

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	1
What is C	2
Getting Started with C	4
The C Character Set	5
Constants, Variables and Keywords	6
Types of C Constants	7
Rules for Constructing Integer Constants	8
Rules for Constructing Real Constants	9
Rules for Constructing Character Constants	10
Types of C Variables	11
Rules for Constructing Variable Names	11
C Keywords	12
The First C Program	13
Compilation and Execution	19
Receiving Input	21
C Instructions	23
Type Declaration Instruction	24
Arithmetic Instruction	25
Integer and Float Conversions	29
Type Conversion in Assignments	29
Hierarchy of Operations	31
Associativity of Operators	34
Control Instructions in C	37
Summary	37
Exercise	38
 2. The Decision Control Structure	 49
Decisions! Decisions!	50
The <i>if</i> Statement	51
The Real Thing	55
Multiple Statements within <i>if</i>	56
The <i>if-else</i> Statement	58

Nested <i>if-elses</i>	61
Forms of <i>if</i>	62
Use of Logical Operators	64
The <i>else if</i> Clause	66
The ! Operator	72
Hierarchy of Operators Revisited	73
A Word of Caution	73
The Conditional Operators	76
Summary	77
Exercise	78
 3. The Loop Control Structure	 97
Loops	98
The <i>while</i> Loop	99
Tips and Traps	101
More Operators	105
The <i>for</i> Loop	107
Nesting of Loops	114
Multiple Initialisations in the <i>for</i> Loop	115
The Odd Loop	116
The <i>break</i> Statement	118
The <i>continue</i> Statement	120
The <i>do-while</i> Loop	121
Summary	124
Exercise	124
 4. The Case Control Structure	 135
Decisions Using <i>switch</i>	136
The Tips and Traps	140
<i>switch</i> Versus <i>if-else</i> Ladder	144
The <i>goto</i> Keyword	145
Summary	148
Exercise	149

5. Functions & Pointers	157
What is a Function	158
Why Use Functions	165
Passing Values between Functions	166
Scope Rule of Functions	171
Calling Convention	172
One Dicey Issue	173
Advanced Features of Functions	174
Function Declaration and Prototypes	175
Call by Value and Call by Reference	178
An Introduction to Pointers	178
Pointer Notation	179
Back to Function Calls	186
Conclusions	189
Recursion	189
Recursion and Stack	194
Adding Functions to the Library	197
Summary	201
Exercise	201
 6. Data Types Revisited	 213
Integers, <i>long</i> and <i>short</i>	214
Integers, <i>signed</i> and <i>unsigned</i>	216
Chars, <i>signed</i> and <i>unsigned</i>	217
Floats and Doubles	219
A Few More Issues...	221
Storage Classes in C	223
Automatic Storage Class	224
Register Storage Class	226
Static Storage Class	227
External Storage Class	230
Which to Use When	233
Summary	234
Exercise	235

7. The C Preprocessor	241
Features of C Preprocessor	242
Macro Expansion	244
Macros with Arguments	248
Macros versus Functions	252
File Inclusion	253
Conditional Compilation	255
<i>#if</i> and <i>#elif</i> Directives	258
Miscellaneous Directives	260
<i>#undef</i> Directive	260
<i>#pragma</i> Directive	261
Summary	263
Exercise	264
 8. Arrays	 269
What are Arrays	270
A Simple Program Using Array	272
More on Arrays	275
Array Initialization	275
Bounds Checking	276
Passing Array Elements to a Function	277
Pointers and Arrays	279
Passing an Entire Array to a Function	286
The Real Thing	287
Two Dimensional Arrays	289
Initializing a 2-Dimensional Array	290
Memory Map of a 2-Dimensional Array	291
Pointers and 2-Dimensional Arrays	292
Pointer to an Array	295
Passing 2-D array to a Function	297
Array of Pointers	300
Three Dimensional Array	302
Summary	304

Exercise	304
9. Puppetting On Strings	327
What are Strings	328
More about Strings	329
Pointers and Strings	334
Standard Library String Functions	335
<i>strlen()</i>	337
<i>strcpy()</i>	339
<i>strcat()</i>	342
<i>strcmp()</i>	343
Two-Dimensional Array of Characters	344
Array of Pointers to Strings	347
Limitation of Array of Pointers to Strings	351
Solution	352
Summary	353
Exercise	354
10. Structures	363
Why Use Structures	364
Declaring a Structure	367
Accessing Structure Elements	370
How Structure Elements are Stored	370
Array of Structures	371
Additional Features of Structures	374
Uses of Structures	383
Summary	384
Exercise	384
11. Console Input/Output	393
Types of I/O	394
Console I/O Functions	395
Formatted Console I/O Functions	396

<i>sprintf()</i> and <i>scanf()</i> Functions	404
Unformatted Console I/O Functions	405
Summary	409
Exercise	409
12. File Input/Output	415
Data Organization	416
File Operations	417
Opening a File	418
Reading from a File	420
Trouble in Opening a File	421
Closing the File	422
Counting Characters, Tabs, Spaces, ...	422
A File-copy Program	424
Writing to a File	425
File Opening Modes	426
String (line) I/O in Files	427
The Awkward Newline	430
Record I/O in Files	430
Text Files and Binary Files	434
Record I/O Revisited	437
Database Management	441
Low Level Disk I/O	447
A Low Level File-copy Program	448
I/O Under Windows	453
Summary	453
Exercise	454
13. More Issues In Input/Output	465
Using <i>argc</i> and <i>argv</i>	466
Detecting Errors in Reading/Writing	470
Standard I/O Devices	472
I/O Redirection	473
Redirecting the Output	474

