Chapter 6 Pointers

Basics of pointer:

A pointer is a variable that contains address of another variable. Pointers contain memory address manipulate data stored in memory. It is called pointer because it points to a particular location in memory by storing address of that location.

Declaration of Pointer:

Syntax: data_type *pt_name;

Example: int *p; //pointer to integer type
float *q; //pointer to float type
char *c; //pointer to character type

Initialization of pointer:

```
int x;     //Declares variable x
int *p;   //Declares pointer variable p
x=10;
p=&x;  // Initializes p with address of x
```

Example:

```
void main()
{
        int a=10, *p;
        p = &a; \\ Assign memory address of a to pointer variable p
        printf(“%d %d %d”, a, *p, p);   //* is indirection or dereferencing operator which returns
                                            value stored at that memory address.
}
```

Output:

10 10 5000

Example:

```
#include <stdio.h>
int main(){
        int* pc;
        int c;
        c=22;
        printf("Address of c:%u\n",&c);
        printf("Value of c:%d\n\n",c);
        pc=&c;
        printf("Address of pointer pc:%u\n",pc);
        printf("Content of pointer pc:%d\n\n",*pc);
        c=11;
```

```
        printf("Address of pointer pc:%u\n",pc);
        printf("Content of pointer pc:%d\n\n",*pc);
       *pc=2;
        printf("Address of c:%u\n",&c);
        printf("Value of c:%d\n\n",c);
        return 0;
}
```

Output:

Address of c: 2686784
Value of c: 22
Address of pointer pc: 2686784
Content of pointer pc: 22
Address of pointer pc: 2686784
Content of pointer pc: 11
Address of c: 2686784
Value of c: 2

Pointer to Pointer:

Addition of pointer variable stored in some other variable is called pointer to pointer variable.

Or Pointer within another pointer is called pointer to pointer.

Syntax:-

```
data type **p;

int x=22;

int *p=&x;

 int **p1=p;

printf(“value of x=%d”,x);

printf(“value of x %d”,*p);

printf(“Value of x= %d”

 printf(“value of x =

%d”,*&x);

printf(“value of x

%d”,**p1);

printf(“value of p=

%u”,&p);

printf(“address of p=

%u”,p1);
```

```
printf(“address of x=

%u”,p);

printf(“address of p1=

%u”,&p1);

 printf(“value of p=%u”,p);

printf(“value of p=

%u”,&x);
```

Pointer arithmetic

As the variables store the address value, which are number we can do arithmetic operations on pointer variables.

So valid arithmetic operations on pointers are:

- Increment
- Decrement
- Adding integer number to pointer variable.
- Subtracting integer number from pointer variable.
- Subtracting one pointer from other pointer if pointing to same array but, following arithmetic operational is not valid.
- Addition of two pointers.
- Multiplication operation with any number.
- Division operation with any number.

Pointer arithmetic example:

Operation	Expression	Initial value of ptr	New value of ptr for data type		
			character	Integer	Float
Increment	Ptr++	100	101	102	104
Decrement	Ptr-	100	99	98	96
Adding integer number	Ptr=ptr+5;	100	105	110	120
Subtracting integer number	Ptr=ptr-3	100	97	94	88

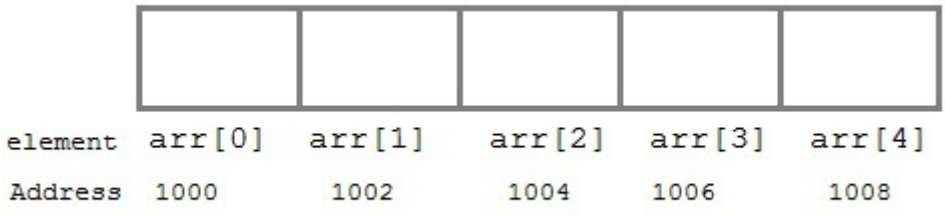
Subtracting Of one pointer from another	Ptr=ptr-1; Ptr2;	Ptr1=105 Ptr2=100	5	5	5

Pointer and Array

When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array. Base address which gives location of the first element is also allocated by the compiler.

Suppose we declare an array arr,

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



Here variable arr will give the base address, which is a constant pointer pointing to the element, arr[0]. Therefore arr is containing the address of arr[0] i.e 1000.

We can declare a pointer of type int to point to the array arr.

int *p;

p = arr;

or p = &arr[0]; //both the statements are equivalent.

Now we can access every element of array arr using p++ to move from one element to another.

