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.