

C Programming and Numerical Analysis

An Introduction

Synthesis Lectures on Mechanical Engineering

Synthesis Lectures on Mechanical Engineering series publishes 60–150 page publications pertaining to this diverse discipline of mechanical engineering. The series presents Lectures written for an audience of researchers, industry engineers, undergraduate and graduate students.

Additional Synthesis series will be developed covering key areas within mechanical engineering.

[C Programming and Numerical Analysis: An Introduction](#)

Seiichi Nomura
2018

[Mathematical Magnetohydrodynamics](#)

Nikolas Xiros
2018

[Design Engineering Journey](#)

Ramana M. Pidaparti
2018

[Introduction to Kinematics and Dynamics of Machinery](#)

Cho W. S. To
2017

[Microcontroller Education: Do it Yourself, Reinvent the Wheel, Code to Learn](#)

Dimosthenis E. Bolanakis
2017

[Solving Practical Engineering Mechanics Problems: Statics](#)

Sayavur I. Bakhtiyarov
2017

[Unmanned Aircraft Design: A Review of Fundamentals](#)

Mohammad Sadraey
2017

Introduction to Refrigeration and Air Conditioning Systems: Theory and Applications

Allan Kirkpatrick

2017

Resistance Spot Welding: Fundamentals and Applications for the Automotive Industry

Menachem Kimchi and David H. Phillips

2017

MEMS Barometers Toward Vertical Position Detecton: Background Theory, System Prototyping, and Measurement Analysis

Dimosthenis E. Bolanakis

2017

Engineering Finite Element Analysis

Ramana M. Pidaparti

2017

Copyright © 2018 by Morgan & Claypool

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews, without the prior permission of the publisher.

C Programming and Numerical Analysis: An Introduction

Seiichi Nomura

www.morganclaypool.com

ISBN: 9781681733111 paperback

ISBN: 9781681733128 ebook

ISBN: 9781681733135 hardcover

DOI 10.2200/S00835ED1V01Y201802MEC013

A Publication in the Morgan & Claypool Publishers series

SYNTHESIS LECTURES ON MECHANICAL ENGINEERING

Lecture #13

Series ISSN

Print 2573-3168 Electronic 2573-3176

C Programming and Numerical Analysis

An Introduction

Seiichi Nomura

The University of Texas at Arlington

SYNTHESIS LECTURES ON MECHANICAL ENGINEERING #13



MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This book is aimed at those in engineering/scientific fields who have never learned programming before but are eager to master the C language quickly so as to immediately apply it to problem solving in numerical analysis. The book skips unnecessary formality but explains all the important aspects of C essential for numerical analysis. Topics covered in numerical analysis include single and simultaneous equations, differential equations, numerical integration, and simulations by random numbers. In the Appendices, quick tutorials for gnuplot, Octave/MATLAB, and FORTRAN for C users are provided.

KEYWORDS

C, numerical analysis, Unix, gcc, differential equations, simultaneous equations, Octave/MATLAB, FORTRAN, gnuplot

Contents

| | | |
|----------|--|--------------|
| | Preface | xi |
| | Acknowledgments | xiii |
| | PART I Introduction to C Programming | 1 |
| 1 | First Steps to Run a C Program | 5 |
| | 1.1 A Cycle of C Programming | 5 |
| | 1.2 UNIX Command Primer | 8 |
| | 1.3 Overview of C Programming | 10 |
| | 1.3.1 Principles of C language | 10 |
| | 1.3.2 Skeleton C program | 11 |
| | 1.4 Exercises | 14 |
| 2 | Components of C Language | 17 |
| | 2.1 Variables and Data Types | 17 |
| | 2.1.1 Cast Operators | 18 |
| | 2.1.2 Examples of Data Type | 19 |
| | 2.2 Input/Output | 20 |
| | 2.3 Operators between Variables | 22 |
| | 2.3.1 Relational Operators | 23 |
| | 2.3.2 Logical Operators | 23 |
| | 2.3.3 Increment/Decrement/Substitution Operators | 24 |
| | 2.3.4 Exercises | 25 |
| | 2.4 Control Statements | 25 |
| | 2.4.1 if Statement | 26 |
| | 2.4.2 for Statement | 27 |
| | 2.4.3 while Statement | 31 |
| | 2.4.4 do while Statement | 32 |
| | 2.4.5 switch Statement | 33 |

