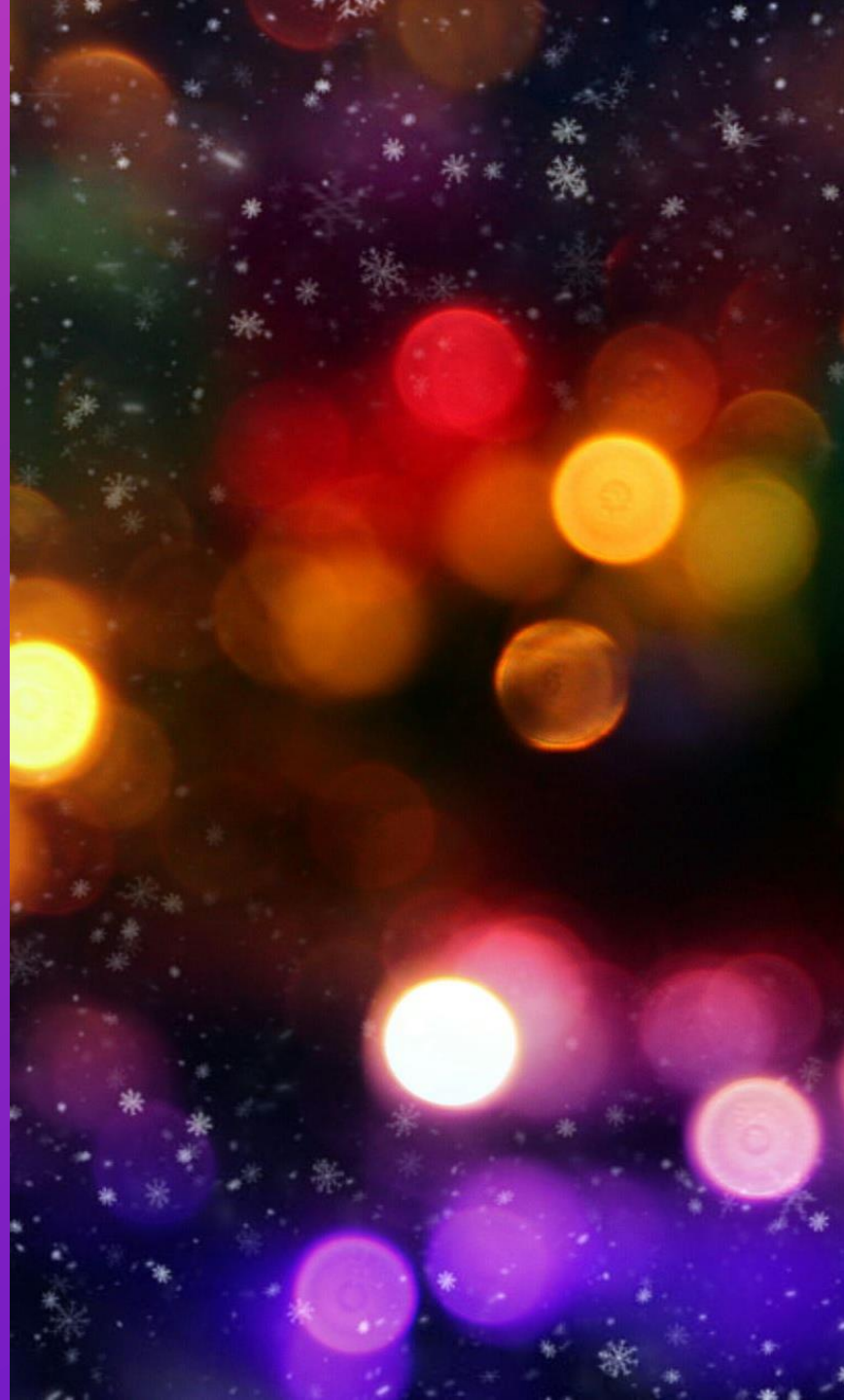
MNNIT COMPUTER CODING CLUB

CLASS-9

BASICS OF C



POINTERS

- The address of the first byte allocated to variable is known as address of variable.
- **'&' operator is the "address of" operator** in C which returns the address.
- The address operator cannot be used with a constant or an expression

Eg -

int a = 5;	-	&a	Valid
float f = 8.6;	-	&f	Valid
Constant	-	&26	Invalid
Expression	-	&(a+f)	Invalid

- Pointer variables are used to store the memory address. Used like normal variables.

POINTERS

- The general syntax of declaration of pointer variable is: datatype *p_name;
- The size allocated to all pointer variables is same as they all store integers.
- **'*' is used to dereference the pointer**
- In the program attached,
  - ptr can be used to access the address of variable a and *ptr can be used to access its value
  - Writing *(&a) and a is same

```c
#include<stdio.h>

int main()
{
    int a = 5;
    int *ptr = &a; // Address of a assigned to a pointer variable
    printf("Value of ptr : %u\n", ptr);
    /*
     Pointer variable can be copied and then
     both the pointers point to same address
    */
    int *ptr2 = ptr;
    printf("Value of ptr2 : %u\n", ptr2);
    //Dereferencing operator
    printf("Value of data at address ptr is: %d\n", *ptr);
    return 0;
}

```

POINTER ARITHMETIC

- Addition and subtraction of an integer and a pointer variable is supported
- Pointer variables can also be used for increment and decrement operation
- **The increment/decrement operation changes the value of pointer variable by the size of data type that it points to.**
- Subtraction of two pointer variables of same base type returns the number of values present between them

