UE24CS151B:Problem Solving with C - Introduction for B.Tech Second Semester A & N Section - EC Campus Lecture Slides - Slot #4, #5, #6

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

UE24CS151B: Problem Solving with C - Syllabus

Unit I: Problem Solving Fundamentals - 14 Hours - 18 Slots

Introduction to Programming, Salient Features of ‘C’, Program Structure, Variables, Data Types & range of values ,Qualifiers, Operators and Expressions, Control Structures, Input/Output Functions, Language Specifications -Behaviors, Single character input and output, Coding standards and guidelines

Unit II: Counting, Sorting and Searching – 14 Hours - 18 Slots

Arrays–1D and 2D, Pointers, Pointer to an array, Array of pointers, Functions, Call back, Storage classes, Recursion, Searching, Sorting

Unit III: Text Processing and User-Defined Types - 14 Hours - 18 Slots

Strings, String Manipulation Functions & Error handling, Command line arguments, Dynamic Memory Management functions & Error handling, Structures, #pragma, Array of Structures, Pointer to structures, Passing Structure and Array of structure to a function, Bit fields, Unions, Enums, Lists, Stack, Queue, Priority Queue.

Unit IV: File Handling and Portable Programming – 14 Hours – 18 Slots

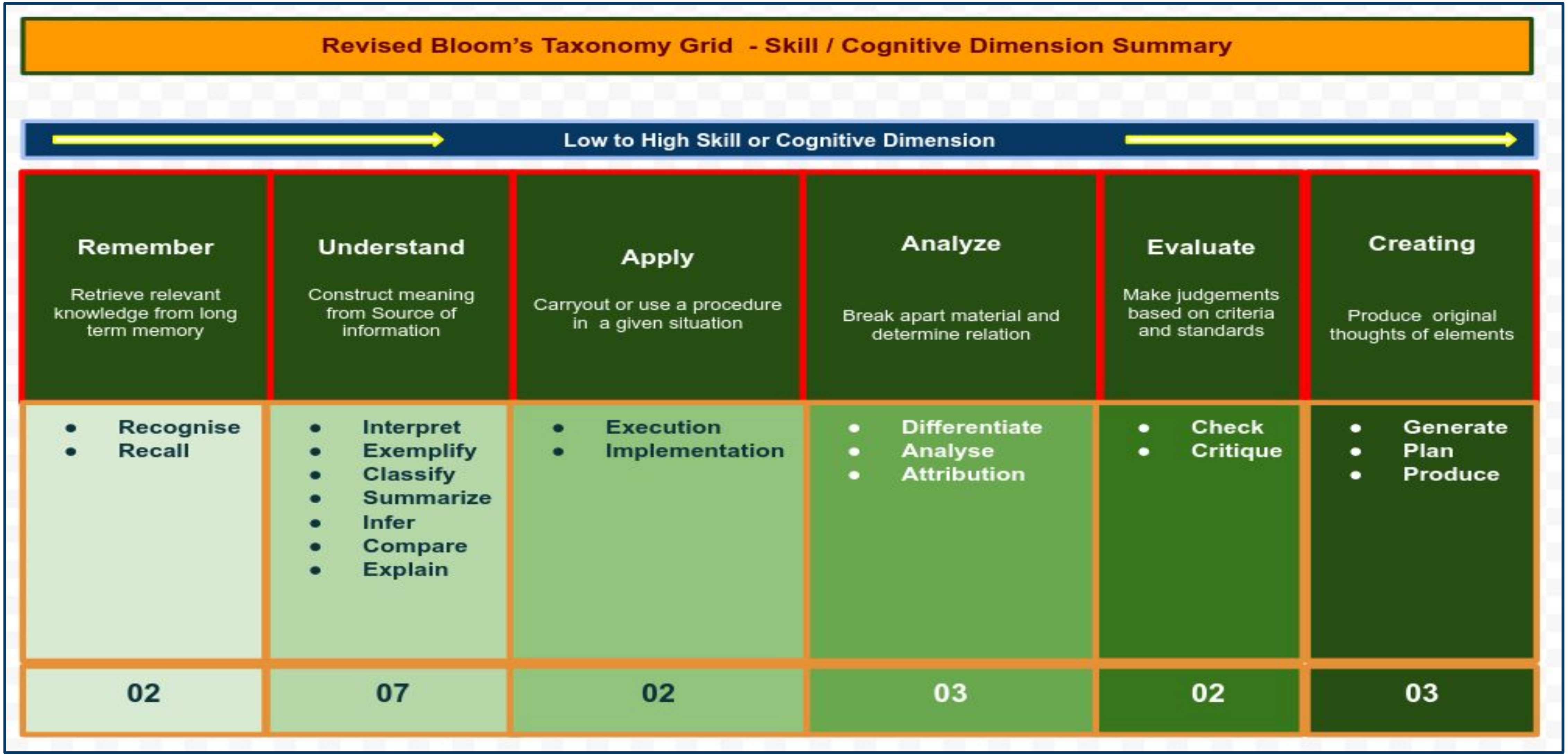
File IO using redirection, File Handling functions of C, Searching, Sorting, Header files, Comparison of relevant User defined and Built-in functions, Variable Length Arguments, Environment variables, Preprocessor Directives, Conditional Compilation.

UE24CS151B: Problem Solving with C - Course Objectives

The objective(s) of this course is to make students

- CObj1: Acquire knowledge on how to solve relevant and logical problems using computing machine
- CObj2: Map algorithmic solutions to relevant features of C programming language constructs
- CObj3: Gain knowledge about C constructs and its associated ecosystem
- CObj4: Appreciate and gain knowledge about the issues with C Standards and its respective behaviours