| | | |
|--------|---|----|
| 2.4.6 | Miscellaneous Remarks | 34 |
| 2.4.7 | Exercises | 38 |
| 2.5 | Functions | 39 |
| 2.5.1 | Definition of Functions in C | 39 |
| 2.5.2 | Locality of Variables within a Function | 41 |
| 2.5.3 | Recursivity of Functions | 42 |
| 2.5.4 | Random Numbers, rand() | 44 |
| 2.5.5 | Exercises | 50 |
| 2.6 | Arrays | 52 |
| 2.6.1 | Definition of Arrays | 52 |
| 2.6.2 | Multi-dimensional Arrays | 54 |
| 2.6.3 | Examples | 55 |
| 2.6.4 | Exercises | 58 |
| 2.7 | File Handling | 60 |
| 2.7.1 | I/O Redirection (Standard Input/Output Redirection) | 60 |
| 2.7.2 | File Handling (From within a Program) | 61 |
| 2.8 | Pointers | 63 |
| 2.8.1 | Address Operator & and Dereferencing Operator * | 63 |
| 2.8.2 | Properties of Pointers | 65 |
| 2.8.3 | Function Arguments and Pointers | 68 |
| 2.8.4 | Pointers and Arrays | 70 |
| 2.8.5 | Function Pointers | 72 |
| 2.8.6 | Summary | 73 |
| 2.8.7 | Exercises | 74 |
| 2.9 | String Manipulation | 75 |
| 2.9.1 | How to Handle a String of Characters (Text) | 75 |
| 2.9.2 | String Copy/Compare/Length | 78 |
| 2.10 | Command Line Arguments | 80 |
| 2.10.1 | Entering Command Line Arguments | 80 |
| 2.10.2 | Exercises | 82 |
| 2.11 | Structures | 83 |
| 2.11.1 | Mixture of Different Types of Variables | 83 |
| 2.11.2 | Exercises | 86 |

| | | |
|----------------|---|------------|
| PART II | Numerical Analysis | 89 |
| 3 | Note on Numerical Errors | 93 |
| 4 | Roots of $f(x) = 0$ | 99 |
| 4.1 | Bisection Method | 99 |
| 4.2 | Newton's Method | 102 |
| 4.2.1 | Newton's Method for a Single Equation | 102 |
| 4.2.2 | Newton's Method for Simultaneous Equations (Optional) | 106 |
| 4.2.3 | Exercises | 108 |
| 5 | Numerical Differentiation | 109 |
| 5.1 | Introduction | 109 |
| 5.2 | Forward/Backward/Central Difference | 109 |
| 5.3 | Exercises | 114 |
| 6 | Numerical Integration | 115 |
| 6.1 | Introduction | 115 |
| 6.2 | Rectangular Rule | 115 |
| 6.3 | Trapezoidal Rule | 117 |
| 6.4 | Simpson's Rule | 118 |
| 6.5 | Exercises | 121 |
| 7 | Solving Simultaneous Equations | 123 |
| 7.1 | Introduction | 123 |
| 7.2 | Gauss-Jordan Elimination Method | 126 |
| 7.3 | LU Decomposition (Optional) | 129 |
| 7.4 | Gauss-Seidel Method (Jacobi Method) | 133 |
| 7.5 | Exercises | 135 |
| 8 | Differential Equations | 137 |
| 8.1 | Initial Value Problems | 137 |
| 8.1.1 | Euler's Method | 138 |
| 8.1.2 | Runge-Kutta Method | 143 |
| 8.2 | Higher-order Ordinary Differential Equations | 144 |
| 8.3 | Exercises | 146 |

| | | |
|----------|--|------------|
| A | Gnuplot | 149 |
| B | Octave (MATLAB) Tutorial for C Programmers | 153 |
| | B.1 Introduction | 153 |
| | B.2 Basic Operations | 153 |
| | B.2.1 Principles of Octave/MATLAB | 153 |
| | B.2.2 Reserved Constants | 155 |
| | B.2.3 Vectors/Matrices | 156 |
| | B.2.4 Graph | 158 |
| | B.2.5 I/O | 159 |
| | B.2.6 M-files | 160 |
| | B.2.7 Conditional Statement | 161 |
| | B.3 Sketch of Comparison Between C and Octave/MATLAB | 162 |
| | B.4 Exercises | 167 |
| C | FORTRAN Tutorial for C Programmers | 169 |
| | C.1 FORTRAN Features | 169 |
| | C.2 How to Run a FORTRAN Program | 170 |
| | C.3 Sketch of Comparison Between C and FORTRAN | 171 |
| | C.4 Exercises | 178 |
| | Author's Biography | 181 |
| | Index | 183 |

Preface

This book is aimed at those who want to learn the basics of programming quickly with immediate applications to numerical analysis in mind. It is suitable as a textbook for sophomore-level STEM students.

The book has two goals as the title indicates: The first goal is to introduce the concept of computer programming using the C language. The second goal is to apply the programming skill to numerical analysis for problems arising in scientific and engineering fields. No prior knowledge of programming is assumed but it is desirable that the readers have a background in sophomore-level calculus and linear algebra.

