Let Us C

Fifth Edition

Yashavant P. Kanetkar

Dedicated to baba

Who couldn't be here to see this day...

About the Author

Destiny drew Yashavant Kanetkar towards computers when the IT industry was just making a beginning in India. Having completed his education from VJTI Mumbai and IIT Kanpur in Mechanical Engineering he started his training company in Nagpur.

Yashavant has a passion for writing and is an author of several books in C, C++, VC++, C#, .NET, DirectX and COM programming. He is a much sought after speaker on various technology subjects and is a regular columnist for Express Computers and Developer 2.0. His current affiliations include being a Director of KICIT, a training company and DCube Software Technologies, a software development company. In recognition to his contribution Microsoft awarded him the prestigious ***"Best .NET Technical Contributor"*** award recently. He can be reached at kanetkar@kicit.com.

Preface to the Fifth Edition

It is mid 2004. World has left behind the DOTCOM bust, 9/11 tragedy, the economic downturn, etc. and moved on. Countless Indians have relentlessly worked for close to two decades to successfully establish “India” as a software brand. At times I take secret pleasure in seeing that a book that I have been part of, has contributed in its own little way in shaping so many budding careers that have made the “India” brand acceptable.

Computing and the way people use C for doing it keeps changing as years go by. So overwhelming has been the response to all the previous editions of “Let Us C” that I have now decided that each year I would come up with a new edition of it so that I can keep the readers abreast with the way C is being used at that point in time.

There are two phases in every C programmer’s life. In the first phase he is a learner trying to understand the language elements and their nuances. At this stage he wants a simple learning environment that helps him to master the language. In my opinion, even today there isn’t any learning environment that can beat Turbo C/C++ for simplicity. Hence the first fifteen chapters are written keeping this environment in mind, though a majority of these programs in these chapters would work with any C compiler.

Armed with the knowledge of language elements the C programmer enters the second phase. Here he wishes to use all that he has learnt to create programs that match the ability of programs that he see in today’s world. I am pointing towards programs in Windows and Linux world. Chapters 16 to 21 are devoted to this. I would like to your attention the fact that if you want to program Windows or Linux you need to have a very good grasp over the programming model used by each of these OS. Windows messaging architecture and Linux signaling mechanism are the cases in point. Once you understand these thoroughly rest is just a

matter of time. Chapters 16 to 21 have been written with this motive.

In Linux programming the basic hurdle is in choosing the Linux distribution, compiler, editor, shell, libraries, etc. To get a head-start you can follow the choices that I found most reasonable and simple. They have been mentioned in Chapter 20 and Appendix H. Once you are comfortable you can explore other choices.

In fourth edition of Let Us C there were chapters on 'Disk Basics', 'VDU Basics', 'Graphics', 'Mouse Programming', 'C and Assembly'. Though I used to like these chapters a lot I had to take a decision to drop them since most of them were DOS-centric and would not be so useful in modern-day programming. Modern counterparts of all of these have been covered in Chapters 16 to 21. However, if you still need the chapters from previous edition they are available at www.kicit.com/books/letusc/fourthedition.

Also, all the programs present in the book are available in source code form at www.kicit.com/books/letusc/sourcecode. You are free to download them, improve them, change them, do whatever with them. If you wish to get solutions for the Exercises in the book they are available in another book titled 'Let Us C Solutions'.

'Let Us C' is as much your book as it is mine. So if you feel that I could have done certain job better than what I have, or you have any suggestions about what you would like to see in the next edition, please drop a line to letuscsuggestions@kicit.com.

All the best and happy programming!

Contents

1 Getting Started

- What is C
- Getting Started with C
  - The C Character Set
  - Constants, Variables and Keywords
  - Types of C Constants
  - Rules for Constructing Integer Constants
  - Rules for Constructing Real Constants
  - Rules for Constructing Character Constants
  - Types of C Variables
  - Rules for Constructing Variable Names
  - C Keywords
- The First C Program
- Compilation and Execution
- Receiving Input
- C Instructions
  - Type Declaration Instruction
  - Arithmetic Instruction
  - Integer and Float Conversions
  - Hierarchy of Operations
  - Associativity Of Operators
- Control Instruction in C
- Summary
- Exercise

Before we can begin to write serious programs in C, it would be interesting to find out what really is C, how it came into existence and how does it compare with other computer languages. In this chapter we would briefly outline these issues.

Four important aspects of any language are the way it stores data, the way it operates upon this data, how it accomplishes input and output and how it lets you control the sequence of execution of instructions in a program. We would discuss the first three of these building blocks in this chapter.

What is C

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc. No one pushed C. It wasn't made the 'official' Bell Labs language. Thus, without any advertisement C's reputation spread and its pool of users grew. Ritchie seems to have been rather surprised that so many programmers preferred C to older languages like FORTRAN or PL/I, or the newer ones like Pascal and APL. But, that's what happened.