Bloom's Taxonomy



Revised Bloom's Taxonomy Grid - Skill / Cognitive Dimension Summary

Low to High Skill or Cognitive Dimension

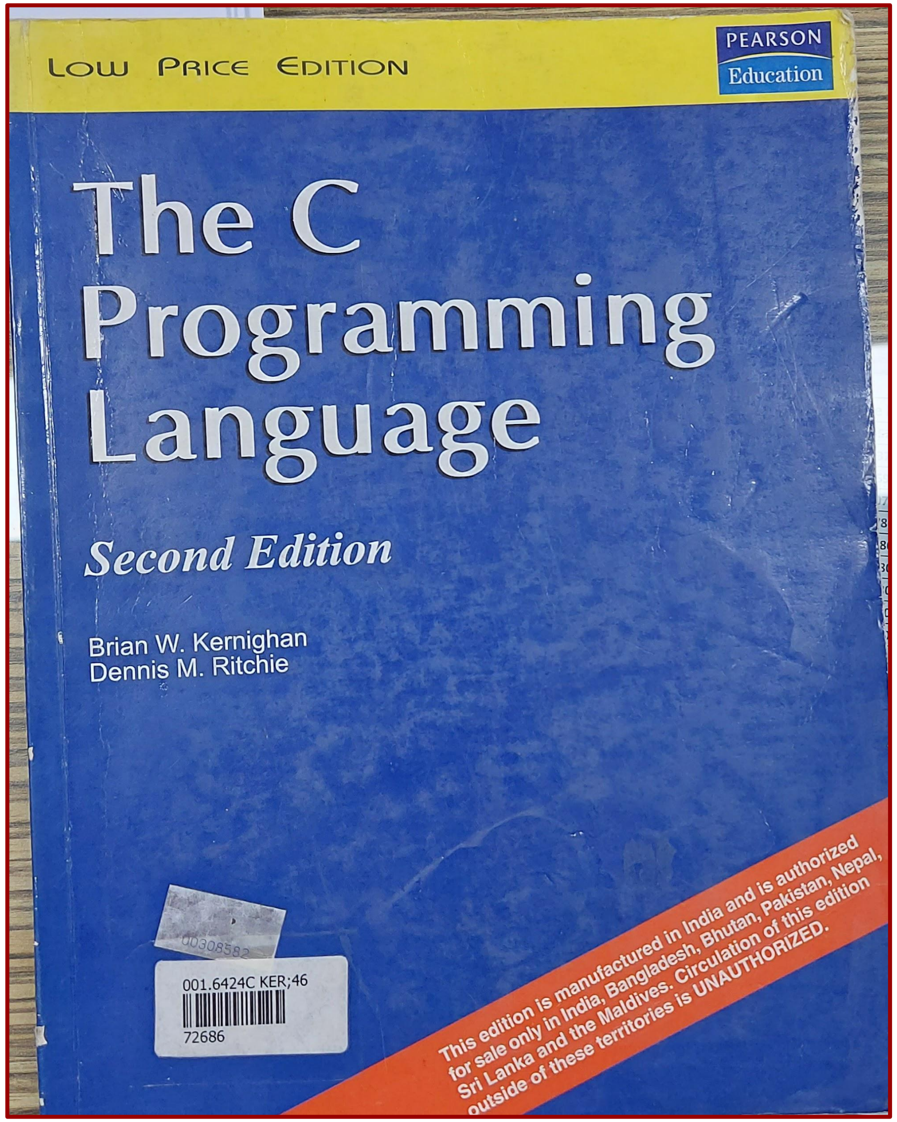
Remember	Understand	Apply	Analyze	Evaluate	Creating
Retrieve relevant knowledge from long term memory	Construct meaning from Source of information	Carryout or use a procedure in a given situation	Break apart material and determine relation	Make judgements based on criteria and standards	Produce original thoughts of elements
• Recognise • Recall	• Interpret • Exemplify • Classify • Summarize • Infer • Compare • Explain	• Execution • Implementation	• Differentiate • Analyse • Attribution	• Check • Critique	• Generate • Plan • Produce
02	07	02	03	02	03

UE24CS151B: Problem Solving with C - Course Outcomes

At the end of the course, the student will be able to

- CO1: Understand and apply algorithmic solutions to counting problems using appropriate C constructs
- CO2: Understand, analyse and apply Sorting and Searching techniques
- CO3: Understand, analyse and apply text processing and string manipulation methods using Arrays, Pointers and functions
- CO4: Understand user defined type creation and implement the same using C structures, unions and other ways by reading and storing the data in secondary systems which are portable.