C was selected as the computer language of choice in this book. There have been continuous debates as to what programming language should be taught in college. Until around the 1990s, FORTRAN had been the dominating programming language for scientific and engineering computation which was gradually taken over by modern programming languages as PASCAL and C. Today, MATLAB is taught in many universities as a first computer application/language for STEM students. Python is also gaining popularity as a general purpose programming language suitable as the first computer language to be taught.

Despite many options for the availability of various modern computer languages today, adopting C for scientific and engineering computation still has several merits. C contains almost all the concepts and syntax used in the modern computer languages less the paradigm of object-oriented programming (use C++ and Java for that). It has been observed that whoever learns C first can easily acquire other programming languages and applications such as MATLAB quickly. The converse, however, does not hold. C is a compiled language and preferred over interpreted languages for programs that require fast execution.

There is no shortage of good textbooks for the C language and good textbooks for numerical analysis on the market but a proper combination of both seems to be hard to find. This book is not a complete reference for C and numerical analysis. Instead, the book tries to minimize the formality and limits the scope of C to these essential features that are absolutely necessary for numerical analysis. Some features in C that are not relevant to numerical analysis are not covered in this book. C++ is not covered either as the addition of object-oriented programming components offers little benefit for numerical analysis. After finishing this book, the reader should be able to work on many problems in engineering and science by writing their own C programs.

The book consists of two parts. In Part I, the general syntax of the C language is introduced and explained in details. gcc is used as the compiler which is freely available on almost all platforms. As the native platform of gcc is UNIX, a minimum introduction to the UNIX operating system is also presented.

In Part II the major topics from numerical analysis are presented and corresponding C programs are listed and explained. The subjects covered in Part II include solving a single equation, numerical differentiation, numerical integration, solving a set of simultaneous equations, and solving differential equations.

In Appendix A, `gnuplot` which is a visualization application is introduced. The C language itself has no graphical capabilities and requires an external program to visualize the output from the program.

In Appendix B, a brief tutorial of Octave/MATLAB is given. This is meant for those who are familiar with C but need to learn Octave/MATLAB in the shortest possible amount of time.

In Appendix C, a brief tutorial of FORTRAN is given. Again, this is meant for those who are already familiar with C to be able to read programs written in FORTRAN (FORTRAN 77) quickly.

This book is based on the course notes used for sophomore-level students of the Mechanical and Aerospace Engineering major at The University of Texas at Arlington.

Seiichi Nomura
March 2018

Acknowledgments

I want to thank the students who took this course for their valuable feedback. I also want to thank Paul Petralia of Morgan & Claypool Publishers and C.L. Tondo of T&T TechWorks, Inc. for their support and encouragement.

All the programs and tools used in this book are freely available over the internet thanks to the noble vision of the GNU project and the Free Software Foundation (FSF).

Seiichi Nomura
March 2018

PART I

Introduction to C Programming

In Part I, the basic syntax of the C language is introduced so that you can quickly write a program for problems in science and engineering to be discussed in Part II. This is never meant to be a complete reference for the C language. It covers only those items relevant to scientific/engineering computation. However, after Part I, you should be able to explore missing topics on your own. A minimum amount of computer environments is needed and all the programs listed should run on any version of gcc.

The only way to learn programming is to write a program by yourself. You never learn programming if you just read books sitting on a sofa.

CHAPTER 1

First Steps to Run a C Program

In this chapter, the basic cycle of running a C program is explained. To execute a C program, it is necessary to first write a C code using a text editor, save the code with the file extension, “.c”, launch a C compiler to translate the text into an binary code, and, if everything goes well, run an executable (called a .out in UNIX). If this is the first time you program in C, it is important that you try every single step described in the following sections.

1.1 A CYCLE OF C PROGRAMMING

There are a variety of ways to access a C compiler and run a C program. Almost all schools run a UNIX server open to the students. You should be able to activate your account on the UNIX server, connect to the server via an `ssh`¹ client such as PuTTY over the internet, and run a freely distributed C compiler, `gcc`.²

It is also possible to have a similar setup at home by running your own Linux server or installing a PC/Mac version of `gcc`. If you come from a Windows or Mac environment, you are accustomed to the graphical user interface (GUI) clicking an icon to open an application. However, to use `gcc`, you must use the character-based interface (CUI) to compile and run your program in a UNIX shell, in a command line (DOS) window (Windows) or in Terminal App (Mac). To run `gcc` on the Windows system, you can go to www.mingw.org and download the `gcc` installer, `mingw-get-setup.exe`.

In what follows in this book, we use PuTTY³ (terminal emulation software) to access a UNIX server and run `gcc` on the server.

Figure 1.1 shows an opening screen when PuTTY is launched on the Windows system. In the box circled, enter the name of a server that runs `gcc` and press the Open button. It will prompt you to enter your username (case sensitive) and password (won't echo back). Once you are logged on the server, you are prompted to enter a command from the console (see Figure 1.2). If you have never used a UNIX system before, you may want to play with some of the essential UNIX commands.

