Epitech Training

Session 4

C programming language

C is a general-purpose, imperative computer programming language, supporting structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations. By design, C provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, including operating systems, as well as various application software for computers ranging from supercomputers to embedded systems.

C programming language

C is an imperative procedural language. It was designed to be compiled using a relatively straightforward compiler, to provide low-level access to memory, to provide language constructs that map efficiently to machine instructions, and to require minimal run-time support. Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with few changes to its source code. The language has become available on a very wide range of platforms, from embedded microcontrollers to supercomputers.

C programming language

Many later languages have borrowed directly or indirectly from C, including C++, D, Go, Rust, Java, JavaScript, Limbo, LPC, C#, Objective-C, Perl, PHP, Python, Swift, Verilog (hardware description language), and Unix's C shell. These languages have drawn many of their control structures and other basic features from C. Most of them (with Python being the most dramatic exception) are also very syntactically similar to C in general, and they tend to combine the recognizable expression and statement syntax of C with underlying type systems, data models, and semantics that can be radically different.

C - Data Types

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. The types in C can be classified as follows

- Basic Types
  - They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
- Enumerated types
  - They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
- The type void
  - The type specifier void indicates that no value is available.
- Derived types
  - They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

C - Data Types

The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, whereas other types will be covered in the upcoming chapters

C - Data Types

Integer Types

- char 1 byte -128 to 127 or 0 to 255
- unsigned char 1 byte 0 to 255
- signed char 1 byte -128 to 127
- int 2 or 4 bytes -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
- unsigned int 2 or 4 bytes 0 to 65,535 or 0 to 4,294,967,295
- short 2 bytes -32,768 to 32,767
- unsigned short 2 bytes 0 to 65,535
- long 4 bytes -2,147,483,648 to 2,147,483,647
- unsigned long 4 bytes 0 to 4,294,967,295

C - Data Types

To get the exact size of a type or a variable on a particular platform, you can use the sizeof operator. The expressions sizeof(type) yields the storage size of the object or type in bytes.

C - Data Types

Floating-Point Types

- float 4 byte 1.2E-38 to 3.4E+38 6 decimal places
- double 8 byte 2.3E-308 to 1.7E+308 15 decimal places
- long double 10 byte 3.4E-4932 to 1.1E+4932 19 decimal places

C - Data Types

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. The following example prints the storage space taken by a float type and its range values

```c
#include <stdio.h>
#include <float.h>

int main() {

   printf("Storage size for float : %d \n", sizeof(float));
   printf("Minimum float positive value: %E\n", FLT_MIN );
   printf("Maximum float positive value: %E\n", FLT_MAX );
   printf("Precision value: %d\n", FLT_DIG );

   return 0;
}
```

C - Data Types

```
Storage size for float : 4
Minimum float positive value: 1.175494E-38
Maximum float positive value: 3.402823E+38
Precision value: 6
```

C - Data Types

The void Type

The void type specifies that no value is available. It is used in three kinds of situations

- Function returns as void
  - There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, void exit (int status);
- Function arguments as void
  - There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, int rand(void);

C - Data Types

The void Type

The void type specifies that no value is available. It is used in three kinds of situations

- Function returns as void
  - There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, void exit (int status);
- Function arguments as void
  - There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, int rand(void);
- Pointers to void
  - A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function void *malloc(size_t size); returns a pointer to void which can be casted to any data type.

C - Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive.

C - Variables

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

C - Variables

Variable Definition in C

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, type must be a valid C data type including char, w_char, int, float, double, bool, or any user-defined object; and variable_list may consist of one or more identifier names separated by commas.

C - Variables

A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation without requiring the complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use the keyword extern to declare a variable at any place. Though you can declare a variable multiple times in your C program, it can be defined only once in a file, a function, or a block of code.

C - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

C - Operators

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then

- + Adds two operands.
  - A + B = 30
- − Subtracts second operand from the first.
  - A − B = -10
- * Multiplies both operands.
  - A * B = 200
- / Divides numerator by de-numerator.
  - B / A = 2
- % Modulus Operator and remainder of after an integer division.
  - B % A = 0
- ++ Increment operator increases the integer value by one.
  - A++ = 11
- -- Decrement operator decreases the integer value by one.
  - A-- = 9

C - Operators

Relational Operators

The following table shows all the relational operators supported by C. Assume variable Aholds 10 and variable B holds 20 then

- == Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
- != Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
- > Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.
- < Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.
- >= Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.
- <= Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.