Pointer to Array:
As studied above, we can use a pointer to point to an Array, and then we can use that pointer to access the array. Lets have an example,

```
int main( )

{
  /*Pointer variable*/
  int *p;

  /*Array declaration*/
  int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;

  /* Assigning the address of val[0] to pointer: 88820*/
  p = &val[0];

  for ( int i = 0 ; i <= 6 ; i++ )
  {
    printf("val[%d]: value is %d and address is %u", i, *p, p);
    p++;
  }
  return 0;
}
```

output:

val[0]: value is 11 and address is 88820

val[1]: value is 22 and address is 88824

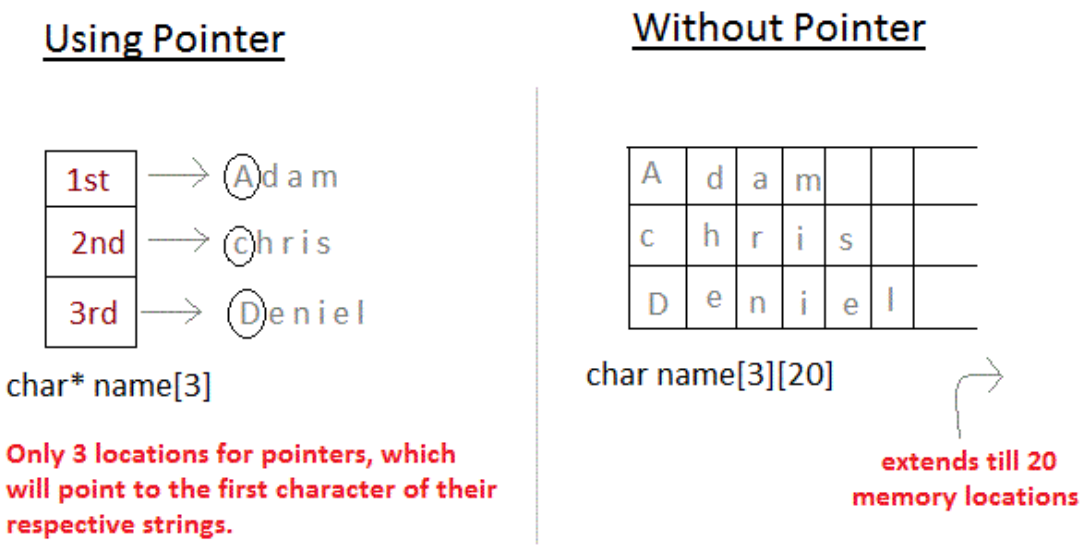
val[2]: value is 33 and address is 88828
val[3]: value is 44 and address is 88832
val[4]: value is 55 and address is 88836
val[5]: value is 66 and address is 88840
val[6]: value is 77 and address is 88844

Array of Pointers:

We can also have array of pointers. Pointers are very helpful in handling character array with rows of varying length.

```
char *name[3]={

              "Adam",
              "chris",
              "Deniel"
            };
//Now see same array without using pointer
char name[3][20]= {
                    "Adam",
                    "chris",
                    "Deniel"
                  };
```



In the second approach memory wastage is more, hence it is prefered to use pointer in such cases.

Function returning Pointers:

A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function, hence if you return a pointer connected to a local variable, that pointer be will pointing to nothing when function ends.

```
#include <stdio.h>
int* larger(int*, int*);
void main()
{
 int a=15;
 int b=92;
 int *p;
 p=larger(&a, &b);
```

```
 printf("%d is larger",*p);
}

int* larger(int *x, int *y)
{
 if(*x > *y)
  return x;  // return address of x;
 else
  return y; // return address of Y;
}
```

Output:
92 is larger

c program for swapping of two numbers using function with two pointers as arguments.

```
#include<stdio.h>

void swap(int *num1, int *num2);

void main() {
   int x, y;

   printf("\nEnter First number : ");
   scanf("%d", &x);

   printf("\nEnter Second number : ");
   scanf("%d", &y);

   printf("\nBefore Swaping x = %d and y = %d", x, y);
   swap(&x, &y); // Function Call - Pass By Reference

   printf("\nAfter Swaping x = %d and y = %d", x, y);
   getch();
}

void swap(int *num1, int *num2) {
   int temp;
   temp = *num1;
   *num1 = *num2;
   *num2 = temp;
}
```

Output :

```
Enter First number : 12
Enter Second number : 21

Before Swaping x = 12 and y = 21
After Swaping x = 21 and y = 12
```