Pointers are a very important feature in C. This feature differentiates C with other programming languages. To understand what a pointer is and how it works we need to understand memory allocation in C.

Memory Allocation in C

the computer memory or RAM as it is commonly called is basically a set of blocks or cells, each numbered from zero up to the maximum size of the memory. Let us see we have declared an int variable name num in which 10 is stored. Let's assume that C allocates the memory location 0x200 to it. Now C puts a label on this location which is the name of the variable, num. Lastly the value 10 is stored in this memory location . From now onwards whenever we refer to num, internally C would refer to the memory location 0x200.

the memory location of a variable can be referenced with the help of the '&' operator. It is known as the address-of operator. Its syntax is &variable-name;

It may happen that we have a memory location and we want to get the value or data stored there. To help in this process we have the dereference operator '*' called the indirection operator. It's syntax is *memory-address;

What is a Pointer?

Pointer is basically like a variable in which we can store the memory location of any variable. Just like variables, pointers also have a data- associated to them, i.e., an int pointer can hold only an int variable address.

We can declare appointed variable through the following syntax,

```
data-type *pointerVariableName;
```

It has three parts:

a) data-type: It is a valid C data-type.
b) *: This signifies that we are declaring a pointer and it is necessary whenever we are declaring a pointer.
c) pointerVariableName: It is any valid variable name.

There are two ways to initialize a pointer:

- With the help of '&'. For example,
  ```
  int num = 10;
  int *ptr = &num;
  ```
- With the help of another pointer. For example,
  ```
  Int *ptr1 = ptr;
  ```

Q. Write a program to swap two numbers using pointers.

https://youtu.be/P-2_iN9B5QI

Pointer Arithmetic

Just like any other variable pointers can also be added subtracted multiplied and divided. Also we can use the increment and the decrement operator on pointers. Unlike variables pointer arithmetic changes the position to which the pointer is pointing. Let's say for example a pointer, ptr, is pointing to the memory location 0x200, then on using ptr++ the pointer points to the memory location 0x201.

Pointers and Arrays

Pointers can be used to refer an array. In the following code the pointer refers to the zeroth index of the array arr.

```
int arr = {1,2,3,4,5};
int *ptr = arr;
```

This is a special property of pointers which helps in the interchangeability of arrays and pointers, i.e., both arr[0] and ptr[0] refers to the data at the zeroth index and the following two lines do the same thing, store 100 in the zeroth position.

```
*arr = 100;          *ptr = 100;
```

As the array end element or sequentially pointer arithmetic can help us to iterate through the array elements. The following are few examples of such operations,

```
int arr(5);
int * ptr = arr;      //ptr points at arr(0)
```

```
ptr[0] = 10;          // Assigns 10 to arr[0]
ptr++;                // ptr now points at arr[1]
ptr--;                // ptr now points back at arr[0]
*(ptr + 2) = 90;      // Assigns 90 to arr[2].
   // Note, ptr currently pointing at arr[0] not arr[1].
   // Hence (ptr + 4) will point at arr[4]
printf("%d", *ptr); //prints 10 as ptr points to arr[0]
*(arr + 0) = 107;      // Assigns 107 to arr[0]
```

NOTE :-

- while(ptr < &arr[5]) this condition will be true till ptr points within the range of the array.
- Please notice that in scanf("%d", ptr); statement, '&' is not used. The reason is that scanner accepts the memory location and pointer points to it. Hence we can directly use a pointer.

Dynamic Memory Allocation

dynamic memory allocation is a process by which whatever memory allocation needs to be done happens during the runtime. This is a very helpful tool to use with arrays when we don't know their sizes. Just like we had the string.h library which had certain string functions, there is a stdlib.h which has 4 functions regarding dynamic memory allocation. Most of the functions return a pointer which is of a void type. They are described below:

1. malloc() :-

The "malloc" or memory allocation method in C is used to dynamically allot a big single chunk of memory which has specified size. As malloc does not initialise the memory at execution time, the memory has garbage value in each block. If memory allocation has been unsuccessful then 'NULL' gets stored in the pointer, otherwise the address of the memory is stored in it. The syntax for this method is, `ptr = (cast-type*) malloc(byte-size);`

For example, `ptr = (int*) malloc(100 * sizeof(int));`

So what this does is that it creates a 4*100 = 400 bytes of memory space where only integer values can store. Sample program, which prints 'memory not allocated' if memory creation fails otherwise it ask the user to enter the digits into an array.

https://youtu.be/Os3kR8G-45U

2. calloc() :-

It is very similar to malloc() but has two differences,

- It initialises the memory with zero.
- it takes two parameters as input.

its syntax is, ptr = (cast-type*)calloc(n, element-size);

where, n is the no. of elements and element-size is the size of each element.

For example,

ptr = (float*) calloc(25, sizeof(float));

This creates 25 blocks of memory each with 0 value in it and can be used to store float values. Sample program, which prints 'memory not allocated' if memory creation fails otherwise it ask the user to enter the digits into an array.

https://youtu.be/sdrdYKrj0ik

3. free() :-

This method is used to dynamically deallocate the memory. The memory which we allocate using the above mentioned functions are not deallocated on their own hence every time we allocate the memory it is appreciated if we use the free method to deallocate it as well. This helps us to reduce wastage of memory by freeing it. The syntax is free(ptr);

A sample program which shows the use of free().

https://youtu.be/cM91uXgr0gY

4. realloc() :-

This function is used to dynamically change the memory allocation of a previously allocated memory. In other words if say we have allotted a memory with the help of malloc() or calloc() but we find out that it is not sufficient then we can reallocate the memory in them with the help of realloc(). Its syntax is ptr = realloc(ptr, newSize);
where ptr is reallocated with new size 'newSize'. If space is insufficient, allocation fails and returns a NULL pointer.

https://youtu.be/ZDbK3_vjaVo

SigmaPi Academy