Try the following:

1. Login to the server via PuTTY.

¹`ssh` (Secure Shell) is a networking protocol by which two computers are connected via a secure channel.

²`gcc` is an abbreviation for GNU Compiler Collection. It is a compiler system produced by the GNU Project.

³PuTTY is a free and open-source terminal emulator available for the Windows system that can be downloaded from www.putty.org. The size of the executable is less than 1 MB and the program loads very fast.

6 1. FIRST STEPS TO RUN A C PROGRAM

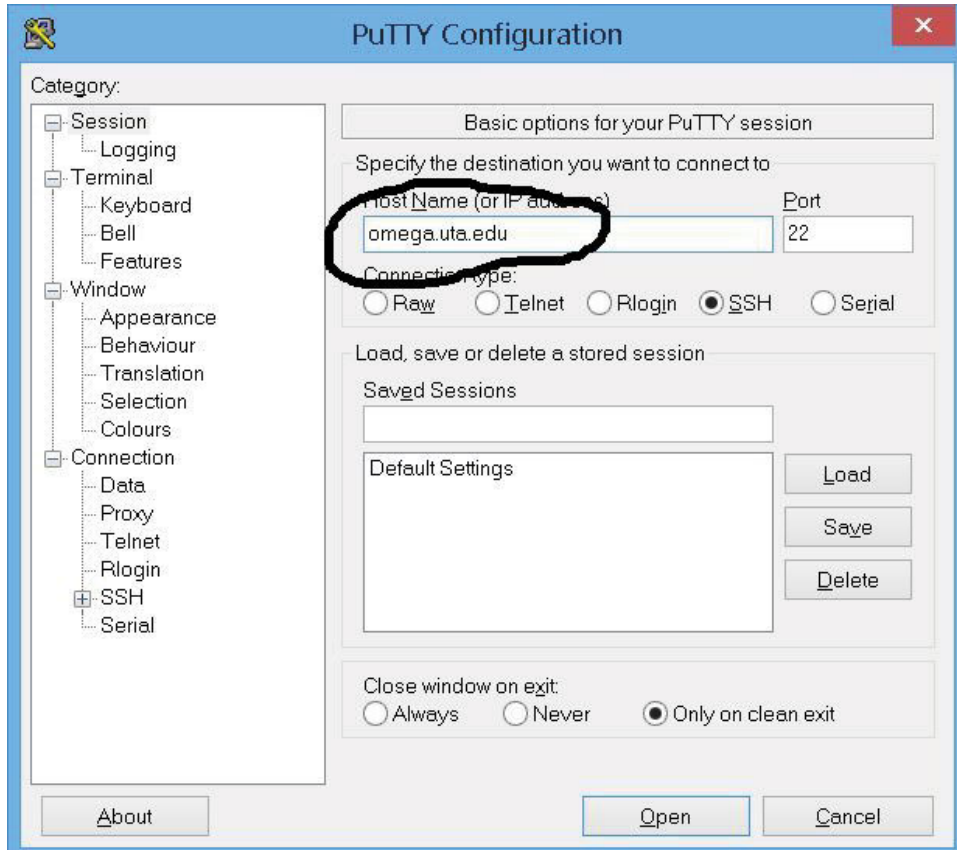


Figure 1.1: Opening screen of PuTTY.

2. Using nano,⁴ a simple text editor, compose your C program (Figure 1.3).

```
$ nano MyProgram.c
```

The symbol, \$, is the system prompt so do not type it. Enter the following text into nano. Note that all the input in UNIX is case-sensitive.

```
#include <stdio.h>
int main()
{
printf("Hello, World!\n");
return 0;
}
```

⁴nano is a simple editor that comes with all the installation of UNIX. It is a clone of another simple text editor, pico.



Figure 1.2: A UNIX session in PuTTY.

3. After you finish entering the text, save the file (Control-O⁵) by entering `MyProgram.c`⁶ as the file name to be saved and press Control-X to exit from nano. This will save the file you just created permanently under the name of `MyProgram.c`.
4. The file you created with nano is a text file that is not understood by the computer. It is necessary to translate this text file into a code which can be run on the computer. This translation process is called compiling and the software to do this translation is called a compiler. We use `gcc` for this purpose.

