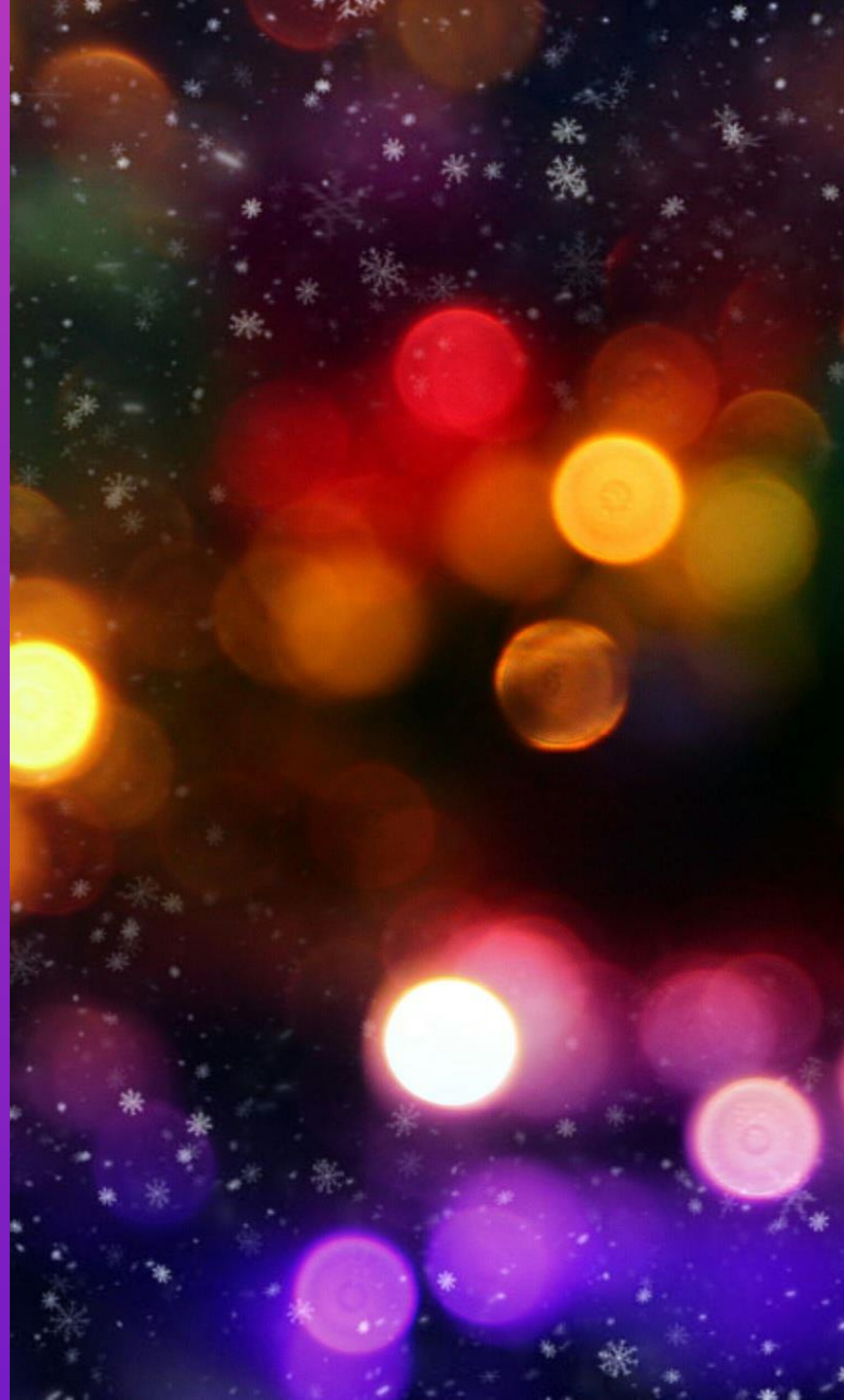
MNNIT COMPUTER CODING CLUB

CLASS-10

BASICS OF C



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:

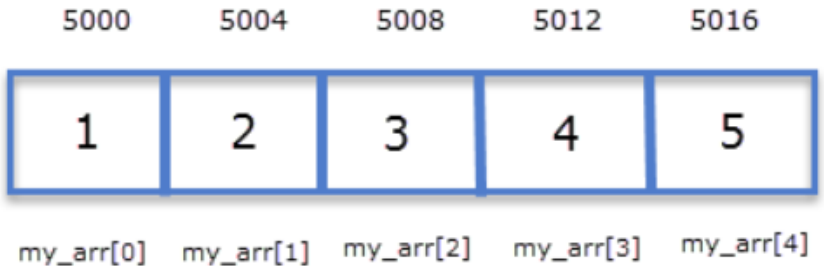
datatype **pp_name;

```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?

DYNAMIC MEMORY ALLOCATION

void pointer

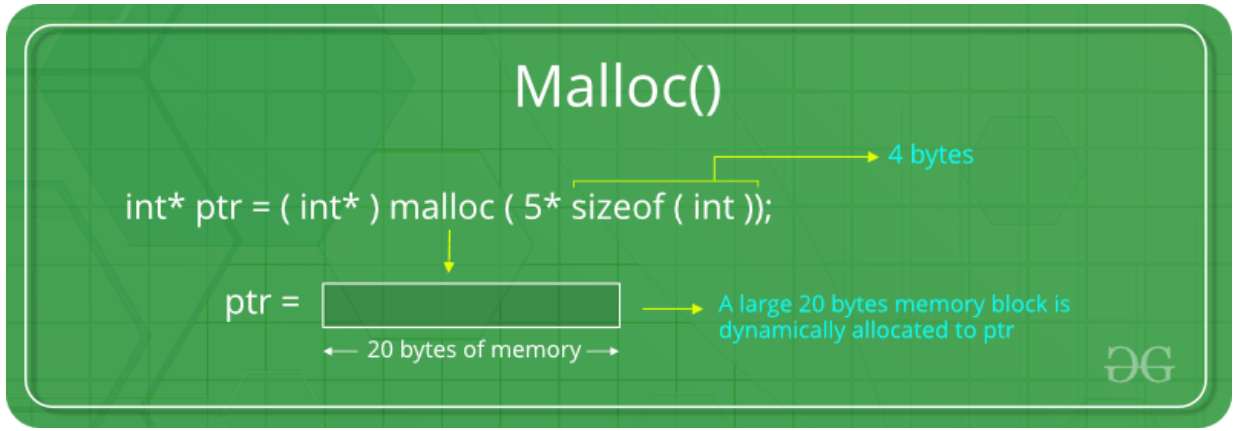
- The void pointer in C is a pointer which is not associated with any data types.
- It is a general-purpose pointer.
- It can point to any data type.

```
int a = 7;
float b = 7.6;
void *p;

p = &a;
printf("Integer variable is = %d", *( (int*) p) );

p = &b;
printf("\nFloat variable is = %f", *( (float*) p) );
```

DYNAMIC MEMORY ALLOCATION

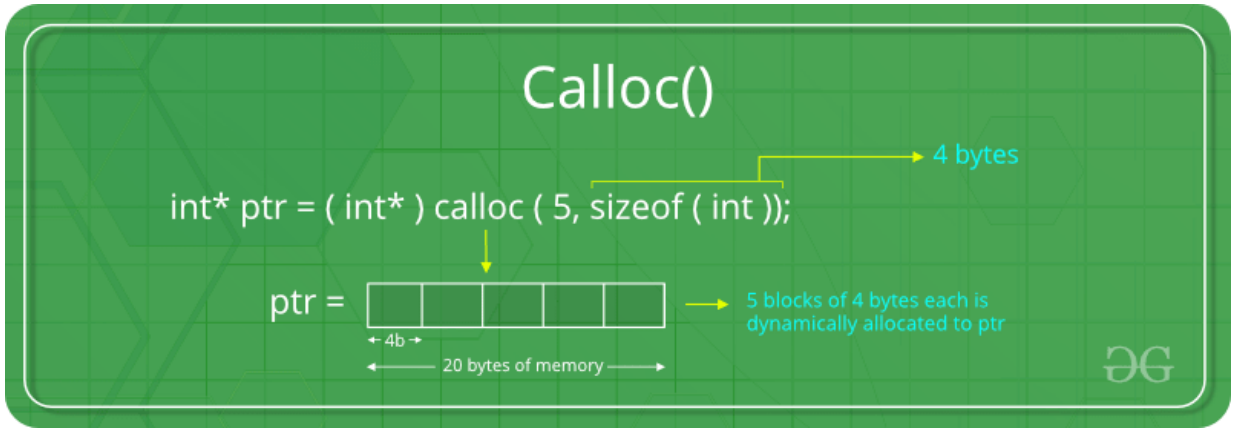
- **Malloc**

```
ptr = (cast-type*) malloc(byte-size)
```



- **Calloc**

```
ptr = (cast-type*)calloc(n,element-size);
```



src: https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/

STRUCTURE

A structure is a user-defined data type available in C that allows to combining data items of different kinds. Structures are used to represent a record.

Syntax:

```
struct [structure name]
{
    member definition;
    member definition;
    ...
    member definition;
};
```

```
struct struct_example
{
    int integer;
    float decimal;
    char name[20];
};
```

<u>Creating an object:</u>

```
struct struct_example s={10,10.0,"abcdef"};
```

<u>Access(read/write):</u>

s.integer
s.decimal
s.name

UNION

A union is a special data type available in C that allows storing different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time.

Syntax:

```
union [union name]
{
   member definition;
   member definition;
   ...
   member definition;
};
```

```
union union_example
{
    int integer;
    float decimal;
    char name[20];
};
```

<u>Creating an object:</u>

```
union union_example u;
```

<u>Access(read/write):</u>

u.integer
u.decimal
u.name

MNNIT COMPUTER CODING CLUB

WHAT'S NEXT?