Programming

and

Problem Solving using ‘C’

Lecture Notes

by

N S Kumar

kumaradhara@gmail.com

**Introduction**

We have already learnt programming and learnt programming in the language Python. In this course, we will learn programming and problem solving in ‘C’. In the beginning of each unit, we will state a project or a problem to be solved and we will develop the required constructs in ‘C and then at the end of the unit, we will solve the given problem and complete the given project.

1. Why should one learn C after having mastered Python? What are the advantages and disadvantages of Python over ‘C’? Is not ‘C’ an outdated language?

We have to fill our stomach every day 3 or 4 times so that our brain and body get enough energy to function. How about eating Vidyarthi Bhavan Dosa every day? What about Fridays when the eatery is closed? Why not buy Dosa batter from some nearby shop? Or do you prefer to make the batter yourself? Would you have time to do that? Would that depend on how deep your pockets are? Would you like to decrease your medical bills?

Every language has a philosophy. The language used by poets may not be suitable for conversation. Poets use ambiguity in meaning to their advantage, and some verses in Samskrita have more than one meaning. But that will not be suitable for writing a technical report.

Python supports flexibility. Time taken for development tends to be small in Python. But Python is not necessarily efficient. If the goal is efficiency, Python cannot be the language of choice. Python is itself built over the scaffolding provided by ‘C’.

The goal of ‘C’ is efficiency. The safety is in the hands of the programmer. ‘C’ does very little apart from what the programmer has asked for.

Example: When we index outside the bounds of a list in Python, we get an “index error” at runtime. To support this feature, Python runtime should know the current size of a list and should also check whether the index is valid each time we index on a list. You are all very good programmers and I am sure you never get an index error. But Python has to check even if you are right. So Python penalizes the right programmer. ‘C’ does not. You get what you deserve. If you are lucky, the program crashes. Otherwise something subtle may happen, which later may lead to catastrophic failures.

So,

* C gives importance to efficiency
* C is not very safe ; you can make your program safe
* C is not very strongly typed; mixing of types may not result in errors
* C is the language of choice for all hardware related softwares – system softwares
* C is the language of choice for softwares like operating system, compilers, linkers, loaders, device drivers.

2. Is ‘C’ not an old language?

Yes and No.

It was designed by Dennis Ritchie – my prostrations to him –most of us working in the field of computer science owe our life to him. We use ‘C’ like languages and unix like operating systems both have his contribution – in 70s. But the language has evolved over a period of time. The latest ‘C’ was revised in 2011.

A language should be understood the same way by compiler writers and programmers all over the world. To facilitate this, the languages are standardized by international organizations like the ISO.

The language ‘C’ has 4 historical land marks.

- K & R ‘C’ : defined by the seminal book on ‘C’ by Kernighan and Ritchie

- C 89

- C 99

- C 11

There is one more reason to learn ‘C’. ‘C’ is the second most popular language as of now. according to Tiobe ratings. <https://www.tiobe.com/tiobe-index/>.

This website gives the popularity ratings of programming languages. The rating of ‘C’ is more than 11% and Java is 14% as on date 8th Feb 2018.

Program Structure:

Let us summarize the characteristics of a program in ‘C’

* The language is case sensitive like Python. The uppercase and lowercase are distinguished.
* The program is immune to indentation. In Python, we introduce indentation to indicate the structure of the program – be it control structure, class or function. ‘C’ follow the free format source code concept. The way of presentation of the program to the compiler has no effect on the logic of the program.

However, we should properly indent the program so that the program becomes readable for humans. When you prepare a new dish, I am sure you will also consider how you present it to your kith and kin.

* The program has a well defined entry point. Every program has to have an entry point called main, which is the starting point of execution.
* We add comments for documentation. There are two types of comments.
  + Line comment : starts with // and ends at the end of the line
  + Block comment : start with *\* and ends with \** and can span multiple lines

Comments are ignored by the compiler.

* A program in ‘C’ normally has one or more pre-processor directives. These start with the symbol #. One of the directives is “include”.

#include <stdio.h>

The above line asks the translator to find the file whose name is stdio.h at some location in our file system and read and expand it at that point in the ‘C’ code.

We will discuss later about .h or header file.

* A program in ‘C’ has to have a function called main. The function returns an int. The function may take a couple of arguments – will discuss this later. A function definition has a return type, then the function name and declaration of parameters with parentheses and a block(equivalent to suite in Python).

The main function is invoked(called) from some external agency – like our shell. It is the duty of this function to tell the caller whether it succeeded or failed. So, by convention, main returns 0 on success and non-zero on failure.

* printf is a library function to display. The first argument is always a string. In ‘C’, string constants or literals are always enclosed in double quotes.
* A block contains a number of statements. Each statement ends in a semicolon. This symbol ; plays the role of statement terminator.

// name: 1\_intro.c

// first program

#include <stdio.h>

int main()

{

printf(“hello world\n”);

return 0;

}

**Steps involved in executing a program in ‘C’**

* We enter the program into the computer. This is called **edit**ing. We save the **source program**.
* Then the source program is pre-processed. All commands starting from the symbol # are processed. The output is called a **translation unit or translation.**
* Then the translation is **compiled**. The output is called an **object** **file**. There may be more than one translation. So, we may get multiple object files.
* We put together all these object files along with the predefined library routines by **linking**. The output is called an **image or a loadable image**.
* Then we **load** the image into the memory of the computer. This creates a **process**.
* We **execute or run** the process. We get some results. In ‘C’, we get what we deserve!

The commands we use are as follows.

a) just preprocess:

gcc -E 1\_intro.c

b) compile

gcc -c 1\_intro.c

c) link

gcc 1\_intro.o

d) load and run a loadable image

./a.out