At the system prompt (`$`), run a C compiler (`gcc`) to generate an executable file (`a.out`⁷).

```
$ gcc MyProgram.c
```

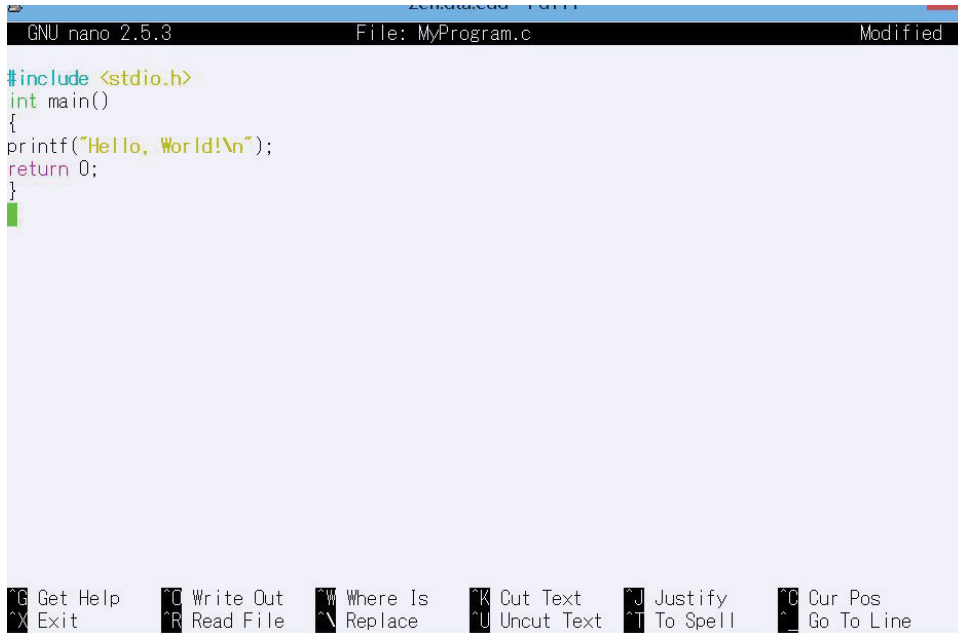
If everything works, `gcc` will create an executable binary file whose default name is `a.out`.

⁵Hold down the control key and press O.

⁶The file name is case sensitive.

⁷`a.out` is an abbreviation for assembler output.

8 1. FIRST STEPS TO RUN A C PROGRAM



```
GNU nano 2.5.3 File: MyProgram.c Modified
#include <stdio.h>
int main()
{
printf("Hello, World!\n");
return 0;
}

Get Help Write Out Where Is Cut Text Justify Cur Pos
Exit Read File Replace Uncut Text To Spell Go To Line
```

Figure 1.3: A nano session in PuTTY.

5. Run the executable file.

```
$ ./a.out8
```

6. If there is a syntax error, go back to item 2 and reissue nano.

```
$ nano MyProgram.c
```

7. If there is no syntax error, run the executable file.

```
$ ./a.out
```

8. To logoff from the server, enter `exit`, `logout`, or hit control-D.

1.2 UNIX COMMAND PRIMER

In a perfect world, you could compose a C program, compile it, and run `a.out` and you are done with it. This scenario may work for a program of less than 10 lines but as the size of the program grows or the program depends on other modules, it is necessary to manipulate and organize files on the UNIX system. Even though this is not an introductory book of the UNIX operating

⁸`./` represents the current directory. If the current directory is included in the `PATH` environmental variable, `./` is not necessary.

system, a minimum amount of knowledge about the UNIX operating system is needed. The following are some of the UNIX commands that are used often. Try each command yourself from the system prompt and find out what it does. It won't damage the machine.

- `ls` (Directory listing.)
- `ls -l` (Directory listing in long format.)
- `ls -lt | more` (Directory listing, one screen at one time, long format, chronological order.)
- `dir` (alias for `ls`)
- `ls .` (Lists the current directory.)
- `cd ..` (Moves to the directory one level up.)
- `pwd` (Shows the present working directory.)
- `cd /` (Moves to the top directory.)
- `cd` (Returns to the home directory.)
- `mkdir MyNewFolder` (Creates a new directory.)
- `nano myfile.txt` (Creates a new file.)
- `cp program1.c program2.c` (Copies `program1.c` to `program2.c`.)
- `mv old.c new.c` (Renames `old.c` to `new.c`.)
- `rm program.c` (Deletes `program.c`.)
- `rm *.c` (Do not do this. It will delete all the files with extension `c`.)
- `whoami` (Shows your username.)
- `who` (Shows who are logged on.)
- `cal` (Shows this year's calendar.)
- `cal 1980` (Shows the calendar of 1980.)

To quickly move while entering/editing a command line and in `nano` sessions, master the following shortcuts. `^f` means holding down the control key and pressing the `f` key.

- `^f` (Moves cursor forward by one character, `f` for forward.)
- `^b` (Moves cursor backward by one character, `b` for backward.)

10 1. FIRST STEPS TO RUN A C PROGRAM

- `^d` (Deletes a character on cursor, d for delete.)
- `^k` (Deletes entire line, k for kill.)
- `^p` (Moves to previous line, same as up arrow, p for previous.)
- `^n` (Moves to next line, same as down arrow, n for next.)
- `^a` (Moves to top of line, a for the first alphabet.)
- `^e` (Moves to end of line, e for end.)

