Epitech Training

Session 5

C - Arrays

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

C - Arrays

Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –

```
type arrayName [ arraySize ];
```

This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and type can be any valid C data type. For example, to declare a 10-element array called balance of type double, use this statement –

```
double balance[10];
```

Here balance is a variable array which is sufficient to hold up to 10 double numbers.

C - Arrays

Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows −

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

C - Arrays

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

```
double salary = balance[9];
```

The above statement will take the 10th element from the array and assign the value to salary variable.

Don’t forget that the first index of an array is ALWAYS 0.

C - Arrays

Arrays in Detail

Arrays are important to C and should need a lot more attention. The following important concepts related to array should be clear to a C programmer −

- Multi-dimensional arrays
  - C supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
- Passing arrays to functions
  - You can pass to the function a pointer to an array by specifying the array's name without an index.
- Return array from a function
  - C allows a function to return an array.
- Pointer to an array
  - You can generate a pointer to the first element of an array by simply specifying the array name, without any index.

C - Pointers

Pointers in C are easy and fun to learn. Some C programming tasks are performed more easily with pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect C programmer. Let's start learning them in simple and easy steps.

As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory. Consider the following example, which prints the address of the variables defined –

C - Pointers

```c
#include <stdio.h>

int main () {

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1  );
   printf("Address of var2 variable: %x\n", &var2  );

   return 0;
}
```

C - Pointers

When the above code is compiled and executed, it produces the following result –

```
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

C - Pointers

What are Pointers?

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

```
type *var-name;
```

C - Pointers

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations −

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

C - Pointers

Pointers in Detail

Pointers have many but easy concepts and they are very important to C programming. The following important pointer concepts should be clear to any C programmer −

- Pointer arithmetic
  - There are four arithmetic operators that can be used in pointers: ++, --, +, -
- Array of pointers
  - You can define arrays to hold a number of pointers.
- Pointer to pointer
  - C allows you to have pointer on a pointer and so on.
- Passing pointers to functions in C
  - Passing an argument by reference or by address enable the passed argument to be changed in the calling function by the called function.
- Return pointer from functions in C
  - C allows a function to return a pointer to the local variable, static variable, and dynamically allocated memory as well.

C - Strings

Strings are actually one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization then you can write the above statement as follows −

```
char greeting[] = "Hello";
```

C - Strings

Useful functions

- strcpy(s1, s2);
  - Copies string s2 into string s1.
- strcat(s1, s2);
  - Concatenates string s2 onto the end of string s1.
- strlen(s1);
  - Returns the length of string s1.
- strcmp(s1, s2);
  - Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

C - Header Files

A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that comes with your compiler.

You request to use a header file in your program by including it with the C preprocessing directive #include, like you have seen inclusion of stdio.h header file, which comes along with your compiler.

C - Header Files

Including a header file is equal to copying the content of the header file but we do not do it because it will be error-prone and it is not a good idea to copy the content of a header file in the source files, especially if we have multiple source files in a program.

A simple practice in C or C++ programs is that we keep all the constants, macros, system wide global variables, and function prototypes in the header files and include that header file wherever it is required.

C - Header Files

Include Syntax

Both the user and the system header files are included using the preprocessing directive #include. It has the following two forms −

```
#include <file>
```

This form is used for system header files. It searches for a file named 'file' in a standard list of system directories. You can prepend directories to this list with the -I option while compiling your source code.

```
#include "file"
```

This form is used for header files of your own program. It searches for a file named 'file' in the directory containing the current file. You can prepend directories to this list with the -I option while compiling your source code.

C - Header Files

Include Operation

The #include directive works by directing the C preprocessor to scan the specified file as input before continuing with the rest of the current source file. The output from the preprocessor contains the output already generated, followed by the output resulting from the included file, followed by the output that comes from the text after the #includedirective.

C - Header Files

Once-Only Headers

If a header file happens to be included twice, the compiler will process its contents twice and it will result in an error. The standard way to prevent this is to enclose the entire real contents of the file in a conditional, like this –

```
#ifndef HEADER_FILE
#define HEADER_FILE

the entire header file file

#endif
```

This construct is commonly known as a wrapper #ifndef. When the header is included again, the conditional will be false, because HEADER_FILE is defined. The preprocessor will skip over the entire contents of the file, and the compiler will not see it twice.