Programing Methodology in C

Lecture 11 – All about Pointers in C

By Elubu Joseph

josebulinda@gmail.com

+256773086497

Agenda

Introduction to C Pointers

i. Declaration and initialization of pointers
ii. Accessing variable values using pointers
iii. Accessing array pointers
iv. Passing pointers as Arguments to functions
v. Returning pointers

What is appointer?

A Pointer in C language is a variable which holds the address of another variable of same data type.

Pointers are used to access memory and manipulate the address.

Pointers are one of the most distinct and exciting features of C language. It provides power and flexibility to the language. It may appear difficult to understand pointers at the beginning, but trust me, once you understand the concept, you will be able to do so much more with C language.

Address in C

Before we start understanding what pointers are and what they can do, let's start by understanding what "Address of a memory location" means?

Address is a memory location assigned to a given variable in which it's value will be stored. We can easily check this memory address, using the & symbol.

If **k** is the name of the variable, then **&k** will give it's address.

Sample program

Let's write a small program to see memory address of any variable that we define in our program

```
#include<stdio.h>
void main()
{
   int var = 7;
   printf("Value of the variable var is: %d\n", var);
   printf("Memory address of the variable var is: %x\n", &var);
}
```

Output

Value of the variable var is: 7
Memory address of the variable var is: **61fe1c** (this is dynamic depends on the actual address on you computer)

You must have also seen in the function scanf(), we mention &k to take user input for any variable k.

```
scanf("%d", &k);
```

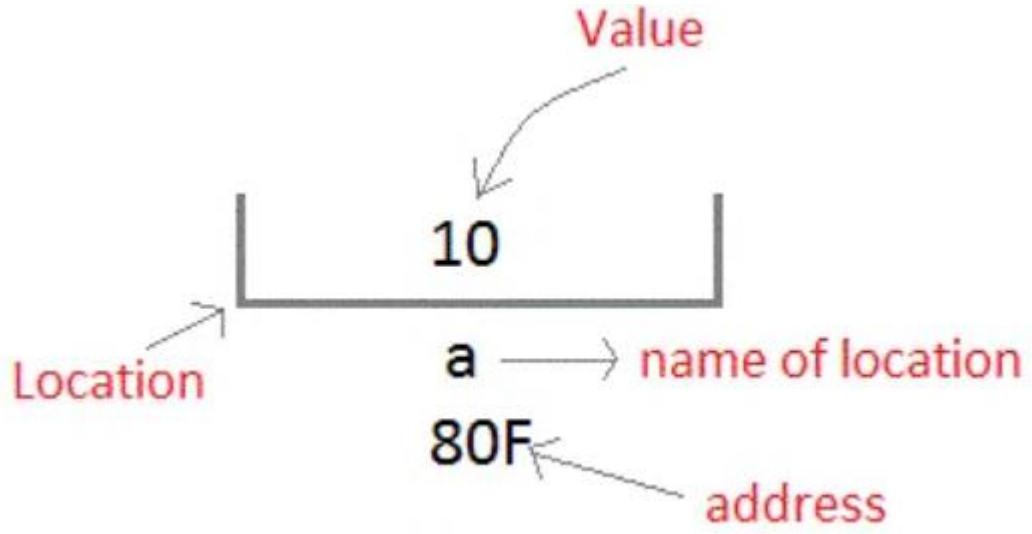
This is used to store the user inputted value to the address of the variable k.

Concept of Pointers

Whenever a **variable** is declared in a program, system allocates an address to that variable in the memory, to hold the assigned value. This location has its own address number, which we just saw above.

Let us assume that system has allocated memory location 80F for a variable a.

int a = 10;



Accessing Variable Value

We can access the value **10** either by using the variable name a or by using its address 80F.

The question is how we can access a variable using it's address?

Since the memory addresses are also just numbers, they can also be assigned to some other variable. The variables which are used to hold memory addresses are called **Pointer variables**.

A **pointer** variable is therefore nothing but a variable which holds an address of some other variable. And the value of a **pointer variable** gets stored in another memory location.

Benefits of using pointers

Below we have listed a few benefits of using pointers:

1. Pointers are more efficient in handling Arrays and Structures.
2. Pointers allow references to function and thereby helps in passing of function as arguments to other functions.
3. It reduces length of the program and its execution time as well.
4. It allows C language to support Dynamic Memory management.