1.3 OVERVIEW OF C PROGRAMMING

Arguably, the most important book on the C language is a book known as “K&R” written by Kernighan and Ritchie⁹ who themselves developed the C language. It is concise yet well-written and is highly recommended for reading.

1.3.1 PRINCIPLES OF C LANGUAGE

Surprisingly, the C language is based on a few simple principles. They are summarized as follows:

1. A C program is a set of functions.
2. A function in C is a code that follows the syntax below:

```
type name(type var)
{
your C code here.....
.....
return value;
}
```

3. A function must be defined before it is used.
4. A function must return a value whose type must be declared (one of `int`, `float`, `double`, `char`). The last line of a function must be a `return xx` statement where `xx` is a value to be returned upon exit.
5. A function must take arguments and must have a placeholder `()` even if there is no argument.
6. The content of a function must be enclosed by “{” and “}”.

⁹Kernighan and Ritchie, *C Programming Language*, 2nd ed., Prentice Hall, 1988.

7. A special function, `int main()`, is the one which is executed first. It is recommended that this function returns an integer value of 0.
8. All the variables used within a function must be declared.

1.3.2 SKELETON C PROGRAM

The following program is absolutely the smallest C program that can be written:

```
int main()
{
return 0;
}
```

You can compile and execute this program by issuing the following commands:

```
$ gcc MyProgram.c
$ ./a.out
```

where `MyProgram.c` is the name under which the file was saved. Even though it is the smallest C program, the program itself is a full-fledged C code. Of course, this program does nothing and when you issue `./a.out`, the program simply exits after being executed and you are returned to the system prompt.

Here is a line-by-line analysis of the program above. Refer to Section 1.3.1 for the list of items. The first line, `int main()`, indicates that a function whose name is `main` is declared that returns an integer value (`int`) upon exit (Item 4). This function takes no arguments (empty parameters within the parentheses) (Item 5). The program consists of only one function, `main()`, which is executed first (Item 7). The content of the function, `main()`, is the line(s) surrounded by `{` and `}` (Item 6). In this case, the program executes the `return 0` statement and exits back to the operating system returning a 0 value to the operating system. As C is a free-form language, the end of each statement has to be clearly marked. A semicolon `;` is placed at the end of each statement. Hence, `return 0;`.

The following program is a celebrated code that appeared first in the K&R book in the *Getting Started* section and later adapted in just about every introductory book for C as the first C program that prints “Hello, World!” followed by an extra blank line.

```
1:#include <stdio.h>
2:int main()
3:{
4:printf("Hello, World!\n");
5:return 0;
6:}
```

12 1. FIRST STEPS TO RUN A C PROGRAM

Each line in the above program is now parsed. The first line, `#include <stdio.h>`, is a bit confusing but let's skip this line for the time being and move to the subsequent lines. If you compile the program and execute `a.out`, you will find out that the program prints `Hello, World!` followed by a new line on the screen. Hence, you can guess that the odd characters, `\n`, represents a blank line. As there is no character that represents a blank line, you figure out that `\n` can be used as printing a blank line.

Next, note that the part `printf` is followed by a pair of parentheses and therefore, it is a function in C (Item 5). It is obvious that this function, `printf()`, prints a string, `Hello, World!`, and quits. As it is a function in C, it has to be defined and declared before it is used. However, no such definition is found above the function, `main()`. The first line, `#include <stdio.h>`, is in fact referring to a file that contains the definition of `printf()` that is preloaded before anything else. The file, `stdio.h`, is one of the header (hence the extension, `h`) files available in the C library that is shipped with `gcc`. As the name indicates (`stdio = Standard Input and Output`), this header file has the definition of many functions that deal with input and output (I/O) functions.

Finally, a function must have information about the type of the value it returns such as `int`, `float`, `double`, etc... (Item 4). In this case, the function `int main()` is declared to return an integer value upon exit. Sure enough, the last statement `return 0;` is to return 0 when the execution is done and 0 is an integer.

Here is how `gcc` parses this program line by line:

Line 1 Before anything else, let's load a header file, `<stdio.h>`, that contains the definition of all the functions that deal with I/O from the system area.

Line 2 This is the start of a function called `main()`. This function returns an integer value `int` upon exit. This function has no parameters to pass so the content within the parentheses is empty.

Line 3 The `{` character indicates that this is the beginning of the content of the function, `main()`.

Line 4 This line calls the function, `printf()`, that is defined in `<stdio.h>` and prints out a string of `Hello, World!` followed by a blank line. A semicolon, `;`, marks the end of this function.

Line 5 This is the last statement of the function, `main()`. It will return the value 0 to the operating system and exit.

