C Pointers

Topics:

- Introduction,
- Pointer to Pointer
- Pointer arithmetic
- Pointer to Function
- Call by value
- Call by reference

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer.

Consider the following example to define a pointer which stores the address of an integer.

```
int n = 10;

int * p = &n;

// Variable p of type pointer is pointing to the address of the variable n of type integer.
```

- Address of a variable is the memory location number which is allotted to the variable. The memory addresses are 0, 1, 2, 3... and so on up to the capacity of the memory.
- The address is normally displayed in hexadecimal form. Hexadecimal form is a representation of number somewhat similar to binary number system studied
- Here four binary digits are combined together to form a hexadecimal number. Pointers unlike other variables do not store values. As stated they store the address of other variables.

- It is already mentioned in the first statement that pointers are also variables. Hence, we can also have a pointer that is pointing to another pointer.
- Let us see the syntax of declaring pointers, operators required for pointers and some basic programs.

- **Syntax of pointer declaration:** Data_type *ptr_name;

  Wherein **"Data_type"** is the data type of the variable to which the pointer is supposed to point. If we want a pointer to point to an integer than, we need to have the data type of the pointer as "int", for a float type data pointer should also be of the "float" type and so on.

- The **"ptr_name"** is an identifier i.e. the name of the pointer. The same rules of identifiers apply to the pointer name as to any other variable declaration. The most important difference in the declaration of a pointer is the "*" sign given before the pointer name.
- Hence, according to the syntax seen above, if we want to declare a pointer for "int" type data then we can declare it as given in the example below:

int *p;

- Here, the pointer name is "p". Hence, "p" can be used as a pointer to point to any of the variable of type "int".

Referencing and De-referencing (Operators in Pointers)

There are two operators required in the pointer based programs viz. Address of operator ("&") and Value of operator (""). Let us see the use of these two operators in detail.

1. Address of operator ("&")

2. Value of operator ("*")

1. Address of operator ("&"):

- It is also called as the referencing operator.
- This operator returns the address of the variable associated with the operator.
- For eg., if we write "&x", it will return the address of the variable "x".
- Hence, if we have a pointer "p", which we want to point to a variable x, then we need to copy the address of the variable "x" in the pointer variable "p".
- This is implemented by the statement: p=&x;

i.e. the address of the variable "x" is copied into the pointer variable "p", hence the pointer "p" pointing to the variable "x".

2. Value of operator ("*"):

- It is also called as the de-referencing operator. This operator returns the value stored in the variable pointed by the specified pointer. For eg., if we write "*p", it will return the value of the variable pointed by the pointer "p".
- Hence, if we want the value of the variable pointed by the pointer "p" to be stored in a variable "y", then the expression can be written as:
  y=*p
- i.e. the value of the variable pointed by the pointer "p" is stored in the variable "y". Let us understand the use of these operators in some sample programs.

Programs Related to Pointers

Simple Referencing and De-referencing using Pointers

Program 1:

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
int a,*p;
clrscr();
a=125;
p=&a;
printf("%d\n",a);
printf("%x\n",p);
printf("%d\n",*p);
getch();
}
```

Output:

125

Address of variable a

125

Explanation:

The declaration of a pointer and a simple integer variable can be seen in the first statement i.e. "int a,*p.". The pointer is the variable "p", as it is associated with a * sign.

The second statement is very simple wherein the value 125 is put into the variable.

The third statement i.e. "p=&a" uses the address of operator. This statement assigns the address of the variable "a" to the pointer variable "p". As already seen address of a variable is the memory location (number), where it is stored.

Pointer to Pointer

Program 2:

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int a,*p,**p1;
   a=125;
   p=&a;
   p1=&p;

   printf("%d\n",a);
   printf("%x\n",p);
   printf("%x\n",p1);
   printf("%d\n",*p);
   printf("%x\n",*p1);
   printf("%d\n",**p1);
   getch();
   return 0;
}
```

Output:

```
125
70e6f054
70e6f058
```

125

70e6f054

125

Pointer Arithmetic

Program 3: Write the output of the following

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,*a1;
float b,*b1;
clrscr();
a1=&a;
b1=&b;
printf("%x\n%x\n",a1,b1);
a1++;
b1++;
printf("%x \n%x\n",a1,b1);
getch();
}
```

Output:

Address of variable a
Address of variable b
(Address of variable a)+2
(Address of variable b)+4

Program 4: Find the output

```
#include <stdio.h>
int main()
{
int x=20, y, *ip;
ip= & x;
y=(*ip)++;
printf("%d\n", y);
printf("%d\n", *ip);
y=++(*ip);
printf("%d\n", y);
printf("%d\n", *ip);
}
```

Output:

```
20
21
22
22
```

Passing Pointer to Functions

```
#include <stdio.h>

void sum(int *p1, int *p2); // Function declaration

void main() {
   int a, b;

   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b); // Reading input from the user

   sum(&a, &b); // Passing the addresses of `a` and `b` to the function
}

void sum(int *p1, int *p2) { // Function definition
   int c;
   c = *p1 + *p2; // Dereferencing pointers to get the values
   printf("The sum is equal to %d\n", c); // Printing the sum
}
```

Output:

Enter two numbers: 4

7

The sum is equal to 11

Explanation:

Note: To pass the parameters using pointers we pass the address of the variables using the address of operator ("&"). And hence to accept this address in the function we must have pointers in the argument list. Remember in this case when address of the variable is passed, the value of the actual parameters can be altered.

Call by Value

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

void main()
{
int a,b;
void swap (int a, int b);
clrscr();
printf("Enter two numbers:");
scanf("%d %d",&a,&b);
printf("The values of a and b in the main function before calling the swap function are %d and %d\n",a,b);
swap(a,b);
printf("The values of a and b in main function after calling the swap function are %d and %d\n",a,b);
getch();
}

void swap (int a, int b)
{
int temp;
temp=a;
a=b;
b=temp:
printf("The values of a and b in the swap function after swapping are %d and %d\n",a,b);
}
```

Output:

```
Enter two numbers: 4
5
The values of a and b in the main function before calling the swap function are 4 and 5
```

The values of a and b in the swap function after swapping are 5 and 4
The values of a and b in main function after calling the swap function are 4 and 5

Call by Reference

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
void swap (int *p1, int *p2);
clrscr();
printf("Enter two numbers:");
scanf("%d %d",&a,&b);

printf("The values of a and b in the main function before calling the swap function are %d and %d\n",a,b);
swap(&a,&b);
printf("The values of a and b in main function after calling the swap function are %d and %d\n”,a,b);
getch();
}
void swap (int *p1, int *p2)
{
int temp;
temp=*p1;
*p2=temp:
printf("The values of a and b in the swap function after swapping are %d and %d\n", *p1,*p2):
```

Output:

Enter two numbers: 4

5

The values of a and b in the main function before calling the swap function are 4 and 5
The values of a and b in the swap function after swapping are 5 and 4
The values of a and b in main function after calling the swap function are 5 and 4