**POINTERS**

**Pointer Variables:**

Pointer is a derived data type in C language. The variables that holds memory addresses are called pointer variables. Pointers are used in C program to access the memory and manipulate the address.

The pointer operators are & (Address operator) and \*(Dereferencing operator or Indirection operator).

**Declaration of Pointer Variables:**

Since pointer variables contain addresses that belong to a separate data type, they must be declared as pointers before we use them. The declaration of a pointer variable has the following form:

General Form:

datatype \*pt\_name;

This tells the compiler three things about the variable pt\_name.

1. The asterisk(\*) tells that the variable pt\_name is a pointer variable.
2. pt\_name needs a memory location.
3. pt\_name points to variable of type data type.

Different forms of declaring integer pointer variables:

Ex: int \*p;

int\* p;

int \* p;

Declaration of float pointer variable

Ex: float \*p;

Declaration of char pointer variable

Ex: char \*p;

Declaration of double pointer variable

Ex: double \*p;

**Initialization of Pointer Variables:**

int i, \*p1=&i; //Address of variable i is stored in pointer variable p1

**Variable Name Address Contents**

i 5600 35

p1 6000 5600

int \*p1=&i,i; ----------> Wrong

Pointer variables can be initialized with the values NULL and 0.

Ex: int \*p=NULL;

int \*p=0;

Same Pointer can point to different data variables in different statements.

Ex: int x, y, z,\*p;

p=&x; p=&y; p=&z;

Different pointers can be used to point to same data variable.

Ex: int x;

int \*p1=&x;

int \*p2=&x;

int \*p3=&x;

**Accessing a variable through its pointer**

Once a pointer has been assigned the address of a variable, the question remains as to how to access the value of the variable using the pointer. This is done by using \*(asterisk), usually known as the indirection operator.

int i,\*p,n; // p is a pointer variable and i is a integer variable

i=10;

p=&i; // p holds the address of variable i

n=\*p; // Returns the value of variable i

This is equivalent to n=\*&i; or n=i.

**Chain of Pointers**

Pointer can point to another pointer.

Ex:

main()

{

int x, \*p1,\*\*p2;

x=100;

p1=&x;

p2=&p1;

printf(“%d”,\*\*p2); //Output is 100

}

In this example, pointer variable p2 contains the address of the pointer variable p1, which points to the location that contains the desired value. This is known as multiple indirections.

**Pointer Expressions**

1. Pointer variables can be used in expressions.

Ex: int \*p1,\*p2;

y=\*p1 \* \*p2;

Sum=Sum+ \*p1;

Z=5\* -\*p2/ \*p1;

1. Short hand operators can be used with pointers.

Ex: int \*p1,\*p2;

p1++;

-p2; sum += \*p2;

**3)**  Pointers can be compared using relational operators.

Ex: p1>p2, p1==p2, p1!=p2

**Pointer Increments and Scale Factor**

When a pointer is incremented, its value is increased by the length of the datatype it points to. This length is called scale factor.

Ex:

int \*p; i=10;

p=&i;

p=p+1;

In this example, if the address value of i is 2800, then pointer variable p holds the address of i, i.e. 2800. If the pointer variable is incremented by 1, it becomes 2802.

**Pointers and Arrays**

When an array is declared, the compiler allocates the base address and sufficient amount of memory to contain all the elements in contiguous memory location. The base address is the location of the first element (index 0) of the array.

Ex: int \*p;

p=&a[0];

p+1=&a[1];

p+2=&a[2];

The address of an element is calculated using its index and scale factor of the data type.

Ex: Address of a[3]= Base Address+(3\*ScaleFactor of int)

Pointers can be used to access array elements instead of using array indexing.

Ex: \*(p+3) ----> It is similar to a[3].

**Tutorial Questions:**

1) Write a program to find the sum of all elements stored in an array using pointers?

main( )

{

int \*p,sum,i;

int a[5]={10,20,30,40,50};

i=0;

p=a;

while(a<5)

{

printf(“%d%d%u”, i,\*p,p);

sum=sum+\*p;

i++; p++;

}

printf(“%d”, sum);

}

2) Find the output of the following program: Assume the address values of m, ptr and y are

2000, 3000 and 4500 respectively.

main()

{

int m;

int \*ptr;

m=15;

ptr=&m;

y=\*ptr;

printf(“Value of m is %d\n\n”, m);

printf(“%d is stored at address %u \n”, m,&m);

printf(“%u%u \n”, ptr, &ptr);

printf(“%d is stored at address %u \n”, y, &y);

\*ptr=30;

printf(“\n Now m=%d\n”,x);

}

**Output:**

15

15 is stored at address 2000

2000 3000

15 is stored at address 4500

Now m=30

1. Write a program to perform the arithmetic operations using pointers

main( )

{

int a, b,\*p1,\*p2;

int sum,diff,prod,div;

a=200, b=10;

p1=&a;p2=&b;

sum=\*p1 + \*p2;

diff=\*p1 - \*p2;

prod=\*p1 \* \*p2;

div=\*p1 / \*p2;

printf(“%d%d%d%d”, sum,diff,prod,div);

}

**Output:**

210

190

2000

20

1. Evaluate the expressions in the following program:

main()

{

int a, b,\*p3,\*p4,i,j,k;

a=31,b=10,p3=&a,p4=&b;

i=\*p3 \* \*p4-6;

j=6 \* - \*p3/ \*p4+20;

a=\*p4+34; b=\*p4-6; k=\*p3 \* \*p4-8;

}

**Output:**

i=304

j=2

a=44

b=4

k=302

1. What are the advantages of Pointers?
2. Pointers can be used to return multiple values from User Defined Function.
3. It allows C language to support dynamic memory management.
4. It increases the execution speed and reduce the program execution time.
5. It provides an efficient tool for manipulating dynamic data structures such as structures, linked lists, queues, stacks and trees.