Line 6 The `}` character indicates the end of the content of the function, `main()`.

You can execute this program by

```
$ nano hello.c
```

(Enter the content of the program above.)

```
$ gcc hello.c
```

(If it is not compiled, reedit hello.c.)

```
$ ./a.out
```

```
Hello, World!
```

Here is another program that does some scientific computation.

```
1:#include <stdio.h>
2:#include <math.h>
3: /* This is a comment */
4:int main()
5: {
6: float x, y;
7: x = 6.28;
8: y=sin(x);
9: printf("Sine of %f is %f.\n", x, y);
10: return 0;
11:}
```

This program computes the value of $\sin x$ where $x = 6.28$. The program can be compiled as

```
$ gcc MyProgram.c -lm
```

Note that the `-lm`¹⁰ option is necessary when including `<math.h>`.¹¹

Here is a line by line analysis of the program:

Line 1 The program preloads a header file, `<stdio.h>`.

Line 2 The program also preloads an another header file, `<math.h>`. This header file is necessary whenever mathematical functions such as $\sin(x)$ are used in the program.

Line 3 This entire line is a comment. Anything surrounded by `/*` and `*/` is a comment and is ignored by the compiler.¹²

Line 4 This is the declaration of a function, `main()`, that returns an integer value but with no parameter.

¹⁰“-l” is to load a library and “m” stands for the math library.

¹¹`<math.h>` only contains protocol declarations for mathematical functions. It is necessary to locally load the mathematical library, `libm.a` by the `-lm` option.

¹²A comment can also start with `//`. This for one-line comment originated in C++.

14 1. FIRST STEPS TO RUN A C PROGRAM

Line 5 The `{` character indicates that this is the beginning of the content of the function, `main()`.

Line 6 Two variables, `x` and `y`, are declared both of which represent floating numbers.

Line 7 The variable, `x`, is assigned a floating number, `6.28`.

Line 8 The function, `sin(x)`, is evaluated where `x` is `6.28` and the result is assigned to the variable, `y`.

Line 9 The result is printed. First, a literal string of “Sine of” is printed followed by the actual value of `x` and “is” is printed followed by the actual value of `y`, a period and a new line.

Line 10 The function, `main()`, exits with a return value of `0`.

Line 11 The `}` character indicates that this is the end of the content of the function, `main()`.

There are several new concepts in this program that need to be explained. The second line is to preload yet another header file, `math.h`, as this program computes the sine of a number. In the fourth line, two variables, `x` and `y`, are declared. The `float` part indicates that the two variables represent floating numbers (real numbers with the decimal point). The fifth line says that a number, `6.28`, is assigned to the variable, `x`. The equal sign (`=`) here is not the mathematical equality that you are accustomed to. In C and all other computer languages, an equal sign (`=`) is exclusively used for substitution, i.e., the value to the right of `=` is assigned to the variable to the left of `=`. In the eighth line, the `printf()` function is to print a list of variables (`x` and `y`) with formatting specified by the double quotation marks (“...”). The way formatting works is that `printf()` prints everything literally within the parentheses except for special codes starting with the percentage sign (`%`). Here, `%f` represents a floating number which is to be replaced by the actual value of the variable. As there are two `%f`'s, the first `%f` is replaced by the value of `x` and the second `%f` is replaced by the value of `y`. The details of the new concepts shown here will be explained in detail in Chapter 2.

1.4 EXERCISES

It is not necessary to know all the syntax of C to work on the following problems. Each problem has a template that you can modify. Start with the template code, keep modifying the code and understand what each statement does. It is essential that you actually write the code yourself (not copy and paste) and execute it.

1. Write a C program to print three blank lines followed by “Hello, World!”. Use the following code as a template:

```
#include <stdio.h>
int main()
{
printf("\nHello, World!\n\n");
return 0;
}
```

`\n` prints a new line.

- Write a program to read two real numbers from the keyboard and to print their product. Use the following code as a template. Do not worry about the syntax, just modify one place.

```
#include <stdio.h>
int main()
{
int a, b; /* to declare that a and b are integer variables */
printf("Enter two integer numbers separated by space =");
scanf("%d %d", &a, &b); /* This is the way to read two integer
numbers and assign them to a and b. */
printf("The sum of the two numbers is %d.\n", a+b); /* %d is for
integer format. */
return 0;
}
```

- Write a program to read a real number, x , and outputs its sine, i.e., $\sin(x)$. You need to use `<math.h>` and the `-lm` compile option. Use the following template program that computes e^x .

```
#include <stdio.h>
#include <math.h>
int main()
{
float x;
printf("Enter a number ="); scanf("%f", &x);
printf("x= %f exp(x)=%f\n",x, exp(x));
return 0;
}
```

16 1. FIRST STEPS TO RUN A C PROGRAM

You have to use the `-lm` option when compiling:

```
$ gcc MyProgram.c -lm  
$ ./a.out
```

CHAPTER 2

Components of C Language

In this chapter, the essential components of the C language are introduced and explained. The syntax covered in this chapter is not exhaustive but after this chapter you should be able to write a simple C program that can solve many problems in engineering and science.

