Experiment no.4

Pointers

1. Pointers in C

A pointer in C holds address of another variable ie. Direct address of memory location. A pointer must be declared before using it to store any address.

Syntax

```
type *variable_name;
```

Here type must be a valid C variable type and variable_name is name of pointer variable.

Example

```
int    *ip;    /* pointer to an integer */ // Also means value at
address contained in ip is an integer

double *dp;    /* pointer to a double */ // Also means value at
address contained in dp is a double

float  *fp;    /* pointer to a float */ // Also means value at
address contained in fp is a float

char   *ch     /* pointer to a character */ // Also means value at
address contained in ch is a character.
```

Note : & is called the address of operator. So if we declare a variable x then &x would denote address of variable x. Similarly printing the value of *(&x) is equivalent to printing value of x.

Example 1

C program to show the use of pointers.

```
#include <stdio.h>
int main ()
{
   int  var = 20;   /* variable declaration */

   int  *ip;        /* pointer variable */
```

```
   ip = &var;       /* store address of var in pointer variable*/


   /* Address by using address of Operator */

   printf("Address of var variable in Hexadecimal Format: %x\n", &var
);

   printf("Address of var variable as integer: %u\n", &var  );


   /* address stored in pointer variable */

   printf("Address stored in ip variable in Hexadecimal Format:
%x\n", ip );

   printf("Address stored in ip variable as integer: %u\n", ip );


   /* access the value using the pointer */

   printf("Value of *ip variable: %d\n", *ip );


return 0;

}
```

2. Null Pointers

A pointer that is assigned a NULL value is called a null pointer.It is good practice to assign NULL value to a pointer variable in case there is no exact address to be assigned.

Example 2

C program to show NULL pointer is a constant with a value of zero.

```
   #include <stdio.h>

   int main () {
```

```
   int  *ptr = NULL;

   printf("The value of ptr is : %x\n", ptr  );  /* %x gives
memory location in Hex*/

   return 0;
}
```

Note: We can check for null pointers using these if statements.

```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

3. Pointer to Pointer

A pointer contains address of another variable. This variable might itself be another pointer. So we now have a pointer that contains another pointer's address.

Example 3

C program to illustrate pointer to pointer

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

      int  i = 3, *j, **k ;
      j = &i ;
      k = &j ;
      printf ( "\nAddress of i = %u", &i ) ;
      printf ( "\nAddress of i = %u", j ) ;
      printf ( "\nAddress of i = %u", *k ) ;
      printf ( "\nAddress of j = %u", &j ) ;
      printf ( "\nAddress of j = %u", k ) ;
      printf ( "\nAddress of k = %u", &k ) ;
      printf ( "\nValue of j  = %u", j ) ;
      printf ( "\nValue of k  = %u", k ) ;
      printf ( "\nValue of i  = %d", i ) ;
      printf ( "\nValue of i  = %d", * ( &i ) ) ;
      printf ( "\nValue of i  = %d", *j ) ;
      printf ( "\nValue of i  = %d", **k ) ;

   return 0;
}
```

4. **Call by Value And Call by Reference for a function in C**

Call by Value

Here actual value of the argument passed to function is copied into the formal parameter defined in function prototype. So changes made to parameters inside function definition has no effect on the argument.

Example 4

C program to demonstrate call by value

```
#include<stdio.h>


void change(int num)

{
   printf("Before adding value inside function num = %d \n",num);

    num=num+100;

    printf("After adding value inside function num = %d \n",num);

}



int main()
{

    int x=100;


    printf("Before function call x = %d \n", x);

    change(x);
   //passing value in function

    printf("After function call x = %d \n", x);

return 0;

}
```

Call by Reference

Here address of the argument passed to function is copied into the formal parameter defined in function prototype. So changes made to parameters inside function definition affects the actual argument through its address.

Example 5

C program to demonstrate call by Reference

```
#include<stdio.h>

void change(int *num)
{

      printf("Before adding value inside function num = %d
\n",*num);

      (*num) += 100;

      printf("After adding value inside function num = %d \n",
*num);

}


int main()
{

      int x=100;

printf("Before function call x = %d \n", x);

      change(&x);     //passing reference in function

      printf("After function call x = %d \n", x);


return 0;

}
```

5. **Pointer Arithmetic in C**

Addition to or Subtraction from pointer

We can add a number or subtract a number from the pointer and since pointer is nothing but an address following formula shows how that address is incremented or decremented on performing additive or subtractive operation.

```
new_address= current_address + (number * size_of(data type to
which pointer points))

new_address= current_address - (number * size_of(data type to
which pointer points))
```

Note: Incrementing or Decrementing pointer is same as adding or subtracting 1 from pointer.