Redirecting the Input	476
Both Ways at Once	477
Summary	478
Exercise	478
14. Operations On Bits	481
Bitwise Operators	482
One's Complement Operator	484
Right Shift Operator	486
Left Shift Operator	488
Bitwise AND Operator	493
Bitwise OR Operator	498
Bitwise XOR Operator	499
The <i>showbits()</i> Function	500
Summary	501
Exercise	501
15. Miscellaneous Features	505
Enumerated Data Type	506
Uses of Enumerated Data Type	507
Renaming Data Types with <i>typedef</i>	510
Typecasting	511
Bit Fields	513
Pointers to Functions	515
Functions Returning Pointers	518
Functions with Variable Number of Arguments	520
Unions	524
Union of Structures	530
Summary	531
Exercise	531

16. C Under Windows	535
Which Windows...	536
Integers	537
The Use of <i>typedef</i>	537
Pointers in the 32-bit World	539
Memory Management	540
Device Access	543
DOS Programming Model	543
Windows Programming Model	547
Event Driven Model	551
Windows Programming, a Closer Look	552
The First Windows Program	554
Hungarian Notation	558
Summary	558
Exercise	559
17. Windows Programming	561
The Role of a Message Box	562
Here Comes the window...	563
More Windows	566
A Real-World Window	567
Creation and Displaying of Window	569
Interaction with Window	570
Reacting to Messages	572
Program Instances	575
Summary	575
Exercise	576
18. Graphics Under Windows	579
Graphics as of Now	580
Device Independent Drawing	580

Hello Windows	582
Drawing Shapes	586
Types of Pens	590
Types of Brushes	592
Code and Resources	596
Freehand Drawing, the Paintbrush Style	596
Capturing the Mouse	600
Device Context, a Closer Look	601
Displaying a Bitmap	603
Animation at Work	607
WM_CREATE and <i>OnCreate()</i>	611
WM_TIMER and <i>OnTimer()</i>	611
A Few More Points...	612
Windows, the Endless World...	613
Summary	614
Exercise	615

19. Interaction With Hardware 617

Hardware Interaction	618
Hardware Interaction, DOS Perspective	619
Hardware Interaction, Windows Perspective	623
Communication with Storage Devices	626
The <i>ReadSector()</i> Function	631
Accessing Other Storage Devices	633
Communication with Keyboard	634
Dynamic Linking	635
Windows Hooks	635
Caps Locked, Permanently	637
Did You Press It TTwwiiccee.....	643
Mangling Keys	644
KeyLogger	645
Where is This Leading	646
Summary	647
Exercise	647

20. C Under Linux	649
What is Linux	650
C Programming Under Linux	651
The ‘Hello Linux’ Program	652
Processes	653
Parent and Child Processes	655
More Processes	659
Zombies and Orphans	660
One Interesting Fact	663
Summary	664
Exercise	664
 21. More Linux Programming	 667
Communication using Signals	668
Handling Multiple Signals	671
Registering a Common Handler	673
Blocking Signals	675
Event Driven Programming	678
Where Do You Go From Here	684
Summary	684
Exercise	685
 <i>Appendix A – Precedence Table</i>	 687
<i>Appendix B – Standard Library Functions</i>	691
<i>Appendix C – Chasing the Bugs</i>	701
<i>Appendix D – Hexadecimal Numbering</i>	713
<i>Appendix E – ASCII Chart</i>	719
<i>Appendix F – Helper.h File</i>	725
<i>Appendix G – Boot Parameters</i>	729
<i>Appendix H – Linux Installation</i>	735
<i>Index</i>	739

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 these devices. These programs not only have to run fast but also have to work in a 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 take 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.

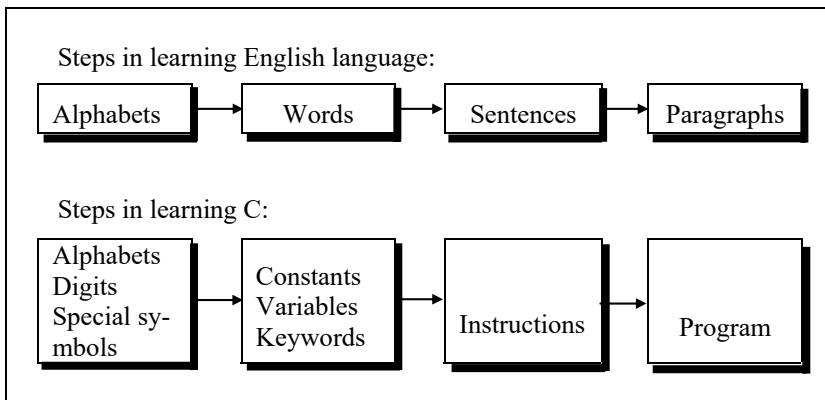


Figure 1.1

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.