Eg – int *ptr1 = 2000, *ptr2 = 2020;

printf("%u\n", ptr2-ptr1); Output : 5

test.c > ...

```c
#include<stdio.h>

int main()
{
    int a = 5, *pi = &a;;
    double b = 8.8, *pd = &b;
    char ch = 'P', *pc = &ch;;
    printf("Old value of pi : %u\n", pi);
    printf("Old value of pd : %u\n", pd);
    printf("Old value of pc : %u\n\n", pc);
    pi++;
    pd = pd + 2;
    pc++;
    printf("New value of pi : %u\n", pi);
    printf("New value of pd : %u\n", pd);
    printf("New value of pc : %u\n", pc);
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
$ gcc -w test.c
$ ./a.out
Old value of pi : 786925444
Old value of pd : 786925448
Old value of pc : 786925443

New value of pi : 786925448
New value of pd : 786925464
New value of pc : 786925444
$
```

COMBINATION OF DEREFERENCE AND INCREMENT/DECREMENT

- The dereference operator (*), address of operator (&) and increment/decrement have same precedence and are **Right to Left Associative**.

Expression	Evaluation	
x = *ptr++	x = *ptr	ptr = ptr + 1
x = *++ptr	ptr = ptr + 1	x = *ptr
x = (*ptr)++	x = *ptr	*ptr = *ptr + 1
x = ++*ptr	*ptr = *ptr + 1	x = *ptr

POINTER TO POINTER

- Pointer variable contains an address, and this variable takes space in memory so it itself has an address
- A pointer-to-pointer variable is used to store the address of a pointer variable
- The general syntax of declaration of pointer variable is:

datatype **pp_name;

```c
#include<stdio.h>

int main()
{
    int a = 5;          // Integer variable
    int *ptr = &a;      // Pointer to int
    int *pptr = &ptr;   // Pointer to pointer to int
    printf("Address of a   : %u\n", &a);
    printf("Value of ptr   : %u\n", ptr);
    printf("Address of ptr : %u\n", &ptr);
    printf("Value of pptr  : %u\n", pptr);
    printf("Address of pptr: %u\n", &pptr);
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
$ gcc -w test.c
$ ./a.out
Address of a   : 3882647076
Value of ptr   : 3882647076
Address of ptr : 3882647080
Value of pptr  : 3882647080
Address of pptr: 3882647088
$
```

COMBINATION OF DEREFERENCE AND INCREMENT/DECREMENT

- The dereference operator (*), address of operator (&) and increment/decrement have same precedence and are **Right to Left Associative**.

Expression	Evaluation	
x = *ptr++	x = *ptr	ptr = ptr + 1
x = *++ptr	ptr = ptr + 1	x = *ptr
x = (*ptr)++	x = *ptr	*ptr = *ptr + 1
x = ++*ptr	*ptr = *ptr + 1	x = *ptr

POINTER TO POINTER

- Pointer variable contains an address, and this variable takes space in memory so it itself has an address
- A pointer-to-pointer variable is used to store the address of a pointer variable
- The general syntax of declaration of pointer variable is:

datatype **pp_name;

```c
#include<stdio.h>

int main()
{
    int a = 5;          // Integer variable
    int *ptr = &a;      // Pointer to int
    int**pptr = &ptr;   // Pointer to pointer to int
    printf("Address of a   : %u\n", &a);
    printf("Value of ptr   : %u\n", ptr);
    printf("Address of ptr : %u\n", &ptr);
    printf("Value of pptr  : %u\n", pptr);
    printf("Address of pptr: %u\n", &pptr);
    return 0;
}
```

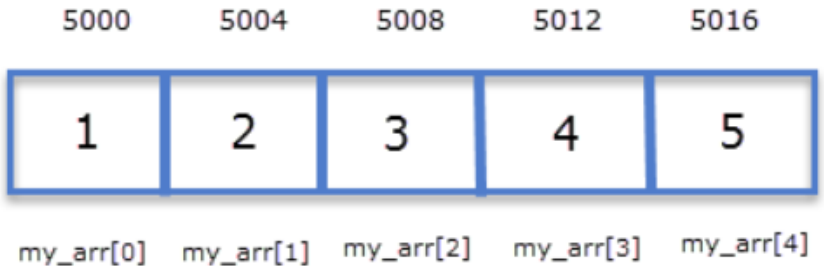
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
$ gcc -w test.c
$ ./a.out
Address of a   : 3882647076
Value of ptr   : 3882647076
Address of ptr : 3882647080
Value of pptr  : 3882647080
Address of pptr: 3882647088
$
```

POINTER WITH 1D ARRAYS

Consider an array

```
int arr[] = {1,2,3,4,5};
```



Here **arr** is a pointer to the first element aka arr is a pointer to int or (int*)

Remember
arr = &arr[0]
arr + 1 = &arr[1]
arr + 2 = &arr[2]
arr + 3 = &arr[3]

Thus
*(arr) = arr[0]
*(arr + 1) = arr[1]
*(arr + 2) = arr[2]
*(arr + 3) = arr[3]

int *p;
int arr[] = {11, 22, 33, 44, 55};
p = arr;

We **can do** p++, p--
but we **can not do** arr++, arr--

POINTER AND FUNCTIONS

- Call by value

```
void swapx(int x, int y)
{
    int t;
    t = x;
    x = y;
    y = t;
    printf("x=%d y=%d\n", x, y);
}
```

- Call by reference

```
void swapx(int* x, int* y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
    printf("x=%d y=%d\n", *x, *y);
}
```

How can we return more than one value from function?