2.1 VARIABLES AND DATA TYPES

Every single variable used in C must have a type which the value of the variable represents. There are four variable types listed in Table 2.1.

Table 2.1: Data types

| Type | Content | Format | Range | Example |
|---------------------|------------------|------------------|--|------------------|
| <code>int</code> | Integer | <code>%d</code> | $-2147483647 \sim +2147483647$ | 10 |
| <code>float</code> | Floating number | <code>%f</code> | $\pm 2.9387e - 39 \sim \pm 1.7014e + 38$ | 3.14 |
| <code>double</code> | Double precision | <code>%lf</code> | $2^{-63} \sim 2^{+63}$ | 3.14159265358979 |
| <code>char</code> | Character | <code>%c</code> | ASCII code | 'a' |

In Table 2.1, the third column shows the format of each data type which is used in the `print()` and `scanf()` functions.

- `int` represents an integer value. The range of `int` depends on the hardware and the version of the compiler. In most modern systems, `int` represents from -2147483647 to 2147483647 .
- `float` represents a floating number. This will take care of most non-scientific floating numbers (single precision). For scientific and engineering computation, `double` must be used.
- `double` is an extension of `float`. This data type can handle a larger floating number at the expense of the amount of memory used (but not much).
- `char` represents a single ASCII character. This data type is actually a subset of `int` in which the range is limited to $0 \sim 255$. The character represented by `char` must be enclosed by a single quotation mark (').

18 2. COMPONENTS OF C LANGUAGE

2.1.1 CAST OPERATORS

When an operation between variables of different types is performed, the variables of a lower type are automatically converted to the highest type following this order:

`int=char < float < double`

For example, for `a * b` in which `a` is of `int` type and `b` is of `float` type, then, `a` is converted to the `float` type automatically and the result is also of `float` type. There are times when two variables are both `int` type yet the result of the operation is desired to be of `float` type. For example,

```
#include <stdio.h>
int main()
{
    int a, b;
    a=3; b=5;
    printf("%f\n", a/b);
    return 0;
}
```

The output is

```
$ gcc prog.c
2.c: In function 'main':
2.c:6:10: warning: format '%f' expects argument of type 'double',
        but argument 2 has type 'int' [-Wformat=]
    printf("%f\n", a/b);
           ^
$ ./a.out
-0.000000
```

It prints 0 with a warning even though the result is expected to be 0.6. To carry out this operation as intended,¹ a cast operator (an operator to allow to change the type of a variable to a specified type temporarily) must be used as

```
#include <stdio.h>
int main()
{
    int a,b;
```

¹Another way of achieving this is to modify `a/b` to `1.0*a/b`.


```

a=3;b=5;
printf("%f\n", (float)a/b);
return 0;
}

```

The output is

```

$ gcc prog.c
$ ./a.out
0.600000

```

The `(float)a/b` part forces both variables to be of `float` type and returns 0.6 as expected.

2.1.2 EXAMPLES OF DATA TYPE

1. This program prints a character, “h”.

```

/*
Print a character
*/
#include <stdio.h>
int main()
{
    char a='h';
    printf("%c\n",a);
    return 0;
}

```

Note that the variable, `a`, is declared as `char` and initialized as “h” on the same line.

2. This program prints an integer 10.

```

/*
Print an integer
*/
#include <stdio.h>
int main()
{
    int a=10;
    printf("%d\n",a);
}

```

```
return 0;
}
```

Note that the variable, `a`, is declared as `int` and initialized as 10 on the same line.

3. This program prints a floating number 10.5.

```
/* Print a floating number */
#include <stdio.h>
int main()
{
    float a=10.5;
    printf("%f\n",a);
    return 0;
}
```

Note that the variable, `a`, is declared as `float` and initialized as 10.5 on the same line.

4. This program prints two floating numbers, 10.0 and -2.3.

```
/* Print floating numbers */
#include <stdio.h>
int main()
{
    float a, b=9.0, c;
    a=10.0; c=-2.3;
    printf("a = %f\n",a);
    printf("c = %f\n",c);
    return 0;
}
```

2.2 INPUT/OUTPUT

Almost all C programs have at least one output statement. Otherwise, the program won't output anything on the screen and there is no knowing if the program ran successfully or not. The most common input/output functions are `printf()` and `scanf()` both of which are defined in the header file `stdio.h`.

Use `printf()` (Print with Format) for outputting data to the console and `scanf()` (Scan with Format) for inputting data from the keyboard.