UE24CS151B: Problem Solving with C Text Book



Low Price Edition

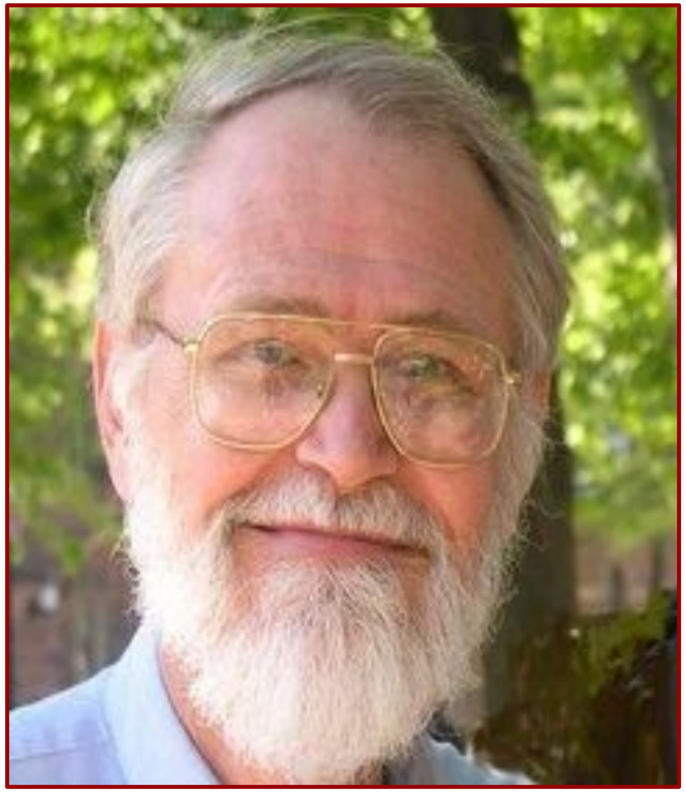
PEARSON Education

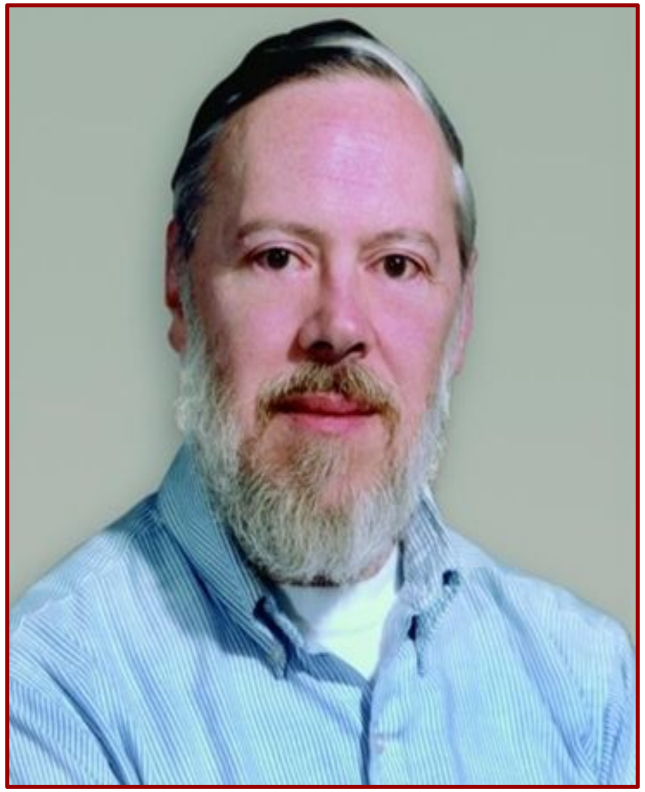
The C Programming Language

Second Edition

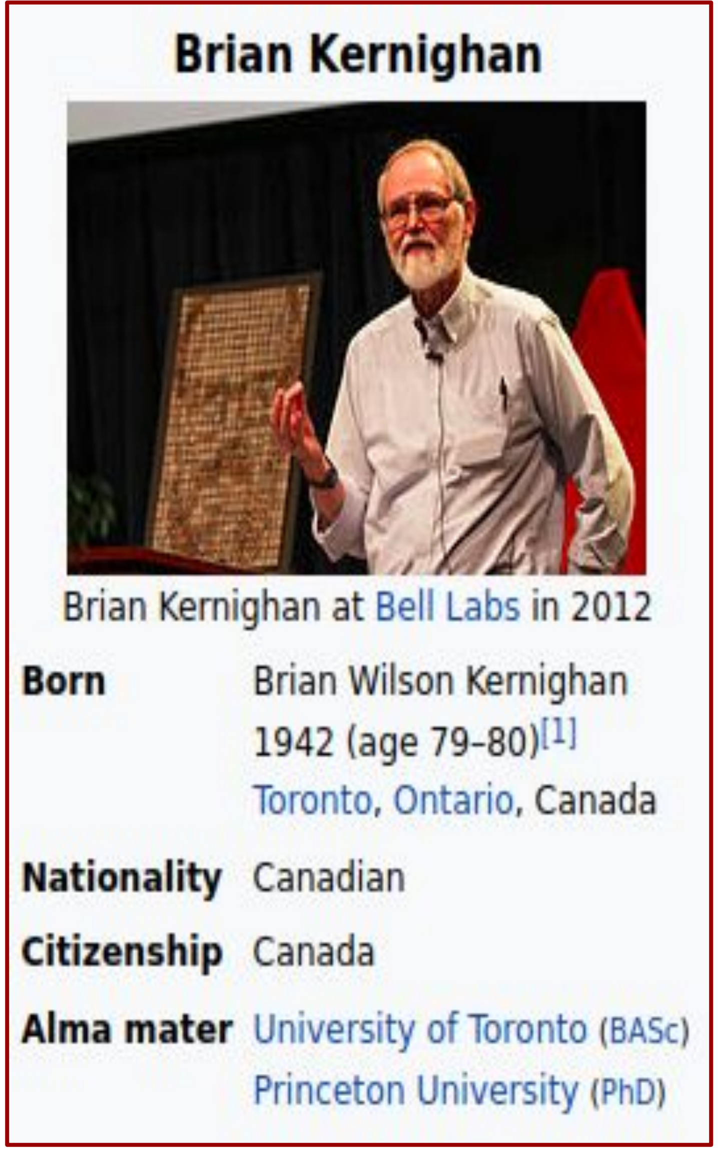
Brian W. Kernighan
Dennis M. Ritchie

This edition is manufactured in India and is authorized for sale only in India, Bangladesh, Bhutan, Pakistan, Nepal, Sri Lanka and the Maldives. Circulation of this edition outside of these territories is UNAUTHORIZED.





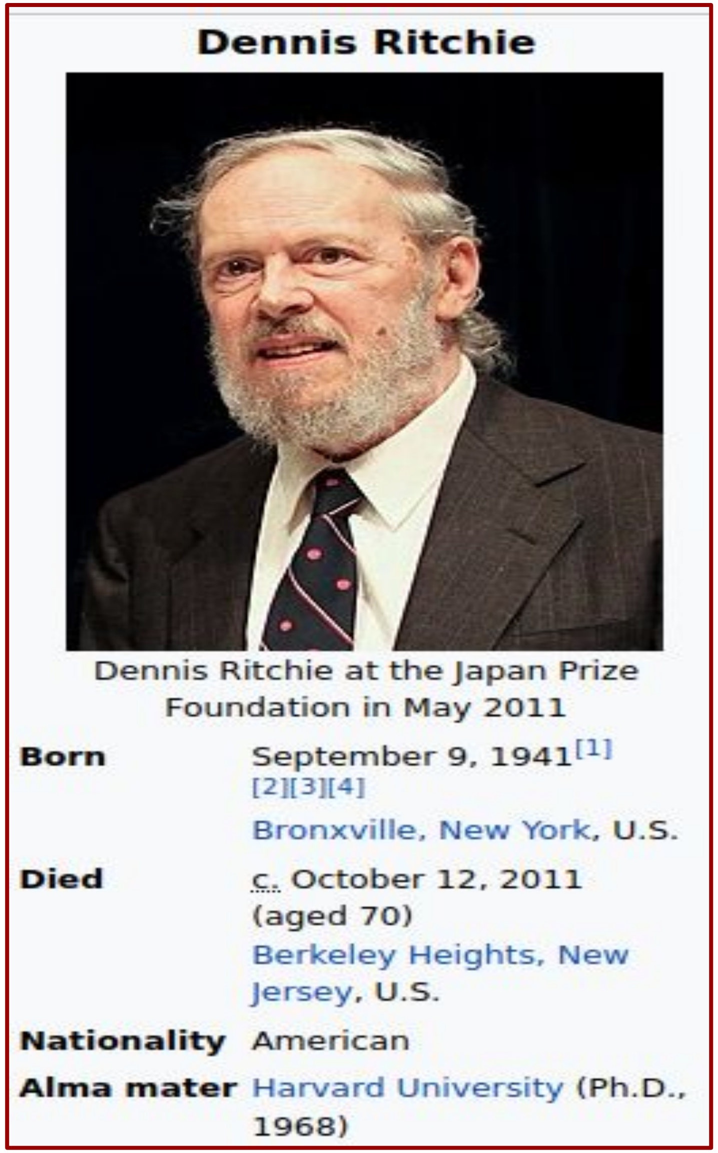
UE24CS151B: About Text Book Authors



Brian Kernighan

Brian Kernighan at Bell Labs in 2012

Born	Brian Wilson Kernighan 1942 (age 79–80)[1] Toronto, Ontario, Canada
Nationality	Canadian
Citizenship	Canada
Alma mater	University of Toronto (BASc) Princeton University (PhD)

