UNIT 3

POINTERS: Variables that store the value of another variable
→ Can be used to access low level
→ Used to created OS and drivers

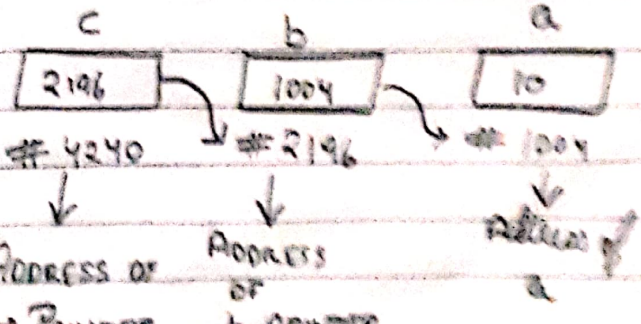
- OPERATORS
  & → Referencing / Address of
  * → Dereferencing / Value at

* VOID pointers can point to any datatype

- POINTER INCREMENT
  In 32 bit machine, increment is by 2
  In 64 bit machine, increment is by 4
* Same for decrement ^

POINTER TO POINTER (DOUBLE POINTER)
Chain of pointers, stores address of another pointer

Eg.
```
void ptr(){
    int a = 3000;
    int *b = &a
    int **c = &b
    printf("%d", a)
    printf("%d", *b)
    printf("%d", **c)
```



DANGLING POINTER

Class Fun
Date
Page No

ARRAY POINTER

Pointer variable points to address of first element.

Eg. // Printing array using pointer

```
int main(){
    int (*a)[5];    // return_type (*name)[size];
    int b[5] = {1,2,3,4,5}; // declaring array
    a = &b
    for (int i=0; i<5; i++){
        printf("%d \n", *(a+i)); // (*a+i) shifts to next subsequent value
    }
}
```

* a points to entire array
* *a points to first element of array
* **a gives value of first element of array.

Eg. // REVERSING ARRAY USING POINTER

Class Fun

Date | Page No.

POINTER TO FUNCTION

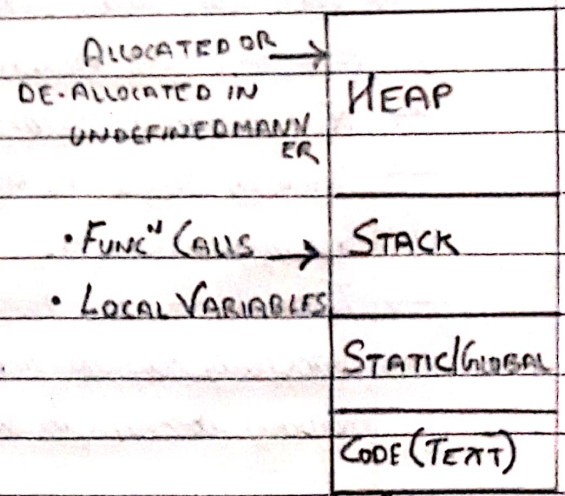
Points to code instead of data

Eg.
```
void func(){
void/int func(int a, int b){
                return a+b;
}
int main(){
        void/int (*func_ptr)(int,int) = &func;
        int result = (*func_ptr)(10,20);
        printf("result is %d \n", result);
}
```

*func_ptr points to start of code in func()

DYNAMIC MEMORY ALLOCATION

→ Unlike stack memory, heap can be modified with the help of dynamic memory allocation.



- FUNCTIONS

1) Malloc()
- Reserves a block of memory of given size. (Dynamic Array)
- Returns a void pointer to allocated space.
- Values are initialized to garbage.

ptr = (int*) malloc (size)

2) (alloc)
- Contiguous memory allocation
- Reserves 'n' blocks of memory
- ptr = (int*) calloc (n, size)

3) Realloc()

- Reallocation of memory
-

```
ptr = (int*) realloc(ptr, new_size)
```

4) Free()

- Used to free allocated memory
- Frees memory being used by program

```
free(ptr)
```

STRUCTURE & UNION

Struct	UNION
→ User can access individual members at given time.	→ Users can access only one member at given time.
→ No shared location	→ Shared location
→ Altering single member does not affect others.	→ Altering one member affects the others.
→ Implementation takes place internally due to seperate memory locations for each member	→ Memory allocation takes place only for largest member. It stores same location for all members.

STRUCTURE : User defined datatype that allows you to group different datatypes.

```
struct student {
    char name[50];
    int age;
    float grades;
};
```

```
int main() {
    // Declare a struct variable
    struct student s;
    // Assigning values to members
    strcpy(s.name, "Michael Jordan");
    s.age = 20;
    s.grade = 4.0;

    // printing values of members
    printf("Name: %s\n", s.name);
    printf("Age: %d\n", s.age);
    printf("Grade: %f\n", s.grade);
}
```

Union

```
union number {
    int i;
    float f;
};
int main() {
    union number num; // Declaring Union variable
    num.i = 10;
    printf("Value of i is %d\n", num.i);
    num.f = 3.142; // Overwrites num.i
    printf("Value of f is %.2f\n", num.f);
    printf("%d", num.i); // Produces garbage value.
}
```

Class Fun

Date | Page No.

NESTED STRUCTURES

Structures within structure

```
struct date {
        int day, month, year;
};

struct user {
        char name[50];
        struct date birth_day ; // Access variable for above
                                  structure
};
```

* To access `user.birth_day.month` // accesses the month for per user.

ARRAYS OF STRUCTURE

Each element of array is a structure

Used to store multiple structures of similar type in contigous block of memory.

→ // declare a array variable for structure student

```
struct student students[size];
```

→ Add values to each individual student using index

```
student[i].age = 20; // assigns age of (i+1)th
                        student
```

Class Fun

Date | Page No.

Self Referential Structure

Union

A user defined datatype similar to structure however all members share same location.

Self-Referential Structure

Structures that contains a reference to data of its same type.

Eg.
```
struct node {
    int data1;
    int data2;
    struct node *link  // Link a pointer pointing to struct node.
};
```

1) Single Link SRS

Only a single pointer that carries address of next node.

2) Multiple Link SRS

Two pointers, pointing to same structure.