By
Trupthi.M
Asst.Prof, IT Department,
CBIT, Gandipet, Hyderabad

Pointers in C Programming

Many C programming learner thinks that pointer is one of the difficult topic in C language but it's not completely true. It is difficult to understand in comparison of other secondary (array, structure etc.) data types available in C. Pointer in C seems to be difficult because it works directly with computer memory (RAM).

When we declare any variable in C, for example integer variable, it occupies a memory according to its size (here 2 bytes). As it occupies a memory so it's natural to have an address for this location so that it can be referenced later by CPU for manipulation. But before we begin with Pointer we must have some idea on the operators we are going to use in Pointer programming.

The '*' And '&' Operators

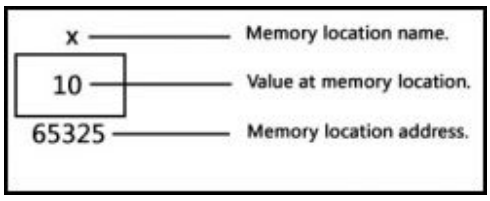
Take a look on the following declaration,

```
int x=10;
```

When we make such declaration how C compiler treat this statement:

1. Reserve memory space for this integer value.
2. Name this memory location as x.
3. Store the value 10 at this location.

This can be illustrated by a simple diagram:



Integer variable location in memory.

So when we declare an integer variable x and assigned value 10 then compiler occupies a 2 byte memory space at memory address 65325 and stored value 10 at that location. Compiler named this address x so that we can use x instead of 65325 in our program. But we can use address or variable in our program and both will point to the same value (example given below).

The memory address of a variable could differ from PC to PC and entirely depends on available free space in memory at time of program execution. As this differs from PC to PC thus we cannot rely on that address (numeric value representing variable address) and we cannot use this address in our program. But have you noticed one thing that address is an integer.

We can print address of any variable or function, following program shows how we can do that:

Address of variable example.

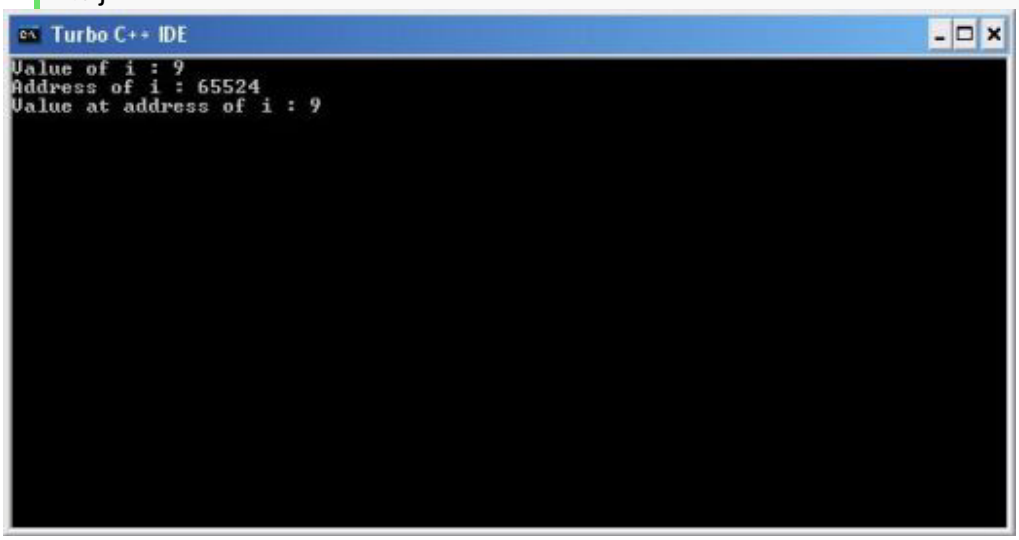
```
1.  #include<stdio.h>
2.  #include<conio.h>
3.
4.  void main()
5.  {
6.     int i=9;
7.     clrscr();
8.
9.     printf("Value of i : %d\n",i);
10.    printf("Address of i : %u",&i);
11.
12.    getch();
13. }
```

This is a very simple c program which prints value and address of an integer. But did you notice line no. 10 in above program? This line output the address of i variable and to get address of i variable we have used ampersand (&) operator. This operator is known as "**Address of**" operator and we already used this operator many times in our program, just recall **scanf** statement which is used to accept input from computer keyboard. So when we use ampersand operator (&i) before any variable then we are instructing c compiler to return its address instead of value.