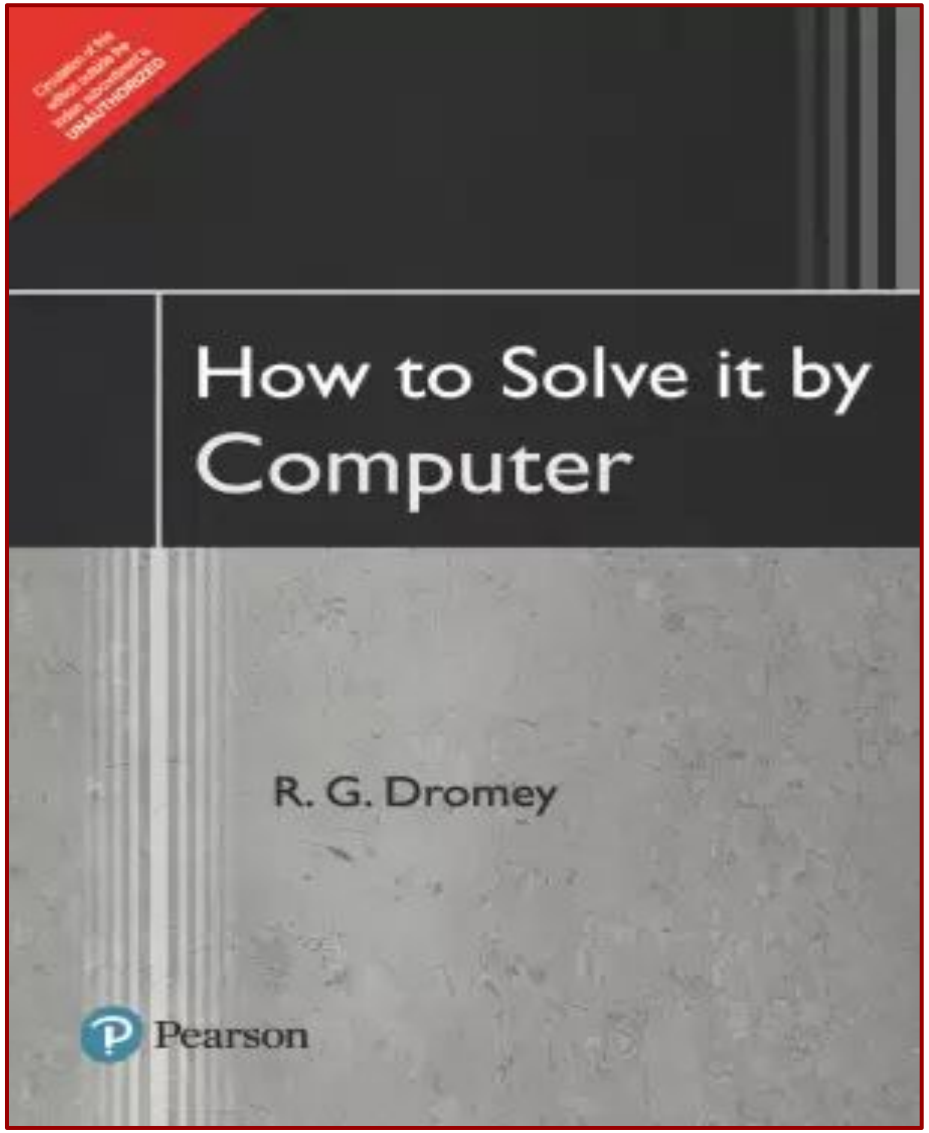


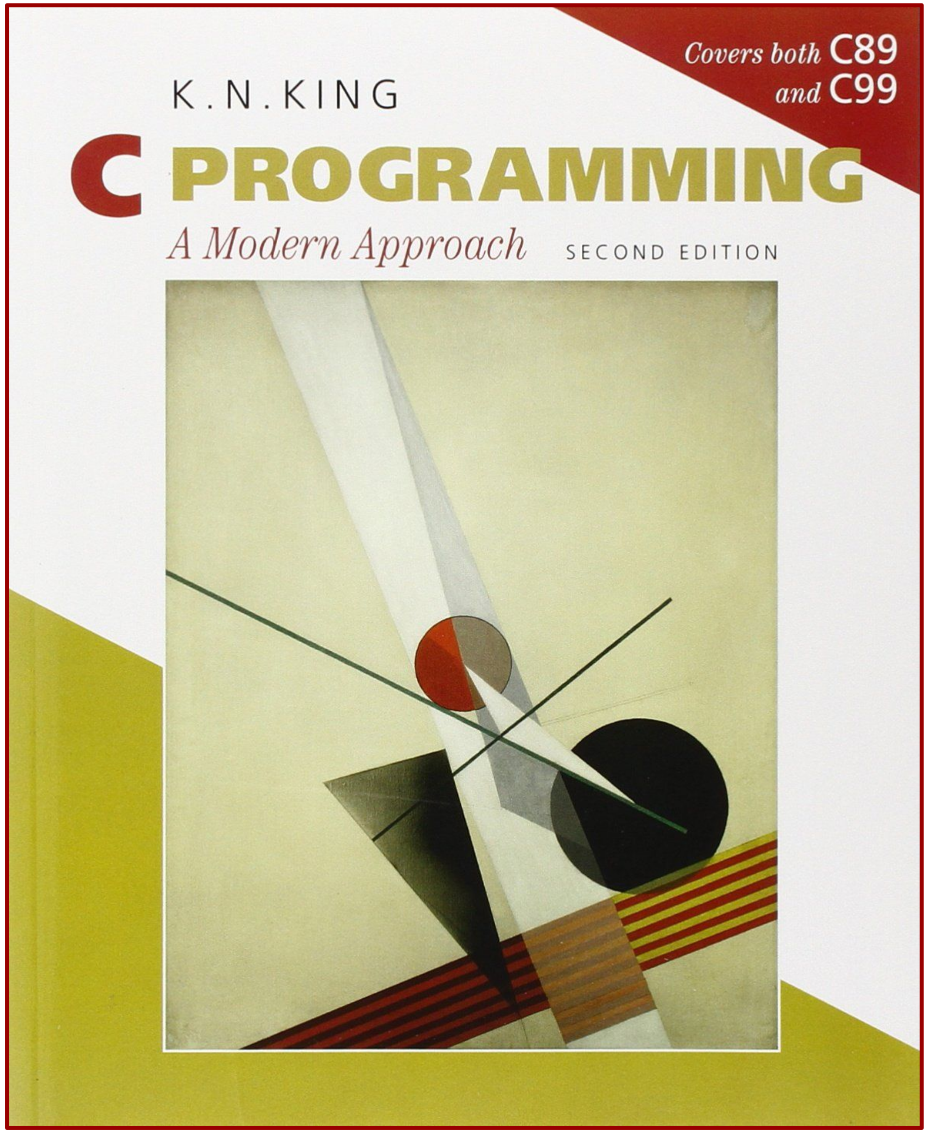
Dennis Ritchie

Dennis Ritchie at the Japan Prize Foundation in May 2011

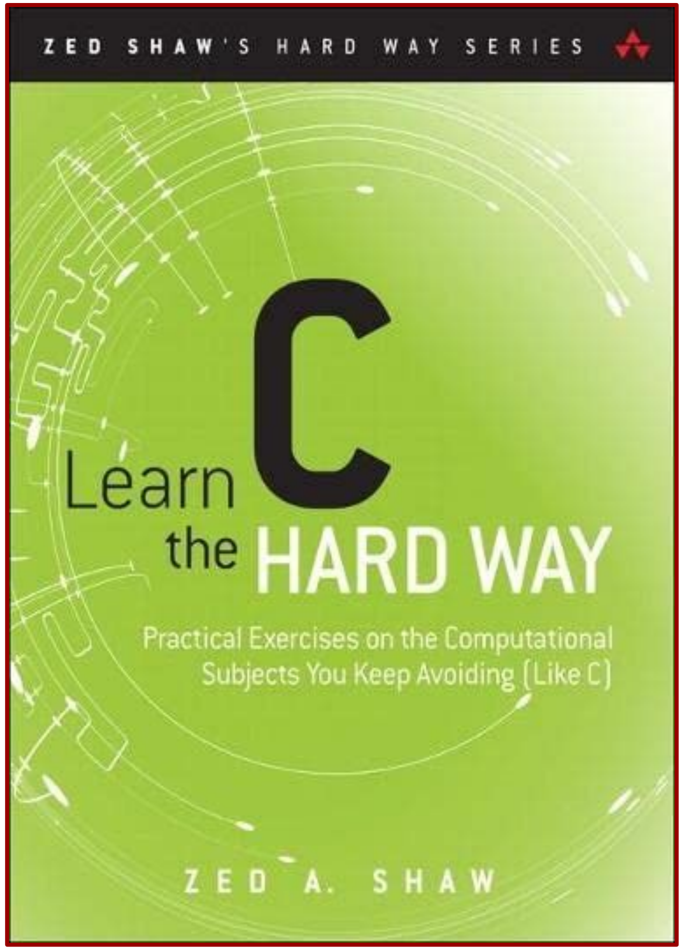
Born	September 9, 1941[1][2][3][4] Bronxville, New York, U.S.
Died	c. October 12, 2011 (aged 70) Berkeley Heights, New Jersey, U.S.
Nationality	American
Alma mater	Harvard University (Ph.D., 1968)

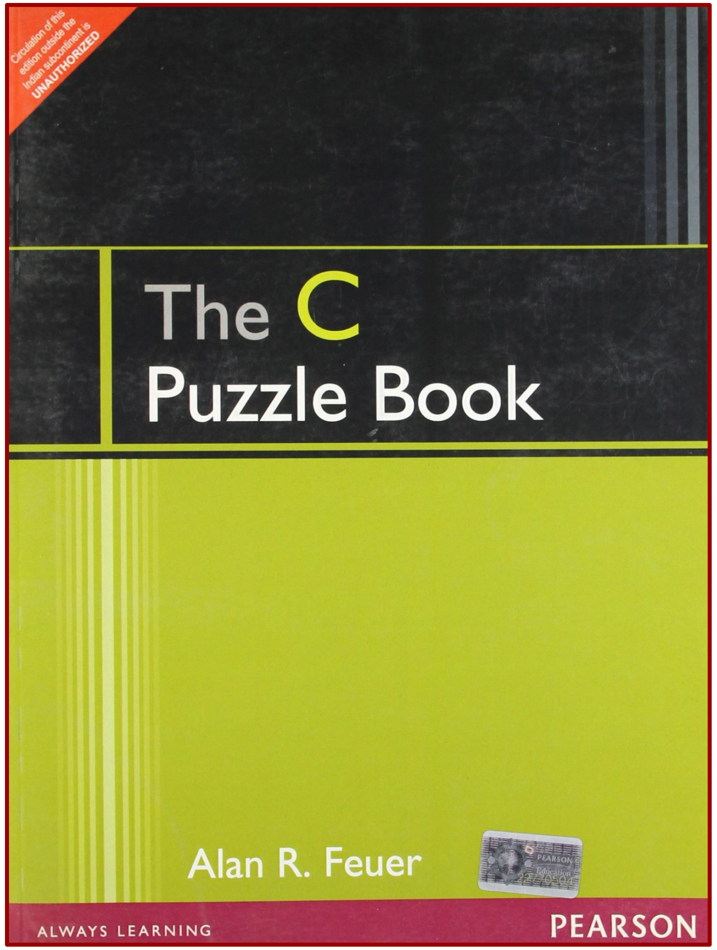
UE24CS151B: Problem Solving with C Reference Books

