CS 240 Programming in C

Constants and Variables

Jan 26, 2023

Basic C program structure

Structure of C Program

	1	`#include <stdio.h>`	Header
	2	`int main(void)`	Main
BODY	3	`{`	
	4	`printf("Hello World");`	Statement
	5	`return 0;`	Return
	6	`}`	

1

[1] Source: Geeksforgeeks

HelloWorld.c

Source Code:

```
#include <stdio.h>
int main()
{
    /* hello world program in C */
    printf("Hello World!\n");
    return 0;
}
```

Commands:
Compile: gcc HelloWorld.c -o hello
Execute: ./hello

Hello World Example

A C program basically consists of the following parts

- Preprocessor Commands
- Functions
- Variables
- Statements Expressions
- Comments

Hello World Example

Let us take a look at the various parts of the Hello world program -

- The first line of the program include <stdio.h> is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line int main() is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
- The next line printf(...) is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line return 0; terminates the main() function and returns the value 0.

Hello World Example

- A C program consists of one or more functions, with the main() function being the entry point of the program.
- The main function contains the instructions that are executed when the program runs.
- The main function is defined with the keyword "int" before the function name, indicating that it returns an integer value.
- The next line printf(...) is another function available in C which causes the message "Hello World!" to be displayed on the screen.
- The main function typically includes a return statement at the end, which returns a value to indicate the status of the program when it exits.

Compile and execute C program

- Open VS Code editor and add the above-mentioned code.
- Save the file as hello.c
- To run the program, you will need a C compiler, such as GCC.
- Once you have a compiler installed, you can use the command line to compile the program by typing "gcc hello.c -o hello"
- This will create an executable file called "hello"
- You can then run the program by typing "./hello"
- The program will print "Hello World!" to the console, and the program will exit with a status of 0 indicating success.

Comments

Comments are like helping text in your C program and they are ignored by the compiler. They start with /* and terminate with the characters */ as shown below

```
/* my first program in C */
```

You cannot have comments within comments and they do not occur within a string or character literals.

Tokens

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens

```
printf("Hello World! \n");
```

The individual tokens are

```
printf
(
    "Hello World! \n"
)
;
```

Basic Data Types

- Data types in C are used to define the type of a variable or a value.
- Different data types have different sizes and ranges, and they are used to store different types of data.

Char data type

- The char data type is used to store character values (single letters or symbols).
- The size of a char variable is typically 1 byte, and the range of values that can be stored is 0 to 255.

Integer data type

- The int data type is used to store integer values (whole numbers).
- The size of an int variable is typically 4 bytes, and the range of values that can be stored is -2147483648 to 2147483647.

Integer data type

Type	Storage size	Value range
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	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

2

[2] Source: tutorialspoint

Sizeof() and Limits.h

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expression sizeof(type) yields the storage size of the object or type in bytes. Given in the next slide is an example to get the size of various types on a machine using different constants defined in limits.h header file

Note: The range and size of data types can vary depending on the platform and the compiler used. The following mentioned ranges and sizes are for 32-bit systems.

Data type range and limits

```
#include <stdio.h>
#include <limits.h>
int main() {
    printf("CHAR_BIT    :   %d\n", CHAR_BIT);
    printf("CHAR_MAX    :   %d\n", CHAR_MAX);
    printf("CHAR_MIN    :   %d\n", CHAR_MIN);
    printf("INT_MAX     :   %d\n", INT_MAX);
    printf("INT_MIN     :   %d\n", INT_MIN);
    printf("SCHAR_MAX   :   %d\n", SCHAR_MAX);
    printf("SCHAR_MIN   :   %d\n", SCHAR_MIN);
    printf("SHRT_MAX    :   %d\n", SHRT_MAX);
    printf("SHRT_MIN    :   %d\n", SHRT_MIN);
    printf("LONG_MAX    :   %ld\n", (long) LONG_MAX);
    printf("LONG_MIN    :   %ld\n", (long) LONG_MIN);
    return 0;
}
```

Float data type

- The float data type is used to store floating-point numbers (decimal numbers).
- The size of a float variable is typically 4 bytes, and the range of values that can be stored is approximately 3.4E-38 to 3.4E+38.

Data type range and limits

```
#include <stdio.h>
#include <float.h>
int main() {
    printf("Storage size for float : %lu \n", sizeof(float));
    printf("FLT_MAX     :   %g\n", (float) FLT_MAX);
    printf("FLT_MIN     :   %g\n", (float) FLT_MIN);
    printf("-FLT_MAX    :   %g\n", (float) -FLT_MAX);
    printf("-FLT_MIN    :   %g\n", (float) -FLT_MIN);
    printf("DBL_MAX     :   %g\n", (double) DBL_MAX);
    printf("DBL_MIN     :   %g\n", (double) DBL_MIN);
    printf("-DBL_MAX      :  %g\n", (double) -DBL_MAX);
    printf("Precision value: %d\n", FLT_DIG);
    return 0;
}
```

Double, Short, Long data types

- The double data type is used to store double-precision floating-point numbers and it has 8 bytes of memory and a range of approximately 1.7E-308 to 1.7E+308
- The short data type is used to store short integer values and it has 2 bytes of memory and a range of -32768 to 32767
- The long data type is used to store long integer values and it has 4 bytes of memory and a range of -2147483648 to 2147483647

Variables

In C, variables are used to store data in a program and allow it to change its behavior based on the data it receives. Each variable has a specific type, which determines its memory layout, range of values, and set of operations that can be applied to it. Variables can be declared and defined in C, with a declaration providing assurance to the compiler that the variable exists and a definition providing the actual details about the variable. Variables can be initialized during declaration and can have static or extern storage duration.