Another operator we going to deal with in this entire pointer tutorial is "*" called "**Value at address**" operator. It is the same operator which we use for multiplication of numbers. As the name suggest, "**value at address**" operator returns value stored at particular address. The "**value at address**" operator also called **indirection** operator. Following example extends above C program and puts "**value at address**" operator in action.

Value at address (*) example

```
1.  #include<stdio.h>
2.  #include<conio.h>
3.
4.  void main()
5.  {
6.     int i=9;
7.     clrscr();
8.
9.     printf("Value of i : %d\n",i);
10.    printf("Address of i : %u\n",&i);
11.    printf("Value at address of i : %d",*(&i));
12.
13.    getch();
14. }
```



Output of "value at address" operator example.

Now in this program notice line no. 11 and use of '&' and '*' operators also see the output of program which prints value of variable i and *(&i) same. This is how these two operator works.

You must have clear idea of '&' and '*' operator usage before you jump into pointer because pointer deals with these to operator. If you find any difficulty then post your doubts and I will try to clear your doubts.

What is a Pointer?

A **pointer** is a secondary data type (also known as derived data type) in C. It is built from one of the primary data types available in C language. Basically pointer contains

memory address of other variable or function as their value. As pointer deals with memory address, it can be used to access and manipulate data stored in memory.

Benefits of using Pointer in C Program

Pointer is one of the most exciting features of C language and it has added power and flexibility to the language. Pointer is in C language because it offers following benefits to the programmers:

1. Pointers can handle arrays and data table efficiently.
2. Pointers support dynamic memory management.
3. Pointer helps to return multiple values from a function through function argument.
4. Pointer increases program execution speed.
5. Pointer is an efficient tool for manipulating structures, linked lists, queues stacks etc.

We can achieve these and much more benefits from pointer only if we can use it properly.

Pointer with an example

Since this tutorial is already become lengthy so I am going to give a small example on how we can use pointer in our C program. Of course we will discuss remaining part in my next pointer tutorial.

Syntax: datatype *pointer_name;

Example : int *iPtr;

float *fPtr;

Pointer Example

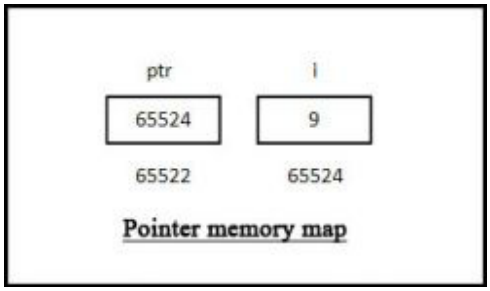
```
1. #include<stdio.h>
2. #include<conio.h>
3.
4. void main()
5. {
6.     int i=9, *ptr;
7.     ptr=&i;
8.     clrscr();
```

```
9.
10.    printf("Value of i : %d\n",i);
11.    printf("Address of i : %u\n",&i);
12.    printf("Value of ptr : %u\n",ptr);
13.    printf("Address of ptr : %u\n",&ptr);
14.    printf("Ptr pointing value : %d", *ptr);
15.
16.    getch();
17. }
```

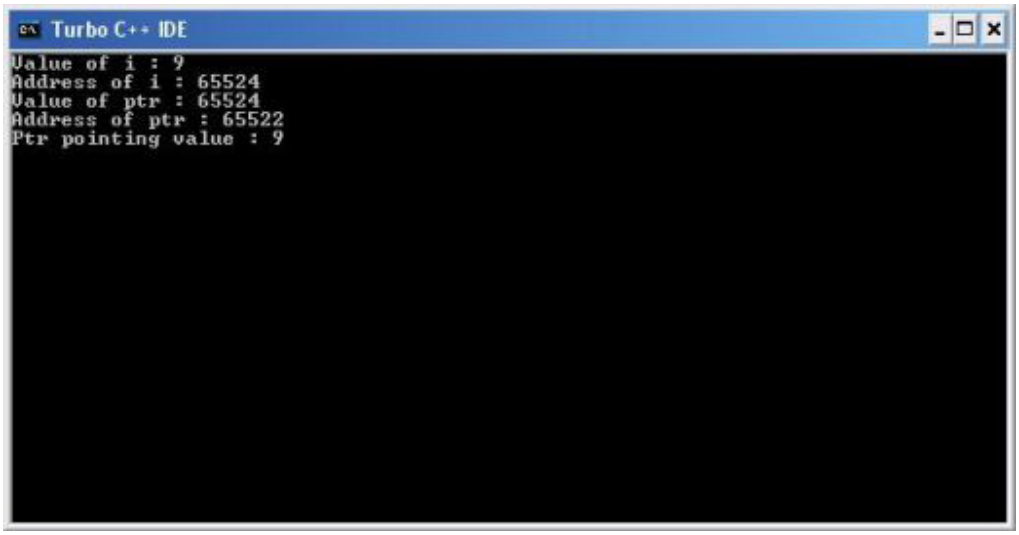
In the above example in line no. 6 we have declared an integer i and assigned 9 to it. Along with variable i, we have also declared an integer pointer. To declare a pointer of any data type we just need to put an * (asterisk) before variable name or identifier. In our example ptr is an integer variable and is capable of holding address of any integer variable.