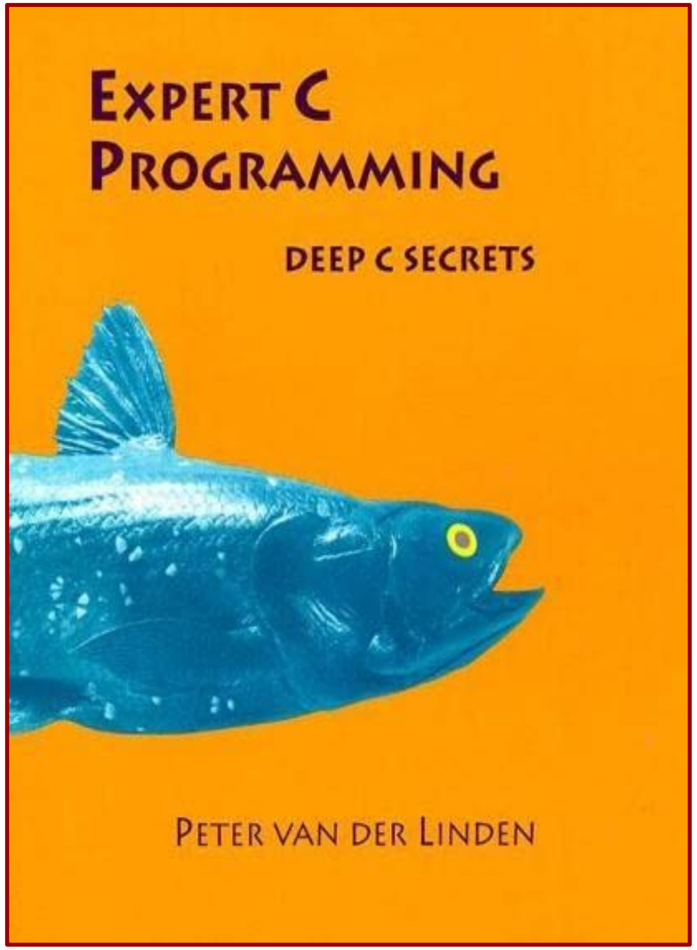




UE24CS151B: Problem Solving with C Reference Books


ZED SHAW'S HARD WAY SERIES
Learn C the HARD WAY
Practical Exercises on the Computational Subjects You Keep Avoiding (Like C)
ZED A. SHAW


Circulation of this edition outside the Indian subcontinent is UNAUTHORIZED
The C Puzzle Book
Alan R. Feuer
ALWAYS LEARNING
PEARSON



UE24CS151B : PSWC: Unit 1 - gcc Insights

- The original author of the GNU C Compiler (**gcc**) is **Richard Stallman**, the founder of the **GNU** Project
- **GNU** stands for **GNU's not Unix**
- The GNU Project was started in **1984** to create a complete Unix-like operating system as **free software**, in order to promote freedom and cooperation among computer users and programmers
- The **first** release of **gcc** was made in **1987,** as the first portable ANSI C optimizing compiler released as free software
- A major revision of the compiler came with the **2.0** series in **1992**, which added the ability to compile **C++**
- The acronym **gcc** is now used to refer to the “**GNU Compiler Collection**”
- **gcc** has been extended to support many additional languages, including Fortran, ADA, Java and Objective-C
- Its development is guided by the **gcc** Steering Committee, a group composed of representatives from **gcc** user communities in industry, research and academia

UE24CS151B : PSWC: Unit 1 - gcc Features

- **gcc** is a portable compiler, it runs on most platforms available today, and can produce **output** for many types of **processors**

- **gcc** also supports **microcontrollers**, **DSPs** and **64-bit** CPUs

- **gcc** is not only a native compiler, it can also cross-compile any program, producing executable files for a different system from the one used by **gcc** itself.

- **gcc** allows software to be compiled for embedded systems which are not capable of running a compile

- **gcc** written in C with a strong focus on portability, and can compile itself, so it can be adapted to new systems easily

- **gcc** has multiple language frontends, for parsing different languages

- **gcc** can compile or cross-compile programs in each language, for any architecture

- **gcc,** for example,can compile an ADA program for a **microcontroller**, or a C program for a **supercomputer**

UE24CS151B : PSWC: Unit 1 - gcc Features

- **gcc** has a modular design, allowing support for new languages and architectures to be added.

- **gcc** is free software, distributed under the GNU General Public License (GNU GPL), which means we have the freedom to use and to modify **gcc**, as with all GNU software.

- **gcc** users have the freedom to share any enhancements and also make use of enhancements to **gcc** developed by others.

- If we need support for a new type of **CPU**, a new language, or a new feature we can add it ourselves, or hire someone to enhance **gcc** for us, in addition we can hire someone to fix a bug if it is important for our work

UE24CS151B : PSWC: Unit 1 - gcc for c programming

- c is one those languages that allow **direct** access to the computer’s **memory**.
- Historically, **c** has been used for writing low-level systems software, and applications where **high performance** or control over resource usage are **critical**
- Great care is required to ensure that memory is accessed correctly, to avoid corrupting other data-structures whenever one uses **c** language for programming
- In addition to C , the GNU Project also provides other high-level languages, such as C++, GNU Common Lisp (gcl), GNU Smalltalk (gst), the GNU Scheme extension language (guile) and the GNU Compiler for Java (gcj).
- These languages with exception to **c** and **c++**, do not allow the user to access memory directly, **eliminating** the possibility of **memory access errors**.
- They are a safer alternative to c and c++ for many applications

UE24CS151B : PSWC: Unit 1 - Compiling a c program using gcc

- **c** programs can be compiled from a single source file or from multiple source files, and may use system libraries and header files

- Compilation refers to the process of converting a program from the textual source code, in a programming language such as **c** into machine code, the sequence of 1’s and 0’s used to control the central processing unit (CPU) of the computer.

- **Machine code** is then **stored** in a file known as an **executable** file, sometimes referred to as a **binary** file

UE24CS151B : PSWC: Unit 1 - Compiling a c program using gcc

-
  ```
  gcc -Wall PESU.c -o PESU
  ```

- This compiles the source code in **PESU.c** to machine code and stores it in an executable file **PESU**’.

- The output file for the machine code is specified using the **‘-o’** option.

- **-o** option is **usually** given as the **last** argument on the **command line**, If it is omitted, the output is written to a default file called **‘a.out’**.

- If a file with the same name as the executable file already exists in the current directory it will be overwritten

- The option **‘-Wall’ turns on** all the most commonly-used **compiler warnings**, it is **recommended** that we always **use** this **option**!

- **gcc** will not produce any warnings **unless** they are **enabled**.