Declaring, Initializing and using a pointer variable in C

We are now ready to learn how to declare, initialize and use a pointer and also about NULL pointer and its uses.

Declaration of C Pointer variable

The general syntax of pointer declaration is;

```
datatype *pointer_name;
```

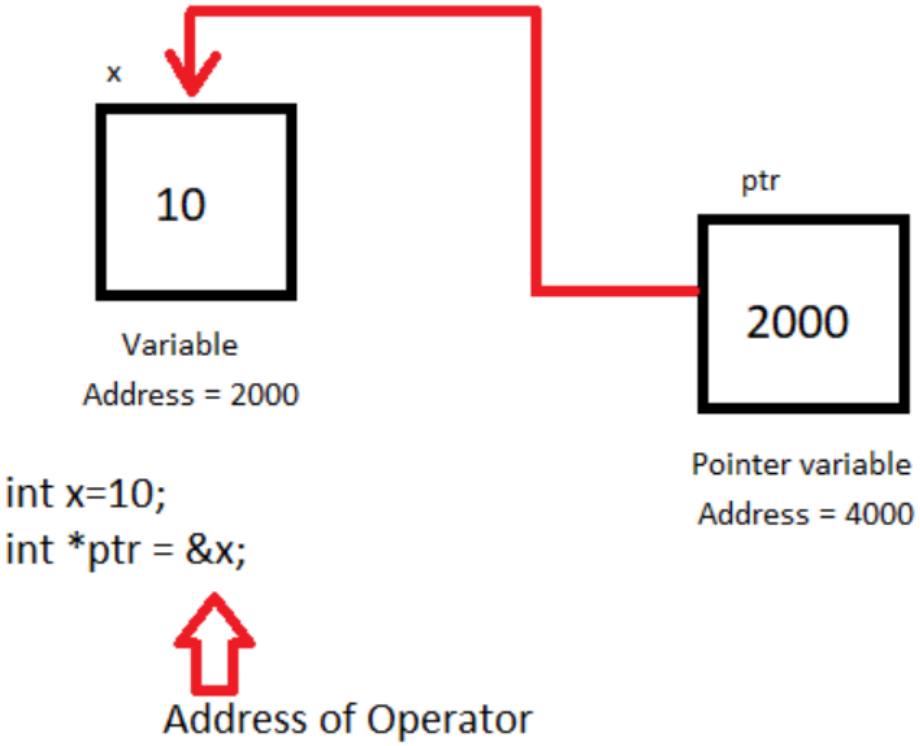
The data type of the pointer and the variable to which the pointer variable is pointing must be the same.

Initialization of C Pointer variable

Pointer Initialization is the process of assigning address of a variable to a **pointer** variable. It contains the address of a variable of the same data type.

In C language **address operator** & is used to determine the address of a variable. The & (immediately preceding a variable name) returns the address of the variable associated with it.

Initialization of C Pointer variable +




```
int a = 10;
int *ptr;      //pointer declaration
ptr = &a;      //pointer initialization
```

Initialization of C Pointer variable ++

Pointer variable always points to variables of the same datatype. For example:

```
float a;
int *ptr = &a;      // ERROR, type mismatch
```

Null pointers; While declaring a pointer variable, if it is not assigned to anything then it contains garbage value. Therefore, it is recommended to assign a NULL value to it,

```
int *ptr = NULL;
```

Using the pointer or Dereferencing of Pointer

Once a pointer has been assigned the address of a variable, to access the value of the variable, the pointer is **dereferenced**, using the **indirection operator** or **dereferencing operator** *. Consider the following example for better understanding.

Using the pointer or Dereferencing of Pointer+

```c
#include <stdio.h>
int main() {
int a; a = 10; int *p = &a; /* declaring and initializing the
pointer */
//prints the value of 'a'
     printf("%d\n", *p);
     printf("%d\n", *&a); //prints the address of 'a'
     printf("%u\n", &a); //using unsigned specifier
     printf("%u\n", p);
     printf("%u\n", &p); //prints address of 'p'
     return 0;
}
```

Output

```
10
10
6422044
6422044
6422032
```

Line three to Five may produce different values when ran on different computer since this prints out variable location address on a computer.

Source Code: UsingPointers.c

Points to remember while using pointers

1. While declaring/initializing the pointer variable, * indicates that the variable is a pointer.
2. The address of any variable is given by preceding the variable name with Ampersand &.
3. The pointer variable stores the address of another variable of same type.