Remember one thing integer pointer can hold only integer variable address, you cannot use integer pointer to hold address of float or char variable. To hold char or float variable address in a pointer you need to declare char or float pointer respectively.

In line no. 7 we assigned integer variable i's address to ptr, now ptr is pointing to i. Line no 10 prints value of i and line no. 11 prints address of i. Next line prints value stored in ptr pointer variable, line no. 13 prints address of ptr pointer. As pointer is different entity so it also requires space in memory. Memory occupied by a pointer depends on its data type. Integer pointer will occupy 2 bytes whereas character pointer will occupy 1 byte. Finally line no. 14 prints the value of address stored in ptr pointer i.e. value of i. And to print value of stored address in a pointer we need write *pointer variable name (in our example *ptr). A memory map will help you to understand this.



Pointer memory map.



Pointer example output.

C Programming – Function Pointer

Like C variables, function too has address and we can use this address to invoke function. So this tutorial is entirely devoted to function-pointer. But before we can call a function using we need to find out its address. So first we will see how to find memory address of a C function then we will call that function using its address.

Memory address of a C Function

```
1.  #include<conio.h>
2.  #include<stdio.h>
3.  int show();
4.
5.  void main()
6.  {
7.     clrscr();
8.     printf("Addres of show() : %u", show);
9.     show();
10.    getch();
11. }
12.
13. int show()
14. {
```

```
15.    printf("\n\nFunction called!");
16.    return 0;
17. }
```

So let's have a quick view of what this code does when you run this program. Line no 3 declares a function called "show" which has return type of integer. This is really a simple function which will print a simple message on calling. If you are new to **C function then you can learn about C function here**.

Line no 5-11 is main function block. In this code block only line no 8 is important for us now. This line contains a printf statement which prints address of user defined function called "show". To print address of "show()" we only need to mention function name only in printf statement as we did in line no 8. And rest of the code is simple enough to understand.

Function pointer in C Language

In the above program we learnt how to obtain address of a function and now we will use that logic to call function using pointer. So below is a C program which is same as the above program but has some extra statements. Go through this code cautiously and right after code you will find its explanation.

```
1.  #include<stdio.h>
2.  #include<conio.h>
3.  int show();
4.
5.  void main()
6.  {
7.  int (*fnPtr)();
```

```
8.  clrscr();
9.
10.    fnPtr=show;
11.
12.    printf("Address of function :%u",show);
13.    (*fnPtr)();
14.    getch();
15. }
16.
17. int show()
18. {
19. printf("\nFunction called using pointer!");
20. return 0;
21. }
```

We have declared prototype of a user defined function “show” which has integer return type. It’s a must condition for a UDF to return any data type because pointers can store address of data type (like char, int etc.). So function returning void will not work.

In our case we have function returning integer so our function pointer have to have integer type. Line no 7 declares an integer type function pointer. To declare a function pointer use the following syntax.

data-type (*function_name)();

You should write function name in bracket otherwise C compiler will throw error at compilation time as it happened in my case.

In line no 10 we are assigning address of show function to our function pointer and in line no 12 simply prints address of “show” function. Line no 13 calls our UDF using pointer. As you can see we just need to write function pointer name as we declared and it will call the function.

Line no 17-21 is “show()” declaration and I think rest is simple. But if find any difficulty then please ask me without hesitation.

Here is output of above program.

Usage of Function Pointer

Function pointer mainly used for writing memory resident programs like virus and anti virus.

