Chapter 5 :- Pointers

Declaration

Pointer declaration is similar to other type of variable except asterisk (*) character before pointer variable name.

Here is the syntax to declare a pointer

data_type *pointer_name;

For Example :-

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

```
{
        int x=20;       //int variable
        int *ptr;       //int pointer declaration
        ptr=&x;                 //initializing pointer
        printf("Memory address of x: %p\n",ptr);
        printf("Value x: %d\n",*ptr);
        return 0;
}
```

********** END OF PROGRAM**********

Pointer Arithmetic

We can perform arithmetic operations on the pointers like addition, subtraction, etc. However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer. In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:

1.Increment 2.Decrement 3.Addition 4.Subtraction 5.Comparison

Incrementing Pointer in C

If we increment a pointer by 1, the pointer will start pointing to the immediate next location.

This is somewhat different from the general arithmetic since the value of the pointer will get increased by the size of the data type to which the pointer is pointing.

Decrementing a Pointer

The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type

Pointer Comparisons

Pointers may be compared by using relational operators, such as ==, <, and >.

If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.

Pointer Subtraction

Like pointer addition, we can subtract a value from the pointer variable. Subtracting any number from a pointer will give an address.

Pointer Addition

Pointer addition, we can add a value from the pointer variable. Adding any number from a pointer will give an address.

Relation Between Array & Pointer

Consider the below array declaration,

int arr[5];

It declares an integer array with a capacity of five elements. To access any element of the given array we use array index notation. For example to access zeroth element we use arr[0], similarly to access fifth element we use arr[4].

Let us transform things in the context of pointers.

int * ptr = &arr[0];

The above statement declares an integer pointer pointing at zeroth array element.

In C programming, array exhibits a special behaviour. Whenever you refer an array name directly, is behaves a pointer pointing at zeroth array element. Which means both of the below statements are equivalent.

int * ptr = &arr[0];

int * ptr = arr;

This special behaviour of array allows many interesting things to happen. Things such as you can interchangeably use array and pointers. You can also use array name as a pointer pointing at zeroth element. In context of array and pointer, all of the following statements are valid.

int arr[5];

int * ptr = arr;

arr[0] = 10; // Stores 10 at 0th element of array

ptr[1] = 20; // Stores 20 at 1st element of array

ptr = arr; // ptr and arr both points to 0th element of array

*ptr = 100; // Stores 100 at 0th element of array (Since ptr points at arr[0])

*arr = 100; // Stores 100 at 0th element of array

For Example :-

```
#include <stdio.h>
int main() {
  int i, x[6], sum = 0;
  printf("Enter 6 numbers: ");
  for(i = 0; i < 6; ++i) {
  // Equivalent to scanf("%d", &x[i]);
      scanf("%d", x+i);
  // Equivalent to sum += x[i]
      sum += *(x+i);
```

```
  }
   printf("Sum = %d", sum);
   return 0;
}
```

Output :- Enter 6 numbers: 2 3 4 4 12 4

Sum = 29

********** END OF PROGRAM**********

Functions & Pointers

Syntax of function pointer

return type (*ptr_name)(type1, type2…);

For example:

int (*ip) (int);

In the above declaration, *ip is a pointer that points to a function which returns an int value and accepts an integer value as an argument.

float (*fp) (float);

In the above declaration, *fp is a pointer that points to a function that returns a float value and accepts a float value as an argument.

We can observe that the declaration of a function is similar to the declaration of a function pointer except that the pointer is preceded by a. So, in the above declaration, fp is declared as a function rather than a pointer.

For Example :-

```
#include<stdio.h>
int subtraction (int a, int b) {
   return a-b;
}
int main() {
   int (*fp) (int, int)=subtraction;
   int result = fp(5, 4); //Calling function using function pointer
   printf(" Using function pointer we get the result: %d",result);
   return 0;
}
```

Output

Using function pointer we get the result: 1

**********END OF PROGRAM **********

Dynamic Memory Mangement

Dynamically allocate memory in your C program using standard library functions: malloc(), calloc(), free() and realloc().

Sr.No	Function & Description
1	void *calloc(int num, int size); This function allocates an array of num elements each of which size in bytes will be size.
2	void free(void *address); This function releases a block of memory block specified by address.
3	void *malloc(size_t size); This function allocates an array of num bytes and leave them uninitialized.
4	void *realloc(void *address, int newsize); This function re-allocates memory extending it upto newsize.

Types Of Pointers

There are eight different types of pointers which are as follows −

1. Null pointer
2. Void pointer
3. Wild pointer
4. Dangling pointer
5. Complex pointer
6. Near pointer
7. Far pointer
8. Huge pointer

1. **Null Pointer** :- You create a null pointer by assigning the null value at the time of pointer declaration.This method is useful when you do not assign any address to the pointer. A null pointer always contains value 0.
2. **Void Pointer** :- It is a pointer that has no associated data type with it. A void pointer can hold addresses of any type and can be typecast to any type. It is also called a generic pointer and does not have any standard data type. It is created by using the keyword void.
3. **Wild Pointer** :- Wild pointers are also called uninitialized pointers. Because they point to some arbitrary memory location and may cause a program to crash or behave badly. This type of C pointer is not efficient. Because they may point to some unknown memory location which may cause problems in our program.
4. **Dangling Pointer** :- A pointer that points to a memory location that has been deleted is called a dangling pointer.

5. Complex Pointer :- Before knowing how to read complex pointers then you should first know associativity and precedence.

6. Associativity : Order operators of equal precedence within an expression are employed.

7. Precedence : Operator precedence describes the order in which C reads expressions.

8. Near Pointer :- Near pointer means a pointer that is utilized to bit address of up to 16 bits within a given section of that computer memory which is 16 bit enabled. It can only access data of the small size of about 64 kb within a given period, which is the main disadvantage of this type of pointer.

9. Far Pointer :- A far pointer is typically 32 bit which can access memory outside that current segment. To utilize the far pointer, the compiler allows a segment register to save segment address, then another register to save offset inside the current segment.

10. Huge Pointer :- Same as far pointer huge pointer is also typically 32 bit which can access outside the segment. A far pointer that is fixed and hence that part of that sector within which they are located cannot be changed in any way; huge pointers can be.