Example *6*

C program to show pointer arithmetic

```
#include<stdio.h>

int main()

{
   int number=50;
   char var = 'A';

   float real=9.0;

   int *p;//pointer to int

   char *q;//pointer to char

   float *flt;//pointer to float


   p=&number;//stores the address of number variable

   q=&var;//stores address of var

   flt=&real;//stores address of real
```

```
      printf("\nsize of int :%u",sizeof(int));

      printf("\nsize of char :%u",sizeof(char));

      printf("\nsize of float :%u",sizeof(float));



      printf("\nAddress stored in  p variable is %u \n",p);

 p=p+3; //subtracting 3 from pointer variable

  printf("After adding 3 to pointer to integer: Address stored in
p variable is %u \n",p);



      printf("Address stored in q variable is %u \n",q);

 q=q-4; //subtracting 3 from pointer variable

  printf("After subtracting 4 from pointer to char: Address stored
in q variable is %u \n",q);



      printf("Address stored in flt variable is %u \n",flt);

     flt++;

      printf("After incrementing pointer to float : Address stored
in flt variable is %u \n",flt);

     flt--;

      printf("After decrementing pointer to float : Address stored
in flt variable is %u \n",flt);




return 0;
```

```
}
```

Subtraction of one pointer from other

One pointer variable can be subtracted from another provided both variables point to elements of the same array. The resulting value indicates the number of bytes separating the corresponding array elements as array elements are stored in contiguous memory locations.

Example 7

C program to show subtraction of a pointer from another

```
#include<stdio.h>
int main( )
{
      int  arr[ ] = { 10, 20, 30, 45, 67, 56, 74 } ;
      int  *i, *j ;

      i = &arr[1] ;
      j = &arr[5] ;
      printf ( "%d %d", j - i, *j - *i ) ;

return 0;
}
```

Pointer Comparison

If p1 and p2 are pointers to variables which are related to each other such as element of same array then p1 and p2 can be meaningfully compared.

Example 8

C program to show pointer comparison

```
#include <stdio.h>

int main () {

   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have address of the first element of array in pointer
*/
   ptr = &var[0];
   i = 0;
```

```
   while ( ptr <= &var[2] ) {

      printf("Address of var[%d] = %u\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* point to the next location */
      ptr++;
      i++;
   }

   return 0;
}
```

Note: Do not attempt the following operations on pointers… they would never work out.

(a) Addition of two pointers
(b) Multiplication of a pointer with a constant
(c) Division of a pointer with a constant

6. Pointers and Arrays

Two things are of interest when we talk about arrays and pointers together

(a) Array elements are always stored in contiguous memory locations.
(b) A pointer when incremented always points to an immediately next location of its type.

Example 9

C program to show array elements are stored in contiguous memory locations

```
#include <stdio.h>

int main( )
{
   int  num[ ] = { 24, 34, 12, 44, 56, 17 } ;
   int  i ;

   for ( i = 0 ; i <= 5 ; i++ )
   {
     printf ( "\nelement no. %d ", i ) ;
     printf ( "address = %u", &num[i] ) ;
   }
return 0;
}
```

Since array elements are stored in contiguous memory locations once we have the base address that is address of the first element of the array we can easily get to other elements by adding offset to the base address.

Example 10

C program to access array elements using the base address or pointer to the first element

```
#include <stdio.h>

int main( )
{
   int  num[ ] = { 24, 34, 12, 44, 56, 17 } ;
   int  i, *j ;
   j = &num[0] ;  /* assign address of zeroth element */
   for ( i = 0 ; i <= 5 ; i++ )
   {

      printf ( "\naddress = %u ", j ) ;
      printf ( "element = %d", *j ) ;
      j++ ;  /* increment pointer to point to next location */

   }

return 0;
}
```

7. **Passing an entire array to a function**

To pass an entire array to a function we need to pass the base address and the array size

Example 11

C program to pass an entire array to a function

```
#include <stdio.h>
void display( int  *j, int  n ) //function prototype declaration
int main( )
{
   int  num[ ] = { 24, 34, 12, 44, 56, 17 } ;
   dislpay ( &num[0], 6 ) ;
```

```
return 0;
}

void display( int  *j, int  n )
{

int  i ;

 for ( i = 0 ; i <= n - 1 ; i++ )
 {
   printf ( "\nelement = %d", *j ) ;
   j++ ;  /* increment pointer to point to next element */
 }

}
```

Note:

1. Mentioning name of the array we get its base addresss.
2. For an array named `num`, `*num` will refer to zeroth element of array.
3. For any array named `num` , C compiler converts `num[i]` to `*(num+i)`

Example 12

C program for accessing array elements in different ways

```
#include <stdio.h>
int main( )
{
      int  num[ ] = { 24, 34, 12, 44, 56, 17 } ;
      int  i ;

      for ( i = 0 ; i <= 5 ; i++ )
      {
            printf ( "\naddress = %u ", &num[i] ) ;
            printf ( "element = %d  %d ", num[i], *( num + i ) ) ;
            printf ( "%d  %d", *( i + num ), i[num] ) ;
      }

return 0;
}
```

Example 13

C program to swap 2 numbers using pointers without using 3rd variable

```
#include<stdio.h>

int main()
{
      int a=10,b=20,*p1=&a,*p2=&b;

      printf("Before swap: a=%d  b=%d",*p1,*p2);
      *p1=*p1+*p2;
      *p2=*p1-*p2;
      *p1=*p1-*p2;
      printf("\nAfter swap: a=%d  b=%d",*p1,*p2);

return 0;
}
```

8. Pointers and 2 Dimensional Array

For a 2 Dimensional array `s[m][n] ... s[0]` gives the address of zeroth 1 dimensional array of size n

`s[1]` gives the address of first 1 dimensional array of size n …. s[m-1] gives the address of (m-1)th dimensional array of size n.