C Programming - Handling of character string

In this tutorial you will learn about Initializing Strings, Reading Strings from the terminal, Writing strings to screen, Arithmetic operations on characters, String operations (string.h), Strlen() function, strcat() function, strcmp function, strcmpi() function, strcpy() function, strlwr () function, strrev() function and strupr() function.

A string is a sequence of characters. Any sequence or set of characters defined within double quotation symbols is a constant string. In c it is required to do some meaningful operations on strings they are:

- Reading string displaying strings
- Combining or concatenating strings
- Copying one string to another.
- Comparing string & checking whether they are equal
- Extraction of a portion of a string

Strings are stored in memory as ASCII codes of characters that make up the string appended with '\0'(ASCII value of null). Normally each character is stored in one byte, successive characters are stored in successive bytes.

Character	m	y		a	g	e		i	s
ASCII Code	77	121	32	97	103	10	32	105	115

Character		2		(	t	w	o	)	\0

ASCII Code	32	50	32	40	116	119	41	0	0

The last character is the null character having ASCII value zero.

Initializing Strings

Following the discussion on characters arrays, the initialization of a string must the following form which is simpler to one dimension array.

char month1[ ]={'j','a','n','u','a','r','y'};

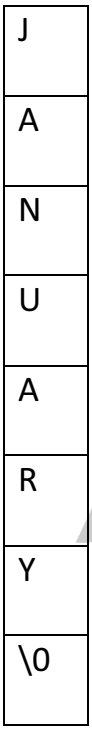
Then the string month is initializing to January. This is perfectly valid but C offers a special way to initialize strings. The above string can be initialized char month1[]="January"; The characters of the string are enclosed within a part of double quotes. The compiler takes care of string enclosed within a pair of a double quotes. The compiler takes care of storing the ASCII codes of characters of the string in the memory and also stores the null terminator in the end.

```
/*String.c string variable*/
#include < stdio.h >
main()
{
char month[15];
printf ("Enter the string");
gets (month);
printf ("The string entered is %s", month);
}
```

In this example string is stored in the character variable month the string is displayed in the statement.

printf("The string entered is %s", month");

It is one dimension array. Each character occupies a byte. A null character (\0) that has the ASCII value 0 terminates the string. The figure shows the storage of string January in the memory recall that \0 specifies a single character whose ASCII value is zero.



Character string terminated by a null character '\0'.

A string variable is any valid C variable name & is always declared as an array. The general form of declaration of a string variable is

Char string_name[size];

The size determines the number of characters in the string name.

Example:

char month[10];
char address[100];

The size of the array should be one byte more than the actual space occupied by the string since the complier appends a null character at the end of the string.

Reading Strings from the terminal:

The function scanf with %s format specification is needed to read the character string from the terminal.

Example:

```
char address[15];
scanf("%s",address);
```

Scanf statement has a draw back it just terminates the statement as soon as it finds a blank space, suppose if we type the string new york then only the string new will be read and since there is a blank space after word "new" it will terminate the string.

Note that we can use the scanf without the ampersand symbol before the variable name.
In many applications it is required to process text by reading an entire line of text from the terminal.

The function getchar can be used repeatedly to read a sequence of successive single characters and store it in the array.

We cannot manipulate strings since C does not provide any operators for string. For instance we cannot assign one string to another directly.

For example:

```
String="xyz";
String1=string2;
```

Are not valid. To copy the chars in one string to another string we may do so on a character to character basis.

Writing strings to screen:

The printf statement along with format specifier %s to print strings on to the screen. The format %s can be used to display an array of characters that is terminated by the null character for example printf("%s",name); can be used to display the entire contents of the array name.

Arithmetic operations on characters:

We can also manipulate the characters as we manipulate numbers in c language. When ever the system encounters the character data it is automatically converted into a integer value by the system. We can represent a character as a interface by using the following method.

```
X='a';
Printf("%d\n",x);
```

Will display 97 on the screen. Arithmetic operations can also be performed on characters for example x='z'-1; is a valid statement. The ASCII value of 'z' is 122 the statement the therefore will assign 121 to variable x.

It is also possible to use character constants in relational expressions for example ch>'a' && ch < = 'z' will check whether the character stored in variable ch is a lower case letter. A character digit can also be converted into its equivalent integer value suppose un the expression a=character-'1'; where a is defined as an integer variable & character contains value 8 then a= ASCII value of 8 ASCII value '1'=56-49=7.

