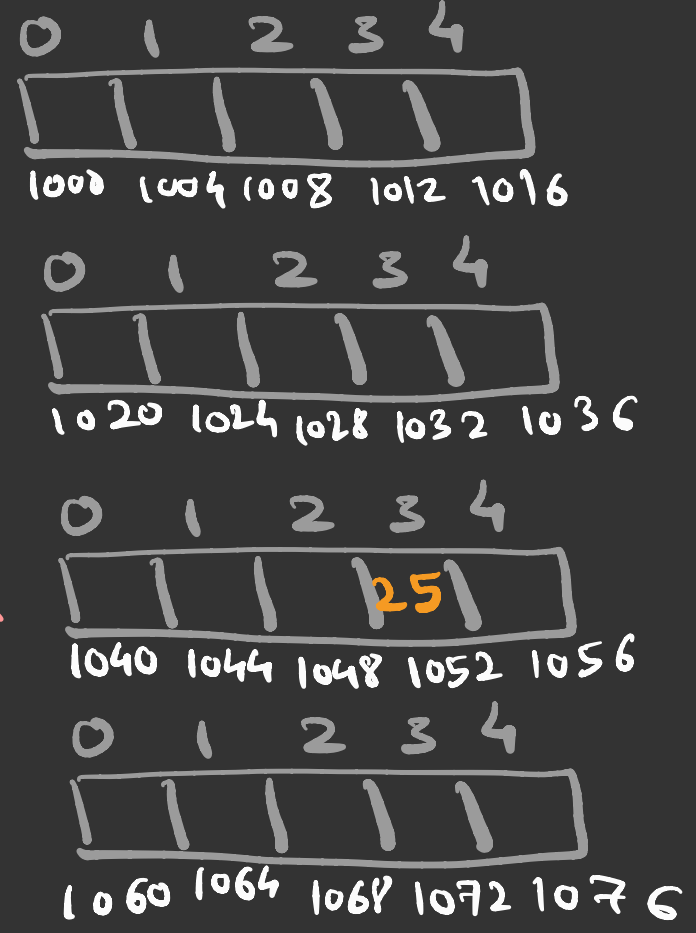
p → 1000

p+1 → 1004
p+2 → 1008

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

p → 1000

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;
}
```

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;
```