Basics of C

Akash Hegde

Seventh Sense Talent Solutions

Vivekananda Institute of Technology

25 March 2021

Introduction to C

- **C** is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system.
- C is the most widely used computer language.
- It keeps fluctuating at number one scale of popularity along with Java programming language, which is also equally popular and most widely used among modern software programmers.

Introduction to C

- **C programming** language is necessary for students and working professionals to become proficient Software Engineers when they are working in Software Development Domain.
- Some of the key advantages of learning C Programming:
  - Easy to learn
  - Structured language
  - It produces efficient programs
  - It can handle low-level activities
  - It can be compiled on a variety of computer platforms

Introduction to C

- Some examples of the use of C are -
  - Operating Systems
  - Language Compilers
  - Assemblers
  - Text Editors
  - Print Spoolers
  - Network Drivers
  - Modern Programs
  - Databases
  - Language Interpreters
  - Utilities

Environment Setup for C

- Text Editor -
  - Used to type your C program.
  - Examples of few editors - Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.
  - The files you create with your editor are called the source files and they contain the program source codes.
  - The source files for C programs are typically named with the extension "**.c**".

Environment Setup for C

- C Compiler -
  - The source code written in source file is the human readable source for your program.
  - It needs to be "compiled", into machine language so that your CPU can actually execute the program as per the instructions given.
  - The compiler compiles the source codes into final executable programs.
  - The most frequently used and free available compiler is the GNU C/C++ compiler.

Program Structure

- A C program basically consists of the following parts –
  - Preprocessor Commands
  - Functions
  - Variables
  - Statements & Expressions
  - Comments
- Hello World example:

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

Program Structure

- 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.

Compile and Execute a C Program

- Open a text editor and add the Hello World example code.
- Save the file as *hello.c*
- Open a command prompt and go to the directory where you have saved the file.
- Type *gcc hello.c* and press enter to compile your code.
- If there are no errors in your code, the command prompt will take you to the next line and would generate *a.out* executable file.
- Now, type *a.out* or *./a.out* to execute your program.
- You will see the output *"Hello, World!"* printed on the screen.

Basic Syntax

- **Tokens in C:**
- 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 has five tokens –

  ```
  printf("Hello, World! \n");
  ```
- The individual tokens are –

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

Basic Syntax

- **Semicolons:**
- In a C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.
- Given below are two different statements –

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

-
  ```
  return 0;
  ```

- **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.

Basic Syntax

- **Identifiers:**
- A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).
- C does not allow punctuation characters such as @, $, and % within identifiers.
- C is a **case-sensitive** programming language.
  Thus, *Manpower* and *manpower* are two different identifiers in C.
- Here are some examples of acceptable identifiers –
  ```
  mohd       zara    abc    move_name   a_123
  ```

-
  ```
  myname50   _temp   j      a23b9       retVal
  ```

Basic Syntax

- **Keywords:**
- These reserved words may not be used as constants or variables or any other identifier names.

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double			

Basic Syntax

- **Whitespace in C:**
- A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.
- Whitespace is the term used in C to describe blanks, tabs, newline characters and comments.
- Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins.
- Therefore, in the following statement –

  ```
  int age;
  ```
- There must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them.

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.

Data Types – Integer Types

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

Data Types – Floating-point Types

Type	Storage size	Value range	Precision
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

Data Types – Void Types

Type	Description
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.

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.

Variables

Type	Description
char	Typically a single octet(one byte). It is a character type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.

- 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.

Constants

- Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.
- Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are enumeration constants as well.
- Constants are treated just like regular variables except that their values cannot be modified after their definition.
- An **integer literal** can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.
- A **floating-point literal** has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

Constants

- **Character literals** are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of **char** type.
  A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').
- **String literals** are enclosed in double quotes "".
  A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.
  You can break a long line into multiple lines using string literals and separating them using white spaces.
- **Defining constants -**
  - Using #define preprocessor –
    `#define identifier value`
  - Using const keyword –
    `const type variable = value;`

Storage Classes

- A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program.
- They precede the type that they modify.
- We have four different storage classes in a C program –
  - **auto** - default storage class for all local variables.
  - **register** - used to define local variables that should be stored in a register instead of RAM.
  - **static** - instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope.
  - **extern** - used to give a reference of a global variable that is visible to all the program files.

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
  - Miscellaneous Operators

Operators - Arithmetic

Operator	Description	Example
+	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

Operators - Relational

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is 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.	(A >= B) is not 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.	(A <= B) is true.

Operators - Logical

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is 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.	!(A && B) is true.

Operators - Bitwise

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Operators - Assignment

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A

Operators - Assignment

Operator	Description	Example
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Operators - Miscellaneous

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression (Ternary operator)	If Condition is true ? then value X : otherwise value Y

Decision Making Statements

- One or more conditions to be evaluated 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.

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

Thank you!