C - Operators

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then −

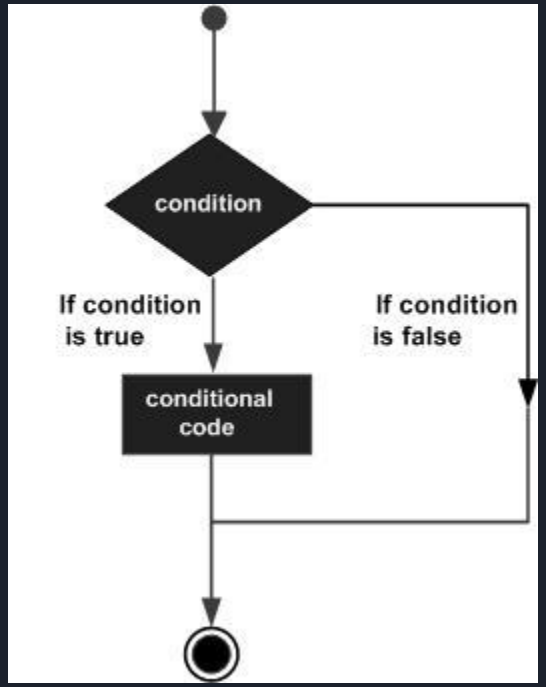
- && Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
- || Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.
- ! Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

C - Decision Making

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

C programming language assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value.

C - Decision Making



C - Decision Making

- if statement
  - An if statement consists of a boolean expression followed by one or more statements.
- if...else statement
  - An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.
- nested if statements
  - You can use one if or else if statement inside another if or else if statement(s).
- switch statement
  - A switch statement allows a variable to be tested for equality against a list of values.
- nested switch statements
  - You can use one switch statement inside another switch statement(s).

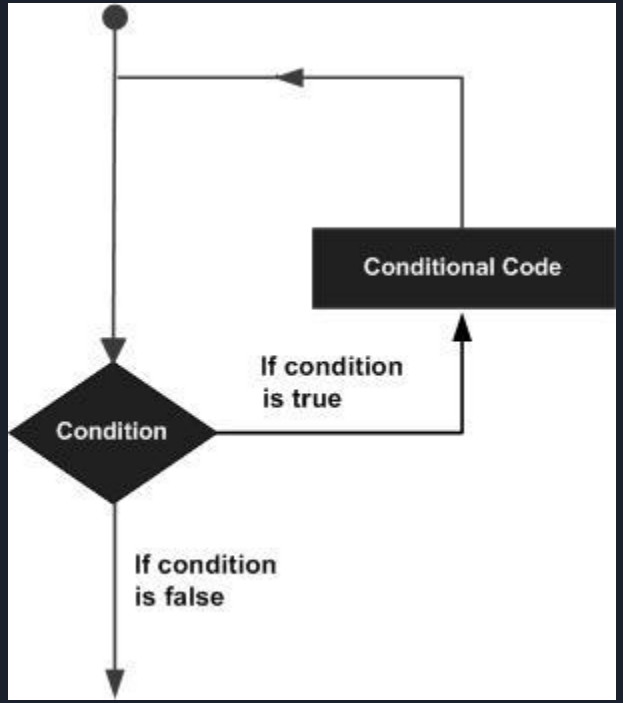
C - Loops

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times.

C - Loops


Conditional Code
If condition
is true
Condition
If condition
is false

C - Loops

- while loop
  - Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
- for loop
  - Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- do...while loop
  - It is more like a while statement, except that it tests the condition at the end of the loop body.
- nested loops
  - You can use one or more loops inside any other while, for, or do..while loop.

C - Loops

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

- break statement
  - Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
- continue statement
  - Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

C - Loops

The Infinite Loop

A loop becomes an infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the for(;;) construct to signify an infinite loop.

C - Loops

```c
#include <stdio.h>

int main () {

   for( ; ; ) {
      printf("This loop will run forever.\n");
   }

   return 0;
}
```

C - Functions

A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

C - Functions

The C standard library provides numerous built-in functions that your program can call. For example, strcat() to concatenate two strings, memcpy() to copy one memory location to another location, and many more functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

C - Functions

Defining a Function

The general form of a function definition in C programming language is as follows –

```
return_type function_name( parameter list ) {
   body of the function
}
```

C - Functions

A function definition in C programming consists of a function header and a function body. Here are all the parts of a function −

- **Return Type** − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
- **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- Function Body − The function body contains a collection of statements that define what the function does.

C - Functions

```c
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

C - Functions

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

```
return_type function_name( parameter list );
```

Parameter names are not important in function declaration only their type is required,

```
int max(int, int);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

C - Functions

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

C - Functions

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

C - Functions

While calling a function, there are two ways in which arguments can be passed to a function

- Call by value
  - This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
- Call by reference
  - This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

Exercices

1. my_print_alpha
   a. Write a function that, beginning with a, displays the lowercase alphabet in ascending order, on a single line. It must
   b. be prototyped as follows: int my_print_alpha ( ) ;
2. my_print_revalpha
   a. Write a function that, beginning with z, displays the lowercase alphabet in descending order, on a single line. It must
   b. be prototyped as follows: int my_print_revalpha ( ) ;
3. my_print_digits
   a. Write a function that displays all the digits, on a single line, in ascending order.
   b. It must be prototyped as follows: int my_print_digits ( ) ;
4. my_isneg
   a. Write a function that displays either N if the integer passed as parameter is negative, P, if positive or null. It must be
   b. prototyped as follows: int my_isneg (int n) ;