Points to remember while using pointers +

4. The declaration int *a doesn't mean that a is going to contain an integer value. It means that a is going to contain the address of a variable storing integer value.
5. To access the value of a certain address stored by a pointer variable * is used. Here, the * can be read as **'value at'**.

Below is a simple program on pointer.

```
#include <stdio.h> int main() {
printf("\n\n\t\n Learning is made easy here.\n\n\n");
int var = 24; // actual variable declaration
int *p; p = &var; /* storing address of int variable var in
pointer p */
printf("\n\nAddress of var variable is: %x \n\n", &var);
printf("\n\nAddress stored in pointer variable p is: %x", p);
printf("\n\nValue of var variable or the value stored at
address p is %d ", *p);
printf("\n\n\t\t\tCoding is Fun !\n\n\n");
return 0;
}
```

See the output

```
 Learning is made easy here.

Address of var variable is: 61fe14



Address stored in pointer variable p is: 61fe14

Value of var variable or the value stored at address p is 24

                        Coding is Fun !
```

Sample Program to access Array of int Pointers

```
#include <stdio.h>
const int MAX = 5;
int main(){
	printf("\n\n\t\tTime to learn Programing\n\n\n");
	int var[]={10, 20, 30, 40, 50};
	int i = 0;
	int *ptr[MAX];
	for(i = 0; i < MAX; i++){ /* Assign the address of each of the array
element to the ptr array */
	ptr[i] = &var[i]; }
	for(i = 0; i < MAX; i++){
 /* ptr[i] stores the address of the element var[i]. Hence, *ptr[i] returns
the value of the element stored at location ptr[i] */
	printf("Value of var[%d] = %i\n\n", i, *ptr[i]);

}
	printf("\n\n\t\t\tCoding is Fun !\n\n\n");
return 0;
}
```

Explanation

```
printf("Value of names[%d] = %s\n\n",i,names[i]);
```

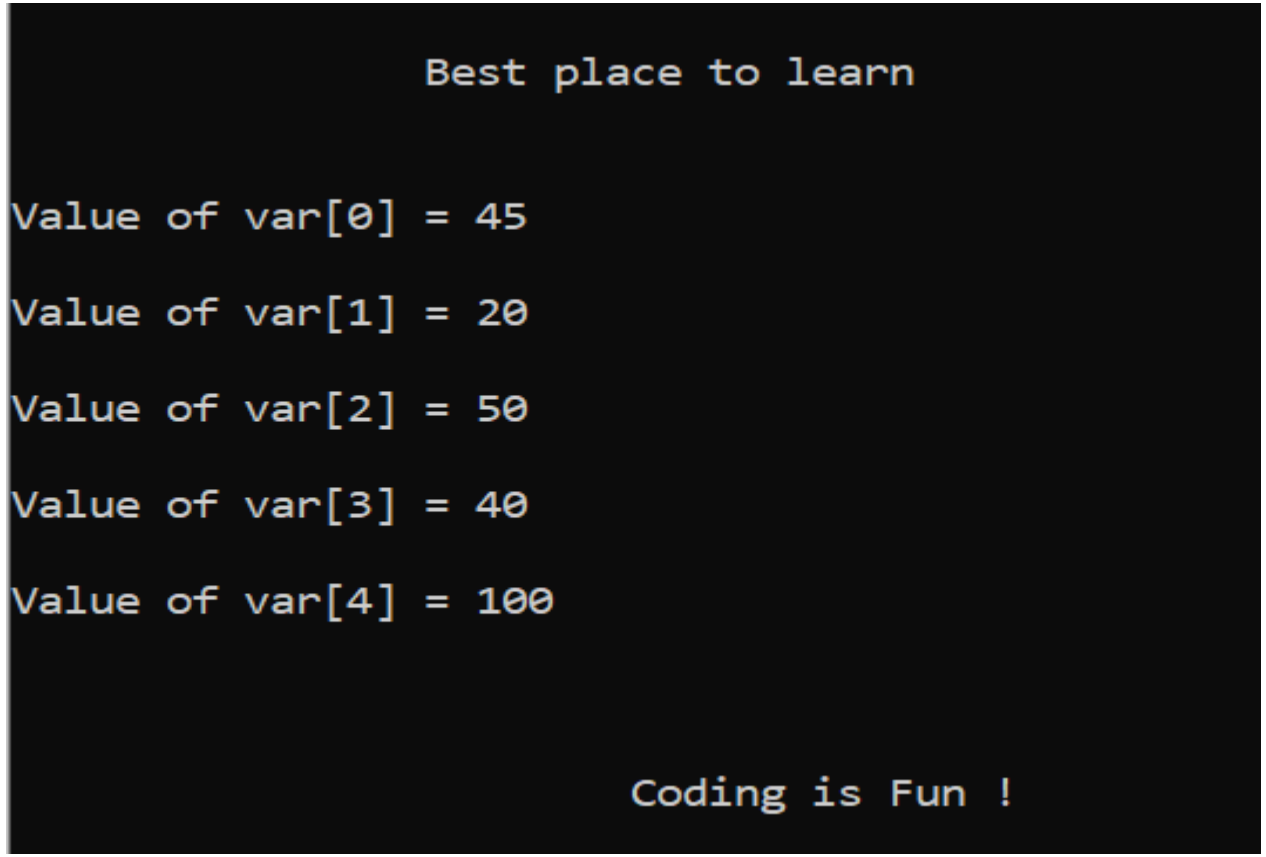
This statement is used for printing the complete name just using the pointer to the first character of each element of the names array.