Example 14

C program to give the address of each array in a array of arrays.

```
#include<stdio.h>
int main( )
{
      int  s[4][2] = {{ 1234, 56 },
                  { 1212, 33 },
                  { 1434, 80 },
                  { 1312, 78 } } ;
      int  i ;

      for ( i = 0 ; i <= 3 ; i++ )
      {
      printf ( "\nAddress of %d th 1-D array = %u", i, s[i] );

      }
return 0;
}
```

Example 15

C program to access each element of a 2 Dimensional array using pointers

```
#include<stdio.h>
int main( )
{
      int  s[4][2] = {{ 1234, 56 },
                  { 1212, 33 },
                  { 1434, 80 },
                  { 1312, 78 }} ;
      int  i, j ;

      for ( i = 0 ; i <= 3 ; i++ )
      {
            printf ( "\n" ) ;
            for ( j = 0 ; j <= 1 ; j++ )
                  printf ( "%d ", *( *( s + i ) + j ) ) ;  //printf
( "%d ", *( s[i]+ j ) )
      }

return 0;
}
```

9. Pointer to an array

A pointer to an array can hold address of an array. It can point to the whole array instead of only one element of an array. A noticeable difference can be seen between the two on doing pointer arithmetic.

Pointer to an array can be more seen in case of multidimensional arrays. An increment to pointer to array will give the address to next array whereas an increment to pointer to an element of array will give address of next array element.

Syntax

```
data_type (*var_name)[size_of_array];
```

For example

```
int (*ptr)[10]
```

Here `ptr` is a pointer pointing to array of 10 integers

Example 16

C program to understand difference between pointer to an integer and pointer to an array of integers

```
#include<stdio.h>

int main()
{
    // Pointer to an integer
    int *p;

    // Pointer to an array of 10 integers
    int (*ptr)[10];
    int arr[10];

    // Points to 0th element of the arr.
    p = arr;

    // Points to the whole array arr.
    ptr = &arr;

    printf("p = %u, ptr = %u\n", p, ptr);

    p++;
    ptr++;

    printf("p = %u, ptr = %u\n", p, ptr);

    return 0;
}
```

10. Dereferencing Pointer to an array

Pointer to an array points to an array, so on dereferencing it, we should get the array, and the name of array denotes the base address. So whenever a pointer to an array is dereferenced, we get the base address of the array to which it points.

Example 17

C program to illustrate dereferencing pointer to an array and size pointer to array

```
#include<stdio.h>
```

```
int main()
{
    int arr[] = { 3, 5, 6, 7, 9 };
    int *p = arr;
    int (*ptr)[5] = &arr;

    printf("p = %u, ptr = %u\n", p, ptr);
    printf("*p = %d, *ptr = %u\n", *p, *ptr);

    printf("sizeof(p) = %u, sizeof(*p) = %u\n", sizeof(p),
sizeof(*p));
    printf("sizeof(ptr) = %u, sizeof(*ptr) = %u\n", sizeof(ptr),
sizeof(*ptr));

return 0;
}
```

11. Array of Pointers

An array of pointers is a collection of addresses

Example 18

C program to illustrate array of pointers

```
#include<stdio.h>

int main( )
{
     int  *arr[4] ;  /* array of integer pointers */
     int  i = 31, j = 5, k = 19, l = 71, m ;
     arr[0] = &i ;
     arr[1] = &j ;
     arr[2] = &k ;
     arr[3] = &l ;

     for ( m = 0 ; m  <= 3 ; m++ )
     printf ( "%d ", * ( arr[m] ) ) ;

return 0;
}
```

12. Functions returning pointers

Example 19

C program to show a function returning a pointer to calling function. The function larger receives addresses of 2 variables and returns address of the larger variable.

```
#include<stdio.h>

int* larger(int*,int*); //Function prototype
int main( )
{

      int a = 10;
      int b = 20;
      int *p;
      p = larger(&a,&b); //Function call
      printf("Larger value is :%d",*p);


return 0;
}

int* larger(int *x, int *y)
{

      if(*x>*y)
        return x;
      else
        return y;
}
```

13. Exercise

1. Write a function in C using pointer parameters that compares 2 integer arrays to see whether they are identical. The function returns 1 if they are identical,0 otherwise

2. Using pointers write a function that receives a character string and a character as argument and deletes all occurrences of this character in the string. The function should return corrected string with no holes.

3. Write a function that receives a sorted array of integers and an integer value, and inserts the value in its correct place.

4. Write a program using pointers to read in an array of integers and print its elements in reverse order.

5. WAP in C using pointers to reverse a string.

6. Read about Pointers to Functions