



C Programming I

Alexander B. Pacheco
LTS Research Computing
May 20, 2015

Outline

- 1 Introduction
- 2 Program Structure
- 3 Basic Syntax
- 4 Data Types, Variables and Constants
- 5 Control Structures: for, if & switch

Introduction

What is the C Language?

- 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.
- The UNIX operating system, the C compiler, and essentially all UNIX applications programs have been written in C.
- C is the most widely used computer language.
 - Easy to learn
 - Structured language
 - Produces efficient programs
 - Handles low-level activities
 - Can be compiled on a variety of computer platforms
- Most of the state-of-the-art softwares have been implemented using C.
- Today's most popular Linux OS and RDBMS MySQL have been written in C.

What do you need to learn C?

1 C Compiler

- What is a Compiler?
 - A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code).
- How does a compiler do?
 - Translate C source code into a binary executable
- List of Common Compilers:
 - GCC GNU Project (Free, available on most *NIX systems)
 - Intel Compiler
 - Portland Group (PGI) Compiler
 - Microsoft Visual Studio
 - IBM XL Compiler

2 Text Editor

- Emacs
- VI/VIM
- Notepad++ (avoid Notepad if you will eventually use a *NIX system)
- Integrated Development Environment: Eclipse, XCode, Visual Studio, etc

Program Structure

Program Structure

A C Program consists of the following parts

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

A Simple Hello World Code

```
#include <stdio.h>

int main ()
{
    /* My First C Code */
    printf("Hello World!\n");
    return 0;
}
```

Compile and execute the code

```
dyn100077:Exercise apacheco$ gcc hello.c
dyn100077:Exercise apacheco$ ./a.out
Hello World!
```

My First C Code

```
#include <stdio.h>

int main ()
{
    /* My First C Code */
    printf("Hello World!\n");
    return 0;
}
```

- `#include <stdio.h>` is a preprocessor command.

It tells a C compiler to include `stdio.h` file before going to actual compilation.

- `int main()` is the main function where program execution begins.
- `/* ... */` is a comment and ignored by the compiler.
- `printf(...)` is function that prints `Hello World!` to the screen.
- `return 0;` terminates `main()` function and returns the value 0.

Basic Syntax

Basic C Syntax I

- C is a case sensitive programming language i.e. program is not the same as Program or PROGRAM.
- Each individual statement must end with a semicolon.
- Whitespace i.e. tabs or spaces is insignificant except whitespace within a character string.
- All C statements are free format i.e. no specified layout or column assignment as in FORTRAN77.

```
#include <stdio.h>
int main () { /* My First C Code */ printf("Hello World!\n"); return 0; }
```

will produce the exact same result as the code on the previous slide.

- In C everything within `/* and */` is a comment. Comments can span multiple lines.

```
/* this is single line comment */
/* This
is a
multiline comment */
```

Basic C Syntax II

- Always use proper comments in your code. Your code will most likely be handed to someone long after you are gone.
- Comments are completely ignored by compiler (test/debug code)

Data Types, Variables and Constants

Data Types

Basic Types: There are five basic data types

- 1 int - integer: a whole number.
- 2 float - floating point value: ie a number with a fractional part.
- 3 double - a double-precision floating point value.
- 4 char - a single character.
- 5 void - valueless special purpose type.

Derived Types: These include

- 1 Pointers
- 2 Arrays
- 3 Structures
- 4 Union
- 5 Function

- The array and structure types are referred to collectively as the aggregate types.
- The type of a function specifies the type of the function's return value.

Basic Data Types: Integer

| Type | Storage size (in bytes) | Value range |
|----------------|-------------------------|---------------------------------------|
| char | 1 | -128 to 127 or 0 to 255 |
| unsigned char | 1 | 0 to 255 |
| signed char | 1 | -128 to 127 |
| int | 2 | -32,768 to 32,767 |
| | or 4 | or -2,147,483,648 to 2,147,483,647 |
| unsigned int | 2 | 0 to 65,535 |
| | or 4 | or 0 to 4,294,967,295 |
| short | 2 | -32,768 to 32,767 |
| unsigned short | 2 | 0 to 65,535 |
| long | 4 | -2,147,483,648 to 2,147,483,647 |
| unsigned long | 4 | 0 to 4,294,967,295 |

- 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.

Basic Data Types: Floating-Point & void

| Type | Storage size | Value range | Precision (decimal places) |
|-------------|--------------|-----------------------|----------------------------|
| float | 4 bytes | 1.2E-38 to 3.4E38 | 6 |
| double | 8 bytes | 2.3E-308 to 1.7E308 | 15 |
| long double | 10 bytes | 3.4E-4932 to 1.1E4932 | 19 |

| Situation | Description |
|----------------------------|-----------------------------------|
| function returns as void | function with no return value |
| function arguments as void | function with no parameter |
| pointers to void | address of an object without type |

Input and Output

- C or any programming language in general needs to be interactive i.e. write something back and optionally read data to be useful.
- Similar to Unix, C treats all devices as files.

| Standard File | File Pointer | Device |
|-----------------|--------------|----------|
| Standard Input | stdin | Keyboard |
| Standard Output | stdout | Screen |
| Standard Error | stderr | Screen |

- C Programming language provides three functions to read/write from standard input/output

| | Unformatted | | Formatted |
|--------|-------------|------|-----------|
| Input | getchar | gets | scanf |
| Output | putchar | puts | printf |

The `getchar()` & `putchar()` functions

- The `int getchar(void)` function reads the next available character from the screen and returns it as an integer.

This function reads only single character at a time.

- The `int putchar(int c)` function puts the passed character on the screen and returns the same character.

This function puts only single character at a time.

The `gets()` & `puts()` functions

- The `char *gets(char *s)` function reads a line from stdin into the buffer pointed to by `s` until either a terminating newline or EOF.
- The `int puts(const char *s)` function writes the string `s` and a trailing newline to stdout.

```
#include <stdio.h>
int main( )
{
    int c;

    printf( "Enter a value :");
    c = getchar( );

    printf( "\nYou entered: ");
    putchar( c );

    return 0;
}
```

```
#include <stdio.h>
int main( )
{
    char str[100];

    printf( "Enter a value :");
    gets( str );

    printf( "\nYou entered: ");
    puts( str );

    return 0;
}
```

Formatted I/O

- The `int scanf(const char *format, ...)` function reads input from the standard input stream `stdin` and scans that input according to format provided.
- The `int printf(const char *format, ...)` function writes output to the standard output stream `stdout` and produces output according to a format provided (optional).

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

```
int main ()
{
    /* My Second C Code */
    char name[100];
    printf("Enter your name:");
    scanf("%s",name);
    printf("Hello %s\n",name);
    return 0;
}
```

- The format specifier: `%[flags][width][.precision][length]specifier`

| flag | meaning |
|------|------------------------|
| - | left justify |
| + | always display sign |
| 0 | pad with leading zeros |

Control Structures: for, if & switch