Explanation

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

Here ptr[i] stores the address of the element var[i]. Hence, *ptr[i] returns the value of the element stored at location var[i].

Sample Program to access Array of int Pointers Output



Pointer to a Pointer in C(Double Pointer)

Pointers are used to store the address of other variables of similar datatype. But if you want to store the address of a pointer variable, then you again need a pointer to store it.

Thus, when one pointer variable stores the address of another pointer variable, it is known as **Pointer to Pointer** variable or **Double Pointer**.

Syntax:

```
int **p1;
```

Here, we have used two indirection operator(*) which stores and points to the address of a pointer variable i.e, int *.

If we want to store the address of this (double pointer) variable p1, then the syntax would become:

```
int ***p2
```

Simple program to represent Pointer to a Pointer

```
#include <stdio.h>
int main() {
      int a = 10;
      int *p1; int **p2;
      p1 = &a; p2 = &p1;
      printf("Address of a = %u\n", &a);
      printf("Address of p1 = %u\n", &p1);
      printf("Address of p2 = %u\n\n", &p2); // below print
statement will give the address of 'a'
printf("Value at the address stored by p2 = %u\n", *p2);
printf("Value at the address stored by p1 = %d\n\n", *p1);
printf("Value of **p2 = %d\n", **p2); //read this *(*p2) /*
This is not allowed, it will give a compile time error- p2 =
&a; printf("%u", p2); */ return 0;
}
```

Output

Address of a = **6422044**
Address of p1 = 6422032
Address of p2 = 6422024

Value at the address stored by p2 = **6422044**
Value at the address stored by p1 = 10

Value of **p2 = 10

Explanation

p1 pointer variable can only hold the address of the variable a (i.e Number of indirection operator(*)-1 variable). Similarly, p2 variable can only hold the address of variable p1. It cannot hold the address of variable a.

*p2 gives us the value at an address stored by the p2 pointer. p2 stores the address of p1 pointer and value at the address of p1 is the address of variable a. Thus, *p2 prints address of a.

Explanation

**p2 can be read as *(*p2). Hence, it gives us the value stored at the address *p2. From above statement, you know *p2 means the address of variable a. Hence, the value at the address *p2 is 10. Thus, **p2 prints 10.

Pointers as Function Argument in C

Pointer as a function parameter is used to hold addresses of arguments passed during function call. This is also known as **call by reference**.

When a function is called by reference any change made to the reference variable will effect the original variable.

Example Time: Swapping two numbers using Pointer

```
#include<stdio.h>
void swap(int *a, int *b);
int main() {
	int m = 10, n = 20;
	printf("m = %d\n", m); printf("n = %d\n\n", n);
	swap(&m, &n); //passing address of m and n to the swap
function
	printf("After Swapping:\n\n");
	printf("m = %d\n", m);
	printf("n = %d", n); return 0;
} /* pointer 'a' and 'b' holds and points to the address of 'm' and
'n' */
	void swap(int *a, int *b) {
	int temp; temp = *a; *a = *b; *b = temp;
}
```

Output

m = 10

n = 20

After Swapping:

m = 20

n = 10

Functions returning Pointer variables

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.

They have scope only inside the function. Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.