Variable declaration

- In C, variables must be declared before they can be used.
- A variable is declared by specifying its type, followed by its name.
- For example- int x; declares a variable called ‘x‘ of type int.
- Variables can also be initialized with a value at the time of declaration, like int x = 10;

Lvalues and Rvalues

There are two kinds of expressions in C

- **lvalue** Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue** The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.

Lvalues and Rvalues

Variables are lvalues and so they may appear on the left-hand side of an assignment. Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side. Take a look at the following valid and invalid statements

```
int g = 20; // valid statement

10 = 20;    // invalid statement;
// would generate a compile-time error
```

Example variable declaration

Variable Declaration

```
int x;
float y;
char c;
```

With initialization

```
int x = 10;
float y = 3.14;
char c = 'a';
```

Naming conventions for variables

- Variable names can contain letters, digits, and underscores.
- Variable names must start with a letter or an underscore.
- Variable names are case-sensitive.
- It is recommended to use meaningful and descriptive variable names, like "age" instead of "a"
- Variables names should not be a C keyword or a library function name.

Constants

- In C, constants are values that are fixed and cannot be changed during the execution of a program.
- Constants can be defined using the keyword "const" or "define" preprocessor directive.
- Constants defined using "const" are called "compile-time constants" and those defined using "define" are called "preprocessor constants".
- "const" variables have to be initialized at the time of declaration and cannot be modified later on in the program.
- "define" constants do not have to be initialized at the time of declaration and can be used as simple text replacement.

Constants with "const"

- Syntax for defining a constant with "const" keyword:
  const data_type constant_name = value;

```
const int MAX_SIZE = 100;
```

- The constant MAX_SIZE cannot be modified once it is defined and can be used throughout the program.
- Using "const" is more type-safe and generates an error if the constant is accidentally modified.

Constants with "#define"

- Syntax for defining a constant with "#define" preprocessor directive: define constant_name value

```
#define PI 3.14
```

- The constant PI can be used throughout the program as a simple text replacement.
- "#define" constants are replaced by the preprocessor before the program is compiled.
- Using "#define" is less type-safe and does not generate an error if the constant is accidentally modified.

Comparison between "const" and "#define"

- "const" variables are actual variables with a memory location and can be used with pointers and arrays. "#define" constants are not actual variables and cannot be used with pointers and arrays.
- "const" variables have a specific data type and generate an error if the type is not matched. "define" constants do not have a specific data type and are replaced by the preprocessor as simple text.

Input using scanf()

- The scanf() function is used to read input from the user.
- The syntax of the scanf() function is as follows: scanf("format string", variable1, variable2, ...);
- The format string specifies the type of input that the function should expect, such as %d for integers or %f for floating point numbers.
- The variables that come after the format string are the addresses of the variables where the input will be stored.
- Example:

```
int age; scanf("\%d", &age);
```

Input using scanf() (cont.)

- The scanf() function returns the number of input items successfully matched and assigned, which can be used to check for errors. Example:

```
int a, b, c;
int count = scanf("%d %d %d", &a, &b, &c);
if(count != 3) printf("Error: invalid input");
```

Output using printf()

- The printf() function is used to display output to the user.
- The syntax of the printf() function is as follows:
  printf("format string", variable1, variable2, ...);
- The format string specifies how the variables should be displayed, such as %d for integers or %f for floating point numbers.
- Example:

```
int age = 30;
printf("Age: %d", age);
```

Output using printf() (cont.)

- The printf() function can also be used to display strings using the %s format specifier.
  Example:

```
char name[20] = "John Smith";
printf("Name: %s", name);
```

- The printf() function also supports various formatting options, such as specifying the width or precision of a value, using flags like %-10d to left-align a value within a field of width 10.

Output using printf() (cont.)

- The printf() function also support to print special characters like \n (new line) and \t (tab)
  Example:

```
printf("Hello\n World\t!");

/*
Output: Hello
World !
*/
```

Output using printf() (cont.)

- The printf() function can also be used to print formatted output using the %x specifier for hexadecimal, %o for octal, and %b for binary. Example:

```
int a = 15;
printf("a in hex: %x, a in octal: %o", a, a);
```

Format specifiers

Format Specifier	Description
%c	a single character
%s	a string
%hi	short (signed)
%hu	short (unsigned)
%Lf	long double
%n	prints nothing
%d	a decimal integer (assumes base 10)
%i	a decimal integer (detects the base automatically)
%o	an octal (base 8) integer
%x	a hexadecimal (base 16) integer
%p	an address (or pointer)
%f	a floating point number for floats
%u	int unsigned decimal
%e	a floating point number in scientific notation
%E	a floating point number in scientific notation

Resource Links

- https://www.tutorialspoint.com/cprogramming/c_basic_syntax.htm
- https://www.tutorialspoint.com/cprogramming/c_data_types.htm
- https://www.tutorialspoint.com/cprogramming/c_variables.htm