- Compiler **warnings** are an essential aid in **detecting** problems when programming in **c**

- Source code which does not produce any warnings by **gcc**, is said to be **compile cleanly**.

UE24CS151B : PSWC: Unit 1 - Finding errors in a simple program using gcc

- Compiler **warnings** are an essential aid when programming in c

- **Error** is not obvious at first sight, but can be detected by the compiler if the warning option ‘-Wall’ has been enabled for **safety**

- The compiler distinguishes between **error** messages, which prevent successful compilation, and **warning** messages which indicate possible problems but **do not stop** the program from compiling

- It is very **dangerous** to develop a program **without checking** for compiler **warnings**.

- If there are any functions which are not used correctly they can cause the program to crash or produce incorrect results.

- Turning on the compiler warning option **‘-Wall’** for safety will catch many of the commonest errors which occur in c programming

UE24CS151B : PSWC: Unit 1 - Creating object files from source files using gcc

- The command-line option ‘**-c**’ is used to compile a source file to an **object** file.

- ‘-c’ produces an object file ‘**<filename>.o**’ containing the machine code for the main function.

- ‘**<filename>.o**’ contains a reference to the **external** function <filename>, but the corresponding memory address is left undefined in the object file at this stage, which will be filled in later by linking

- There is **no need** to use the option ‘-o’ to specify the name of the output file in this case.

- When compiling with ‘-c’ the compiler **automatically** creates an object file whose name is the same as the source file, but with **‘.o’** instead of the original extension

UE24CS151B : PSWC: Unit 1 - Compilation options - Setting search paths using gcc

- The list of directories for **header files** is often referred to as the include path, and the list of directories for libraries as the library search path or link path
- For example, a header file found in ‘/usr/local/include’ takes precedence over a file with the same name in ‘/usr/include’.
- Similarly, a library found in ‘/usr/local/lib’ takes precedence over a library with the same name in ‘/usr/lib’
- When additional libraries are installed in other directories it is necessary to extend the search paths, in order for the libraries to be found.
- The compiler options ‘-I’ and ‘-L’ add new directories to the beginning of the include path and library search path respectively

UE24CS151B : PSWC: Unit 1 - c Language Standards using gcc

- By default, **gcc** compiles programs using the **GNU** dialect of the C language, referred to as **GNU C**

- This dialect incorporates the official ANSI/ISO standard for the C language with several useful GNU extensions, such as nested functions and variable-size arrays.

- Most ANSI/ISO programs will compile under GNU C without changes

  - `gcc -Wall -ansi <filename.c>`
  - `gcc -Wall <filename.c>`

- The command-line option '-pedantic' in combination with '-ansi' will cause gcc to reject all GNU C extensions, not just those that are incompatible with the ANSI/ISO standard

