Day 2

Constant

```
int *ptr
```

- In above statement, ptr is non constant pointer variable, which can store address of non constant integer variable.

```
int *ptr = NULL;
int num1 = 10;
ptr = &num1;
*ptr = 50;
printf("Num1    :       %d\n", *ptr);

int num2 = 20;
ptr = &num2;
*ptr = 60;
printf("Num2    :       %d\n", *ptr);
```

```
const int *ptr
```

- In above statement, ptr is non constant pointer variable which can store address of constant integer variable.

```
const int *ptr = NULL;
const int num1 = 10;
ptr = &num1;
//*ptr  = 50;   //Not OK
printf("Num1    :       %d\n", *ptr);   //OK    :       10

const int num2 = 20;
ptr = &num2;
//*ptr  = 60;   //Not OK
printf("Num2    :       %d\n", *ptr);//OK :     20
```

```
int const *ptr
```

- Above statement is 100% same as : "const int *ptr".

```
const int const *ptr
```

- Above statement is 100% same as : "const int *ptr" or "int const *ptr".
- In this compiler will generate warining: "Duplicate 'const' declaration specifier"

```
int *const ptr
```

- In above statement, ptr is constant pointer variable which can store address of any non constant integer variable.

```
int num1 = 10;
int *const ptr = &num1;
*ptr = 50;
printf("Num1    :       %d\n", *ptr);

int num2 = 20;
//ptr = &num2;  //Not OK
```

```
int *ptr const
```

- Above syntax is invalid.

```
const int *const ptr
```

- In above statement, ptr is constant pointer variable which can store address of constant integer variable.

```
const int num1 = 10;
const int *const ptr = &num1;
//*ptr = 50;    //Not OK
printf("Num1    :       %d\n", *ptr);

const int num2 = 20;
//ptr = &num2;  //Not OK
```

```
int const *const ptr
```

- It is same as "const int *const ptr"

Array

- It is derived data type.
- Array is linear data structure/collection which is used to store elements of same type in continuous memory location.
- If we want to access elements from array then we should use integer index and sub script/index operator([]).
- Array index always begin with 0.
- Types of array:
    1. Single dimensional array
    2. Multi dimensional array
- Syntax:

```
int arr[ ]; //Not OK

int arr[ 5 ];   //OK

#define SIZE    5
int arr[ SIZE ];    //OK

int size = 5;
int arr[ size ];    //Ok

int arr[ ] = { 10, 20, 30 };    //OK

int arr[ 3 ] = { 10, 20, 30 };  //OK

int arr[ 5 ] = { 10, 20, 30 };  //OK

int arr[ 5 ] = { 0 };   //Ok

int arr[ 5 ] = {  };    // OK

int arr[  ] = {  }; //OK
```

- Value stored inside data structure is called element.
- Array name represents address of first element.
- Array of array is called multi dimensional array.
- Syntax

```
//int arr[ 2 ][ 3 ];    //OK
//int arr[ 2 ][ 3 ] = { {1,2,3},{4,5,6}};       //OK
//int arr[ 2 ][  ] = { {1,2,3},{4,5,6}};        //Not OK
//int arr[  ][ 3 ] = { {1,2,3},{4,5,6}};        //OK
//int arr[ ][  ] = { {1,2,3},{4,5,6}};  //Not OK
```

- Advantage(s)

1. If we know index of element then we can access elements of array randomly.

- Limitations

1. It requires continous memory
2. We can not resize array dynamically.
3. Insertion and removal of element from array is time consuming task.
4. Checking array bounds(lower and higher index) is a job of programmer.
5. Using assignment operator, we can not copy state/value of array into another array.

void pointer

- A pointer, which can store address of any type of variable is called void pointer.
- It is also called as generic pointer.

```
void *ptr = NULL;

int num1 = 10;
int *ptrNum1 = &num1;   //Ok
ptr = &num1;    //OK

double num2 = 20;
double *ptrNum2 = &num2;    //Ok
ptr = &num2;    //OK
```

- void pointer can strore address of any object/variable but it can not used to do dereferencing. If we want to do dereferencing then we should use specific pointer.

```
int number = 10;
void *ptr1 = &number;
//printf("Number        :       %d\n", *ptr1);  //Not OK

int *ptr2 = ( int*)ptr1;
printf("Number  :       %d\n", *ptr2);  //OK
```

Dynamic Memory Management.