Sample Code

```
#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;
      else return y;
}
```

Output.
92 is larger

Safe ways to return a valid Pointer.

There are two ways you can do this; -

1.Either use **argument with functions**. Because argument passed to the functions are declared inside the calling function, hence they will live outside the function as well.

2.Or, use static **local variables** inside the function and return them. As static variables have a lifetime until the main() function exits, therefore they will be available througout the program.

Pointer to functions

It is possible to declare a pointer pointing to a function which can then be used as an argument in another function. A pointer to a function is declared as follows,

```
type (*pointer-name)(parameter);
```

Example :

```
int (*sum)(); //legal declaration of pointer to function
int *sum(); //This is not a declaration of pointer to
function
```

A function pointer can point to a specific function when it is assigned the name of that function.

```
int sum(int, int);
int (*s)(int, int); s = sum;
```

Here s is a pointer to a function sum. Now sum can be called using function pointer s along with providing the required argument values.

```
s (10, 20);
```

Example of Pointer to Function

```
#include <stdio.h>
int sum(int x, int y){
     return x+y;
}
int main( ){
     int (*fp)(int, int);
     fp = sum;
     int s = fp(10, 15);
     printf("Sum is %d", s);
return 0;
}
```

Output: 25

Complicated Function Pointer example

Below is an example of complex pointer function

```
void *(*foo) (int*);
```

Explanation

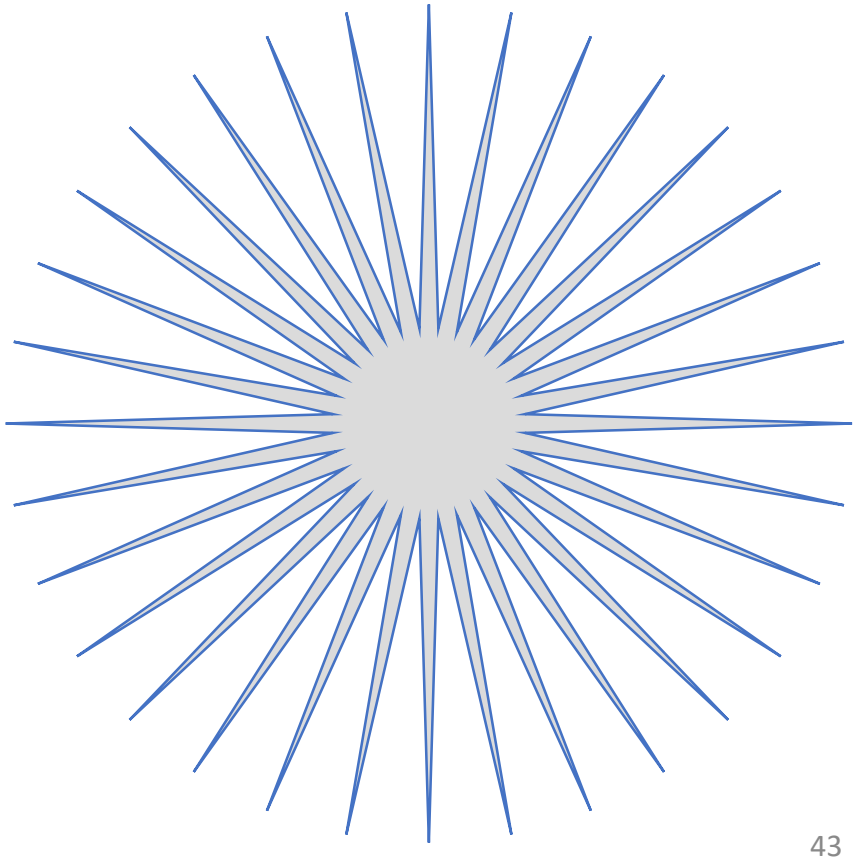
It appears complex but it is very simple. In this case (*foo) is a pointer to the function, whose argument is of int* type and return type is void*.

Summary

In summary therefore, we dealt with Pointers where we handled: -

i. Declaration and initialization of pointers
ii. Accessing variable values using pointers
iii. Accessing array pointers
iv. Passing pointers as Arguments to functions
v. Returning pointers
vi. Dealt with double pointers

Thank you for you attention



Reference

Pointer to a pointer in c(double pointer). Studytonight.com. (n.d.). Retrieved November 6, 2021, from https://www.studytonight.com/c/pointer-to-pointer.php.