  - `gcc -Wall -ansi -pedantic <filename.c>`

- The following options are a good choice for finding problems in C programs

  -
    ```
    gcc -ansi -pedantic -Wall -W -Wconversion -Wshadow -Wcast-qual
    -Wwrite-strings
    ```

Note: **While this list is not exhaustive, regular use of these options will catch many common errors.**

UE24CS151B : PSWC: Unit 1 - Errors

- **Error**
  - a mistake
  - the state or condition of being wrong in conduct or judgement
  - a measure of the estimated difference between the observed or calculated value of a quantity and its true value
- **Preprocessor Error**
  - #error is a preprocessor directive in c which is used to raise an error during compilation and terminate the process

- **Compile time Error**
  - When the programmer does not follow the syntax of any programming language, then the compiler will throw the **Syntax Error**, such errors are also called **Compile Time Error**
  - **Syntax Errors** are easy to figure out because the compiler highlights the line of code that caused the error.

UE24CS151B : PSWC: Unit 1 - Errors

- **Linker Error**
  - **Linker** is a program that takes the object files generated by the compiler and combines them into a single executable file.
  - **Linker Errors** are the errors encountered when the executable file of the code can not be generated even though the code gets compiled successfully.
  - This **Error** is generated when a different object file is unable to link with the main object file.
  - We can run into a linked error if we have imported an incorrect header file in the code

- **Runtime Error**
  - Errors that occur during the execution (or running) of a program are called **Runtime Errors.** These errors occur after the program has been compiled and linked successfully.
  - When a program is running, and it is not able to perform any particular operation, it means that we have encountered a runtime error.

UE24CS151B : PSWC: Unit 1 - Errors

- **Logical Error**
    - Sometimes, we do not get the output we expected after the compilation and execution of a program.
    - Even though the code seems error free, the output generated is different from the expected one.
    - These types of errors are called **Logical Errors**.
- **Semantic Errors**
    - Errors that occur because the compiler is unable to understand the written code are called Semantic Errors.
    - A semantic error will be generated if the code makes no sense to the compiler, even though it is syntactically correct.
    - It is like using the wrong word in the wrong place in the English language

UE24CS151B : PSWC: Unit 1 - Simple Input / Output Function

- Input and output functions are available in the **c language** to perform the most common tasks.

- In every **c program**, three basic functions take place namely **accepting of data as input**, **the processing of data**, and **the generation of output**

- When a **programmer** says **input**, it would mean that they are **feeding** some **data** in the program.

- Programmer can give this **input** from the **command line** or **in the form of any file**.

- The **c programming language** comes with a set of various **built-in functions** for **reading** the **input** and then **feeding** it to the available program as per our requirements.

- When a **programmer** says **output**, they mean **displaying** some **data** and **information** on the **printer**, the **screen**, or any other **file**.

- The **c programming language** comes with various **built-in functions** for **generating** the **output** of the **data** on any **screen** or **printer**, and also redirecting the output in the form of binary files or text file.

UE24CS151B : PSWC: Unit 1 - Unformatted I/O

- The **unformatted functions** are **not capable** of **controlling** the **format** that is involved in **writing** and **reading** the available **data**.

- Hence **these functions** constitute the most **basic** forms of **input** and **output**.

- The supply of input or the display of output **isn’t allowed** in the **user format**, hence we call these functions as **unformatted functions** for **input** and **output**.

- The unformatted input-output functions further have two categories:

    - The **character** functions

        - We use the character input functions for reading only a single character from the input device by default the keyboard
            - **getchar(), getche(),** and the **getch()** refer to the **input functions** of **unformatted type**

        - we use the character output functions for writing just a single character on the output source by default the screen
            - the **putchar()** and **putch()** refer to the output functions of unformatted type

UE24CS151B : PSWC: Unit 1 - Unformatted I/O

- The **unformatted functions** are **not capable** of **controlling** the **format** that is involved in **writing** and **reading** the available **data**.

- Hence **these functions** constitute the most **basic** forms of **input** and **output**.

- The supply of input or the display of output **isn't allowed** in the **user format**, hence we call these functions as **unformatted functions** for **input** and **output**.

- The unformatted input-output functions further have two categories:

  - The **string** functions

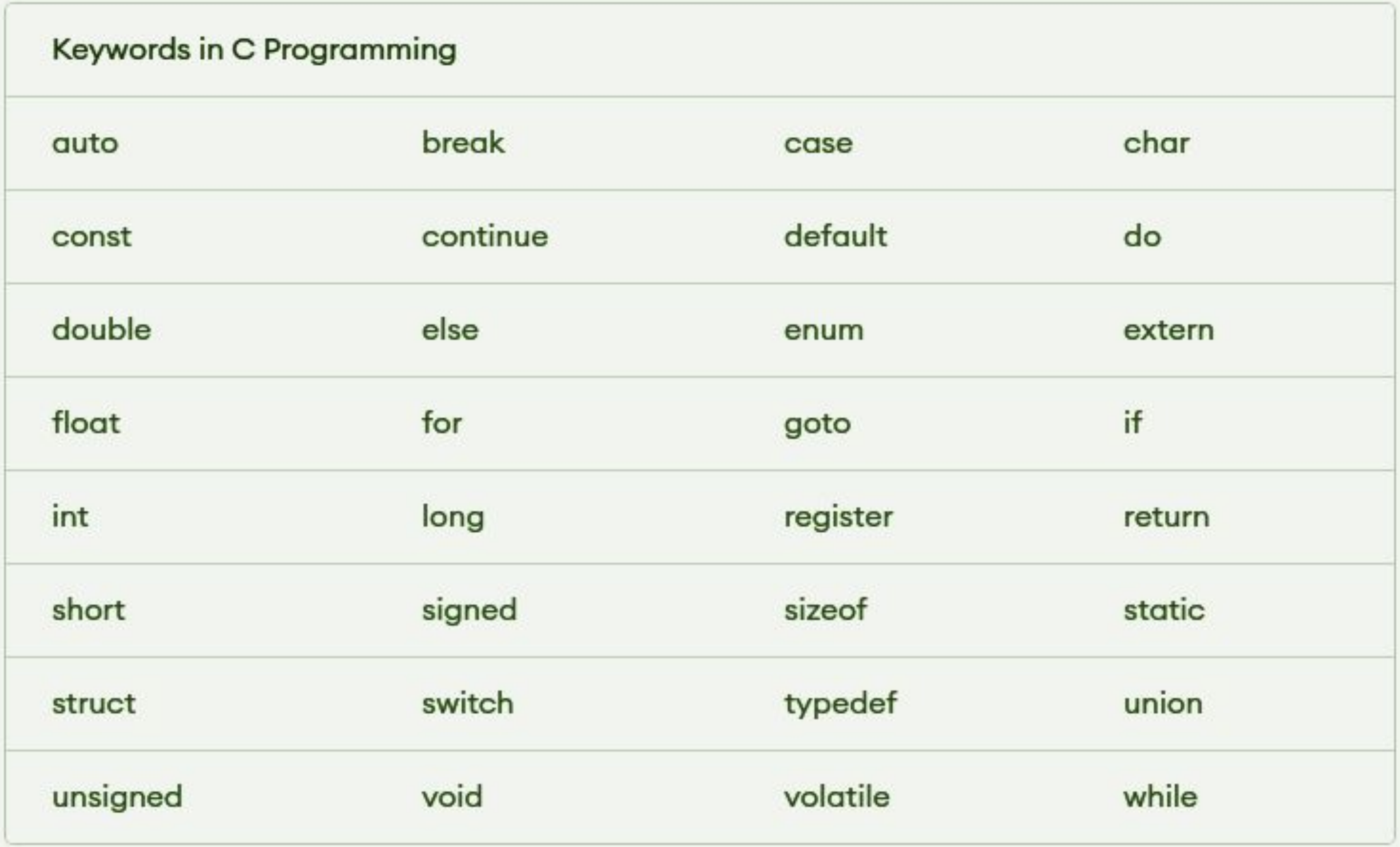
    - In any programming language including c, the **character array** or **string** refers to the **collection** of various **characters**

    - Various types of **input and output** functions are present in **c programming** that can easily read and write these strings.
      - The **puts()** and **gets()** are the most commonly used ones for **unformatted** forms
      - **gets()** refers to the **input** function used for **reading** the **string** characters
      - **puts()** refers to the **output** function used for **writing** the **string** characters

UE24CS151B : PSWC: Unit 1 - Formatted Output in C - printf

- **printf()**
  - This function is used to display one or multiple values in the output to the user at the console.
    - int printf(const char *format, ...)
    - Predefined function in stdio.h
    - Sends formatted output to stdout by default
    - Output is controlled by the first argument
    - Has the capability to evaluate an expression
    - On success, it returns the number of characters successfully written on the output.
    - On failure, a negative number is returned.
    - Arguments to printf can be expressions
  - While calling any of the formatted console input/output functions, we must use a specific format specifiers in them, which allow us to read or display any value of a specific primitive data type.
  - % [flags] [field_width] [.precision] conversion_character where components in brackets [] are optional.
  - The minimum requirement is %   and a conversion character (e.g. %d)
    - %d, %x, %o, %f, %c, %p, %lf, %s