- If we want to manage memory dynamically then we should use functions declared in header file.
- Following are the functions declared in header file:

1. void* malloc(size_t size);
2. void* calloc(size_t count, size_t size);
3. void* realloc(void *ptr, size_t size);
4. void free(void *ptr);

malloc

- It is a function declared in header file
- Syntax:

```
typedef unsigned int size_t;
void* malloc( size_t size );
```

- It is used to allocate memory on heap section only.
- We can use it to allocate memory for single variable as well as array. But it is designed to allocate memory for single variable.
- Everything on heap section is anonymous.
- If we allocate memory using malloc then memory gets initialized with garbage value.
- If malloc function fails to allocate memory then it returns NULL.

free

- It is a function declared in header file.
- Syntax:

```
void free( void *ptr );
```

- It is used to deallocate memory.
- Using free function, we can deallocate memory which is allocated on heap section only.
- If ptr is a NULL pointer, no operation is performed.

```
int *ptr = NULL;    //ptr is NULL pointer
free( ptr );//OK :no operation is performed.
```

- If pointer contains address of deallocated memory then such pointer is called dangling pointer. If we want to avoid it then we should store NULL value inside it.

calloc

- It is a function declared in header file.
- Syntax: void* calloc(size_t count, size_t size);
- We can use it to allocate memory for single variable as well as array. But it is designed to allocate memory for array.
- If we allocate space using calloc then memory gets initialized with 0.
- If calloc function fails to allocate memory then it returns NULL.

Memory allocation and deallocation for multidimensional array

- If we know value of row and col at compile time:

```
int arr[ 2 ][ 3 ];
```

- If we know only value of row at compile time:

```
int *arr[ 3 ];
int col;
printf("Column  :   ");
scanf("%d",&col);
for( int i = 0; i < 3; ++ i )
    arr[ i ] = calloc( col, sizeof(int));
```

- If we dont know value of row and col at compile time:

```
//Memory Allocation
int **ptr = (int**)malloc(3 * sizeof(int*));
for( int i = 0; i <3; ++ i )
    ptr[i] = (int*)malloc( 4 * sizeof(int));

//Memory Deallocation
for( int i = 0; i <3; ++ i )
    free( ptr[i] );
free( ptr );
ptr = NULL;
```

realloc

- It is a function declared in header file.
- Syntax: void* realloc(void *ptr, size_t newSize);
- We can use it to resize/reallocate memory.
- If first argument of realloc is NULL then it behaves like malloc.
- If realloc function fails to allocate memory then it returns NULL.

Function Pointer

- If pointer stores address of function then such pointer is called function pointer.

```
double (*ptr)( int,float,double) = NULL;
ptr = &sum;
double result = (*ptr)( 10, 20.2f,30.5);
```

```
double (*ptr)( int, float, double ) = NULL;
ptr = sum;
double result = ptr( 10, 20.2f,30.5);
```

```
typedef double (*FunPtr)( int, float, double) ;
FunPtr ptrSum = sum;
double result = ptrSum( 10, 20.2f,30.5);
```

- Using function pointer, we can pass function as a argument to the another function.
- Using function pointer, we can reduce maintenance of the application.

Day 3

qsort

- Arranging data eirther ascending or descending order is called sorting.
- qsort() is library function which is available in header file.
- Syntax: void qsort (void *base, //Address of array size_t nel, //count of elements size_t width, //size of single ele. int (compar)(const void,const void*));
- The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

Function

- It is a C language feature which is used to achive reusability.
- Due to reusability,we can reduce:
    1. Development time
    2. Development cost
    3. Developer efforts
- Syntax: ReturnType FunctionName (ParameterType(s) ParameterName(s)) {

    ```
    //Function body
    return returnValue;
    ```

    }
- In C language, we can not define function inside another function and structure.
- gcc compiler allows us to define function inside another function. In other words, gcc allows us to define local function.
- Function implementation is also called as function definition.
- In C language, all the functions are global.
- If want to use function then we must call it from another function.