We can also get the support of the c library function to converts a string of digits into their equivalent integer values the general format of the function in x=atoi(string) here x is an integer variable & string is a character array containing string of digits.

String operations (string.h)

C language recognizes that string is a different class of array by letting us input and output the array as a unit and are terminated by null character. C library supports a large number of string handling functions that can be used to array out many o f the string manipulations such as:

- Length (number of characters in the string).
- Concatentation (adding two are more strings)
- Comparing two strings.
- Substring (Extract substring from a given string)
- Copy(copies one string over another)

To do all the operations described here it is essential to include string.h library header file in the program.

strlen() function:

This function counts and returns the number of characters in a string. The length does not include a null character.

Syntax n=strlen(string);

Where n is integer variable. Which receives the value of length of the string.

Example

length=strlen(“Hollywood”);

The function will assign number of characters 9 in the string to a integer variable length.

```
/*writr a c program to find the length of the string using strlen() function*/
#include < stdio.h >
include < string.h >
void main()
{
char name[100];
int length;
printf(“Enter the string”);
```

```
gets(name);
length=strlen(name);
printf("\nNumber of characters in the string is=%d",length);
}
```

strcat() function:

when you combine two strings, you add the characters of one string to the end of other string. This process is called concatenation. The strcat() function joins 2 strings together. It takes the following form

strcat(string1,string2)

string1 & string2 are character arrays. When the function strcat is executed string2 is appended to string1. the string at string2 remains unchanged.

Example

```
strcpy(string1,"sri");
strcpy(string2,"Bhagavan");
Printf("%s",strcat(string1,string2);
```

From the above program segment the value of string1 becomes sribhagavan. The string at str2 remains unchanged as bhagawan.

strcmp function:

In c you cannot directly compare the value of 2 strings in a condition like
if(string1==string2)
Most libraries however contain the strcmp() function, which returns a zero if 2 strings are equal, or a non zero number if the strings are not the same. The syntax of strcmp() is given below:

Strcmp(string1,string2)

String1 & string2 may be string variables or string constants. String1, & string2 may be string variables or string constants some computers return a negative if the string1 is

alphabetically less than the second and a positive number if the string is greater than the second.

Example:

strcmp("Newyork","Newyork") will return zero because 2 strings are equal.
strcmp("their","there") will return a 9 which is the numeric difference between ASCII 'i' and ASCII 'r'.
strcmp("The", "the") will return 32 which is the numeric difference between ASCII "T" & ASCII "t".

strcmpi() function

This function is same as strcmp() which compares 2 strings but not case sensitive.

Example

strcmpi("THE","the"); will return 0.

strcpy() function:

C does not allow you to assign the characters to a string directly as in the statement
name="Robert";
Instead use the strcpy(0 function found in most compilers the syntax of the function is illustrated below.

strcpy(string1,string2);

Strcpy function assigns the contents of string2 to string1. string2 may be a character array variable or a string constant.

strcpy(Name,"Robert");

In the above example Robert is assigned to the string called name.

strlwr () function:

This function converts all characters in a string from uppercase to lowercase.

syntax

strlwr(string);

For example:

strlwr("EXFORSYS") converts to Exforsys.

strrev() function:

This function reverses the characters in a string.

Syntax

strrev(string);

For ex strrev("program") reverses the characters in a string into "margrop".

strupr() function:

This function converts all characters in a string from lower case to uppercase.

Syntax

strupr(string);

For example strupr("exforsys") will convert the string to EXFORSYS.

```
/* Example program to use string functions*/
#include < stdio.h >
#include < string.h >
void main()
{
char s1[20],s2[20],s3[20];
int x,l1,l2,l3;
```

```
printf(“Enter the strings”);
scanf(“%s%s”,s1,s2);
x=strcmp(s1,s2);
if(x!=0)
{printf(“\nStrings are not equal\n”);
strcat(s1,s2);
}
else
printf(“\nStrings are equal”);
strcpy(s3,s1);
l1=strlen(s1);
l2=strlen(s2);
l3=strlen(s3);
printf(“\ns1=%s\t length=%d characters\n”,s1,l1);
printf(“\ns2=%s\t length=%d characters\n”,s2,l2);
printf(“\ns3=%s\t length=%d characters\n”,s3,l3);
}
```