UE24CS151B : PSWC: Unit 1 - keywords in c language



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

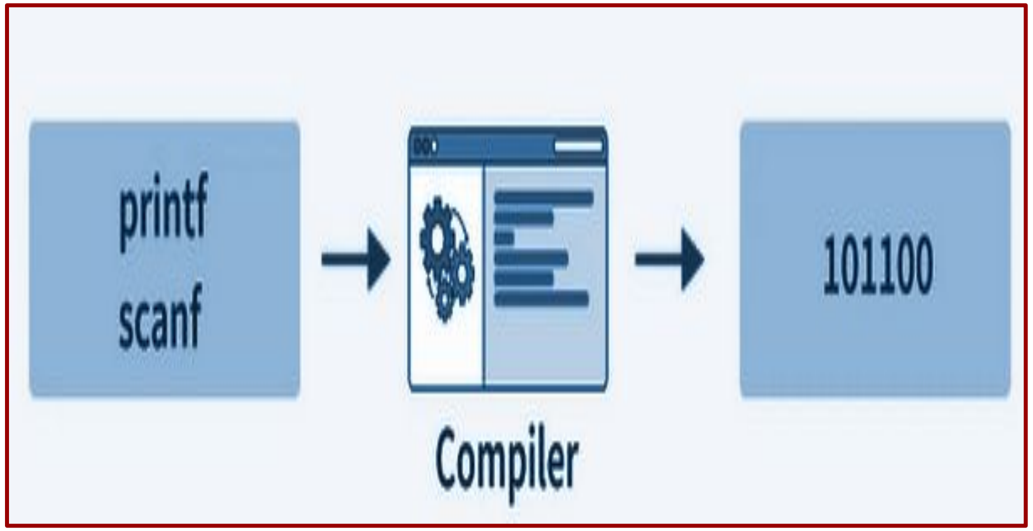
There are **32** keywords in c Language

There are **33** keywords in python

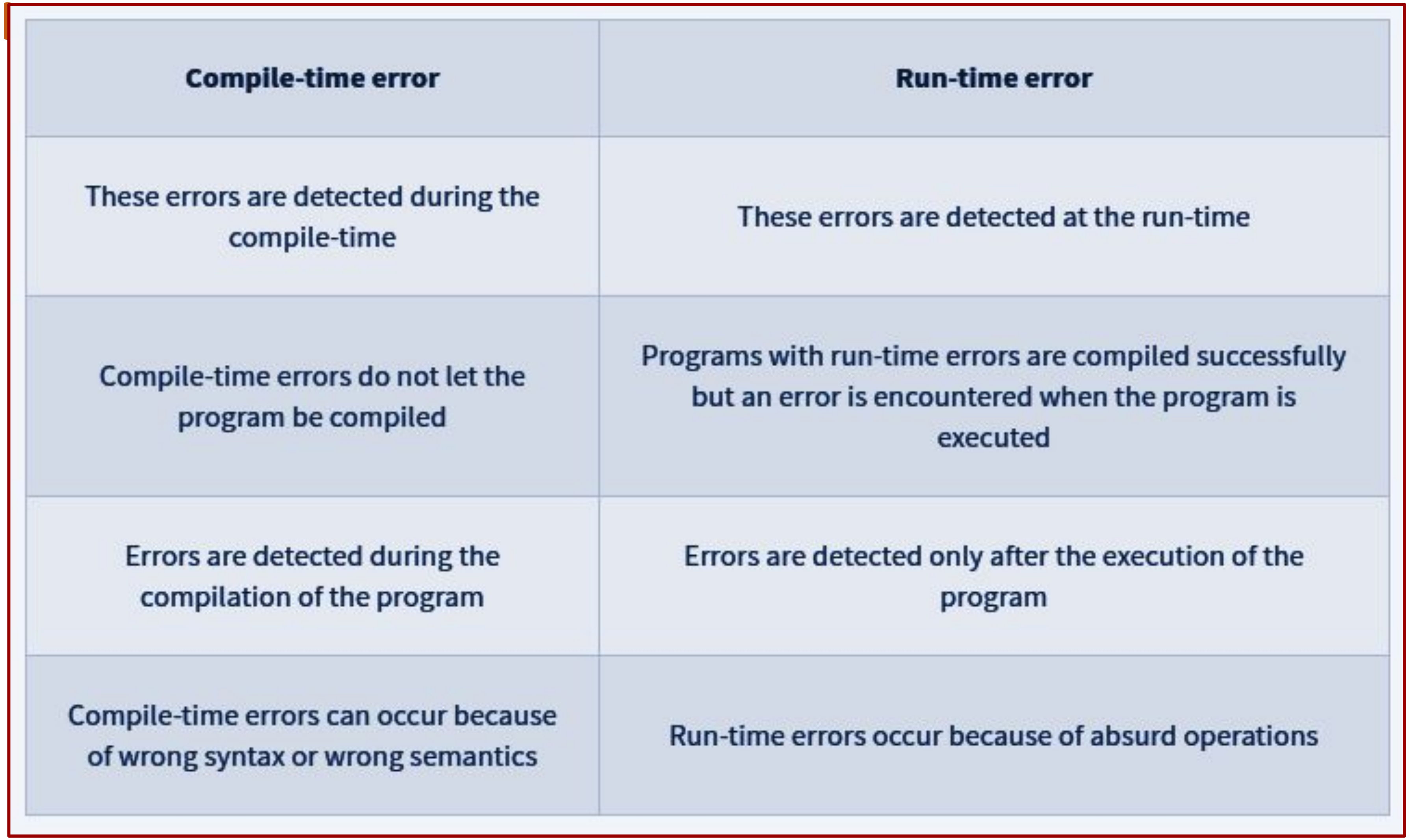
UE24CS151B : PSWC: Unit 1 - Compile time and Runtime in c Language

- **Compile time** is the period when the programming code is converted to the machine code.

- **Runtime** is the period of time when a program is running and generally occurs after compile time



UE24CS151B : PSWC: Unit 1 - Compile time and Runtime in c



Compile-time error	Run-time error
These errors are detected during the compile-time	These errors are detected at the run-time
Compile-time errors do not let the program be compiled	Programs with run-time errors are compiled successfully but an error is encountered when the program is executed
Errors are detected during the compilation of the program	Errors are detected only after the execution of the program
Compile-time errors can occur because of wrong syntax or wrong semantics	Run-time errors occur because of absurd operations

UE24CS151B : PSWC: Unit 1 - sizeof operator in c Language

- The **sizeof** operator is the most common operator in C.
- It is a **compile-time unary operator** and is used to compute the size of its operand.
- It returns the size of a variable.
- It can be applied to any data type, float type, pointer type variables
- When **sizeof()** is used with the data types, it simply returns the amount of memory allocated to that data type.
- The output can be different on **different machines** like a **32-bit system** can show different output while a **64-bit system** can show different of same data types

UE24CS151B : PSWC: Unit 1 - Literals and Constants in c Language

- **Literals** are the constant values assigned to the **constant** variables.

- There are **four** types of literals that exist in c programming:

    - **Integer literal**

        - It is a numeric literal that represents only integer type values.
        - It represents the value neither in fractional nor exponential part.
        - It can be specified in the following three ways
            - **Decimal number (base 10)**
                - It is defined by representing the digits between 0 to 9. Example: 1,3,65 etc

            - **Octal number (base 8)**
                - It is defined as a number in which 0 is followed by digits such as 0,1,2,3,4,5,6,7. Example: 032, 044, 065, etc

            - **Hexadecimal number (base 16)**
                - It is defined as a number in which 0x or 0X is followed by the hexadecimal digits (i.e., digits from 0 to 9, alphabetical characters from (a-f) or (A-F)) Example 0XFE oxfe

UE24CS151B : PSWC: Unit 1 - Literals and Constants in c Language

- **Literals** are the constant values assigned to the constant variables.
  - **Float literal**
    - It is a literal that contains only **floating-point** values or **real** numbers.
    - These real numbers contain the number of parts such as **integer part**, **real part**, **exponential part**, and **fractional part**.
    - The **floating-point literal** must be specified either in **decimal** or in **exponential** form.
      - **Decimal form**
        - The **decimal form** must contain either **decimal point**, **real part**, **or both**.
        - If it does not contain either of these, then the compiler will throw an error.
        - The decimal notation can be prefixed either by '+' or '-' symbol that specifies the positive and negative numbers.
        - Example: +9.5, -18.738

UE24CS151B : PSWC: Unit 1 - Literals and Constants in c Language

- **Literals** are the constant values assigned to the constant variables.
    - **Float literal**
        - The floating-point literal must be specified either in **decimal** or in **exponential** form.
            - **Exponential form**
                - The **exponential form** is useful when we want to represent the number, which is having a big magnitude.
                - It contains two parts, i.e., **mantissa** and **exponent**.
                - For example, the number is **3450000000000**, and it can be expressed as **3.45e12** in an exponential form

UE24CS151B : PSWC: Unit 1 - Literals and Constants in c Language

- **Literals** are the constant values assigned to the constant variables.

    - The floating-point literal must be specified either in decimal or in exponential form.

        - **Exponential form**
            - Syntax of float literal in **exponential form**
                - [+/-] <Mantissa> <e/E> [+/-] <Exponent>
            - Examples of real literal in exponential notation are
                - +3e24, -7e3, +3e-15
            - Rules for creating an **exponential notation**
                - In **exponential notation**, the **mantissa** can be specified either in **decimal** or **fractional** form
                - An **exponent** can be written in both **uppercase** and **lowercase**, i.e., **e** and **E**.
                - We can use both the signs, i.e., **positive** and **negative**, before the **mantissa** and **exponent**. Spaces are not allowed

UE24CS151B : PSWC: Unit 1 - Literals and Constants in c Language

- **Literals** are the constant values assigned to the constant variables.

    - **Character Literal**
        - A **character literal** contains a single character enclosed within single quotes.

        - If we try to store more than one character in a character literal, then the warning of a multi-character character constant will be generated.

        - Representation of **character literal**

            - A **character literal** can be represented in the following ways:
                - It can be represented by specifying a single character within single quotes. For example, 'x', 'y', etc.

                - We can specify the escape sequence character within single quotes to represent a character literal. For example, '\n', '\t', '\b'.

                - We can also use the **ASCII** in integer to represent a character literal. For example, the ascii value of 65 is 'A'.

                - The octal and hexadecimal notation can be used as an escape sequence to represent a character literal. For example, '\043', '\0x22'.

UE24CS151B : PSWC: Unit 1 - Literals and Constants in c Language

- **Literals** are the constant values assigned to the constant variables.

    - **String** Literal

        - A **string** literal represents multiple characters enclosed within double-quotes

        - It contains an additional character, i.e., '\0' (null character), which gets automatically inserted.

        - This **null** character specifies the **termination** of the **string**.

UE24CS151B
End of Slot #4, #5, #6

-

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University
nitin.pujari@pes.edu

For Course Digital Deliverables visit www.pesuacademy.com