- If we want to use function before definition then we must provide its declaration to the compiler.
- We can declare function locally as well as globally.

```
int main( void )        //Calling Function
{
        void sum( void );       //Function Declaration
        sum( ); //Function call
        return 0;
}
//Function Defintion
void sum( void )        //Called Function
{
        int a = 10;
        int b = 20;
        int c = a + b;
        printf("Result  :       %d\n",c);
}
```

- If we implement function above calling function then it is considered as declaration as well as definition.
- Without definition, if we try to use any element then linker generates error.

```
int main( void )
{
        void sum( void );

        sum( ); //Linker Error
        return 0;
}
```

- "/usr/include" directory is called standard directory for header files.
- If we include header file in between "<" and ">" the preprocessor try to locate header file in standard directory only.
- e.g #include<abc.h>
- If we include header file in double quotes then preprocessor try to locate header file in current directory. If it is not found then it try to locate it inside standard directory.
- e.g #include"abc.h"
- If we want to expand contents of header file only once then we should use header guard. It is also called as include guard.
- Syntax

```
#ifndef FILENAME_H
#define FILENAME_H
//TODO : Declaration
#endif
```

Makefile

- "Calculator.h"

```
int sum( int num1, int num2 );
```

- "Calculator.c"

```
int sum( int num1, int num2 )
{
    int result = num1 + num2;
    return result;
}
```

- "Main.c"

```
#include<stdio.h>
#include"Calculator.h"
int main( void )
{
    int result = sum( 10, 20);
    printf("Result  :   %d\n", result);
    return 0;
}
```

- "Makefile"

```
Main.out:Calculator.o Main.o
    gcc Main.o Calculator.o -o Main.out
Main.o:Calculator.h Main.c
    gcc -c Main.c
Calculator.o:Calculator.h Calculator.c
    gcc -c Calculator.c
```

- How to execute it?

```
make
```

- In C, we can pass argument to the function using 2 ways:
    1. By value
    2. By address

- if we want to reflect change made inside called function, into calling function then we should pass argument to the function by address.

```
//In params : x and y
//out params : ptrSum, ptrSub
void calculate( int x, int y, int *ptrSum, int *ptrSub )
{
        *ptrSum = x + y;
        *ptrSub =  x - y;
}
int main( void )//Calling function
{
        int a = 10;
        int b = 20;
        int resSum, resSub;
        calculate( a, b, &resSum, &resSub );
        printf("Sum     :       %d\n",resSum);
        printf("Sub     :       %d\n",resSub);
        return 0;
}
```

Storage Classes

- There are 4 storage classes in C
    1. auto
    2. static
    3. register
    4. extern
- It decides scope and lifetime of the variable.
- Scope decide area / region where we can access element.
- Types of scope:
    1. Block scope
    2. Function scope
    3. Prototype scope
    4. File Scope
    5. Program Scope

```
int num1 = 10;  //Program scope
static int num2 = 20;   //File Scope
void sum( int num3, int num4 );//prototype
int main( void )
{
    int num5 = 30;  //Function Scope
    {
        int num6 = 40;  //Block Scope
    }
    return 0;
}
```

- Lifetime of the variable decides time/duration ie. how long variable can be exist inside memory.
- Types
    1. automatic lifetime
    2. static lifetime
    3. dynamic lifetime
- A variable declared inside block/function is called local variable.
- Local variable get space on stack segment.
- Default value of local variable is garbage.
- Local variable get space per function call.
- If we give call to the function the local variable get space automatically. If function returns control back to the calling function then its space gets deallocated automatically. Hence local variable is also called as automatic variable and its default storage class is auto.
- static variable do not get space per function call rather it gets space one per application during application loading/ before calling main function.
- Uninitialied(local & global)static variable get space on BSS(Block Started by Symbol) segment and initialized(local & global) static variable get space on data segment.
- Default value of static variable is 0.
- If we want to share value of local variable between function call then we should declare local variable static.

```
int num1 = 10;
static int num2 = 20;
int main( void )
{
        return 0;
}
```

- In above code, we can access num1 anywhere inside same file and Using extern inside different file.
- We can access num2 inside same file only.
- We can declare global variable and function static. If it is static then we can not access it in different file.
- If function is static then local variables are not considered static.

String

- Data type of variable describe 3 things:
    1. Memory
    2. Nature
    3. Operation
- In C/C++, string is not a built in data type.
- String is specified in double quotes
- eg. "SunBeam";

```
#include<stdio.h>
#include<string.h>
int main( void )
```

```
{
        char name[ ] = "DAC";

        size_t length = strlen( name );
        printf("Length  :       %lu\n",length); //3

        size_t size = sizeof( name );
        printf("Size    :       %lu\n",size);   //4
        return 0;
}
```

- String size = String length + 1('\0')
- '\0' is called null character which is used to represent end of the string.
- String is array of character which ends with '\0' character.