Possibly why C seems so popular is because it is reliable, simple and easy to use. Moreover, in an industry where newer languages, tools and technologies emerge and vanish day in and day out, a language that has survived for more than 3 decades has to be really good.

An opinion that is often heard today is – "C has been already superceded by languages like C++, C# and Java, so why bother to

learn C today". I seriously beg to differ with this opinion. There are several reasons for this:

(a) I believe that nobody can learn C++ or Java directly. This is because while learning these languages you have things like classes, objects, inheritance, polymorphism, templates, exception handling, references, etc. do deal with apart from knowing the actual language elements. Learning these complicated concepts when you are not even comfortable with the basic language elements is like putting the cart before the horse. Hence one should first learn all the language elements very thoroughly using C language before migrating to C++, C# or Java. Though this two step learning process may take more time, but at the end of it you will definitely find it worth the trouble.

(b) C++, C# or Java make use of a principle called Object Oriented Programming (OOP) to organize the program. This organizing principle has lots of advantages to offer. But even while using this organizing principle you would still need a good hold over the language elements of C and the basic programming skills.

(c) Though many C++ and Java based programming tools and frameworks have evolved over the years the importance of C is still unchallenged because knowingly or unknowingly while using these frameworks and tools you would be still required to use the core C language elements—another good reason why one should learn C before C++, C# or Java.

(d) Major parts of popular operating systems like Windows, UNIX, Linux is still written in C. This is because even today when it comes to performance (speed of execution) nothing beats C. Moreover, if one is to extend the operating system to work with new devices one needs to write device driver programs. These programs are exclusively written in C.

(e) Mobile devices like cellular phones and palmtops are becoming increasingly popular. Also, common consumer devices like microwave oven, washing machines and digital cameras are getting smarter by the day. This smartness comes from a microprocessor, an operating system and a program embedded in this devices. These programs not only have to run fast but also have to work in limited amount of memory. No wonder that such programs are written in C. With these constraints on time and space, C is the language of choice while building such operating systems and programs.

(f) You must have seen several professional 3D computer games where the user navigates some object, like say a spaceship and fires bullets at the invaders. The essence of all such games is speed. Needless to say, such games won't become popular if they takes a long time to move the spaceship or to fire a bullet. To match the expectations of the player the game has to react fast to the user inputs. This is where C language scores over other languages. Many popular gaming frameworks have been built using C language.

(g) At times one is required to very closely interact with the hardware devices. Since C provides several language elements that make this interaction feasible without compromising the performance it is the preferred choice of the programmer.

I hope that these are very convincing reasons why one should adopt C as the first and the very important step in your quest for learning programming languages.

Getting Started with C

Communicating with a computer involves speaking the language the computer understands, which immediately rules out English as the language of communication with computer. However, there is

a close analogy between learning English language and learning C language. The classical method of learning English is to first learn the alphabets used in the language, then learn to combine these alphabets to form words, which in turn are combined to form sentences and sentences are combined to form paragraphs. Learning C is similar and easier. Instead of straight-away learning how to write programs, we must first know what alphabets, numbers and special symbols are used in C, then how using them constants, variables and keywords are constructed, and finally how are these combined to form an instruction. A group of instructions would be combined later on to form a program. This is illustrated in the Figure 1.1.

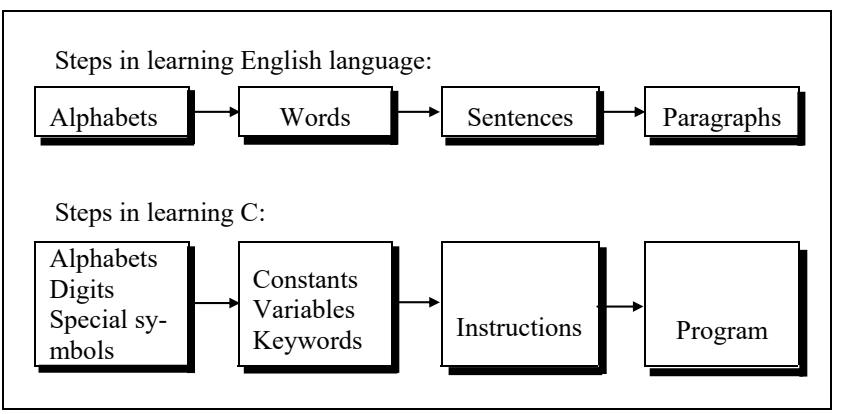

Figure 1.1

The C Character Set

A character denotes any alphabet, digit or special symbol used to represent information. Figure 1.2 shows the valid alphabets, numbers and special symbols allowed in C.