Department of Computer Science and Engineering,
PES University, Bangalore, India

Lecture Notes
Problem Solving With C
UE24CS151B

Lecture #3
GCC, PDLC

By,
Prof. Sindhu R Pai,
Theory Anchor, Feb-May, 2025
Assistant Professor
Dept. of CSE, PESU

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Center, PES University)
Prof. Nitin V Poojari (Dean, Internal Quality Assurance Cell, PES University)

Unit #: 1

Unit Name: Problem Solving Fundamentals

Topic: GCC, PDLC

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

- Acquire knowledge on how to solve relevant and logical problems using computing Machine.
- Map algorithmic solutions to relevant features of C programming language constructs.
- Gain knowledge about C constructs and its associated ecosystem.
- Appreciate and gain knowledge about the issues with C Standards and it's respective behaviours.

Course outcomes: At the end of the course, the student will be able to:

- Understand and Apply algorithmic solutions to counting problems using appropriate C Constructs.
- Understand, Analyze and Apply sorting and Searching techniques.
- Understand, Analyze and Apply text processing and string manipulation methods using Arrays, Pointers and functions.
- 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.

Sindhu R Pai

Theory Anchor, Feb - May, 2025

Dept. of CSE,

PES University

Introduction to GCC

The **original author of the GNU C Compiler 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

Features of GCC:

- A portable compiler, it runs on most platforms available today, and can produce **output** for many types of **processors**
- Supports **microcontrollers**, **DSPs** and **64-bit** CPUs
- It 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.
- Allows software to be compiled for embedded systems
- **Written in C with a strong focus on portability**, and can compile itself, so it can be adapted to new systems easily
- It has multiple language frontends, for parsing different languages
- It can compile or cross-compile programs in each language, for any architecture Example: Can compile an ADA program for a **microcontroller** or a C program for a **supercomputer**
- It has a **modular design**, allowing support for new languages and architectures to be added.
- It 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.

Installation of gcc on different Operating systems:

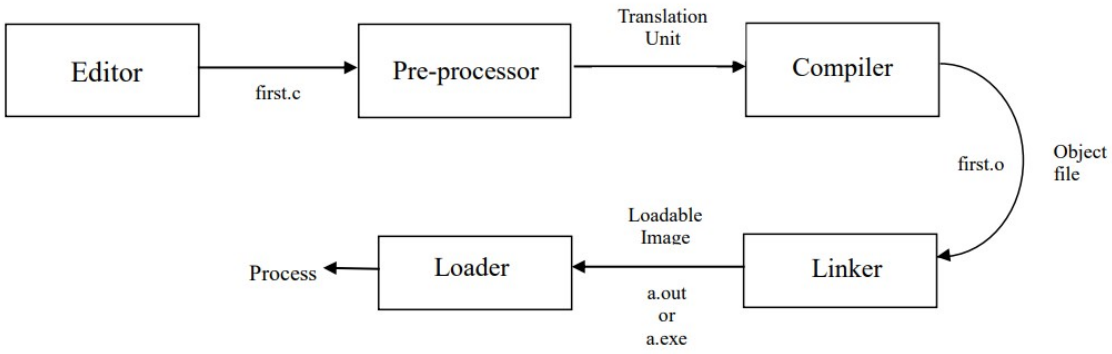
- **Windows OS – Installation of gcc using Mingw.**

  https://www.youtube.com/watch?v=sXW2VLrQ3Bs

- **Linux /MAC OS – gcc available by default**

Program Development Life Cycle [PDLC]

Phases involved in PDLC are **Editing, Pre-processing, Compilation, Linking, Loading and execution.**



The stages for a C program to become an executable are the following:

1. Editing:

We enter the program into the computer. This is called editing. We save the **sourceprogram**. Let us create first.c

$ gedit first.c // press

enter key#include

<stdio.h>

```
int main()
{       // This is a comment
        //Helps to understand the code Used for readability
        printf("Hello Friends");
        return 0;
}// Save this code in first.c
```

2. Pre-Processing:

In this stage, the following tasks are done:

a. Macro substitution–To be discussed in detail in Unit-5

b. Comments are stripped off

c. Expansion of the included files

The output is called a **translation unit or translation.**

To understand Pre-processing better, compile the above 'first.c' program using flag -E, which willprint the pre-processed output to stdout.

$ gcc -E first.c// No file is created. Output of pre–processing on the standard output-terminal

```
.....
# 846 "/usr/include/stdio.h" 3 4
# 886 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute_____((_
nothrow_)); extern int ftrylockfile (FILE *__stream) __attribute
_____((_
nothrow_)) ; extern void funlockfile (FILE *__stream) __
attribute_____((_
nothrow_));
.....
# 916 "/usr/include/stdio.h" 3 4
```

```
# 2

"first.c"

int

main()

{

printf("HelloFriends");

return0;

}
```

In the above output, you can see that the source file is now filled with lots of information, but still at the end of it we can see the lines of code written by us. Some of the observations are as below.

- Comment that we wrote in our original code is not there. This proves that all the comments are stripped off.
- The '#include' is missing and instead of that we see whole lot of code in its place. So it is safe to conclude that stdio.h has been expanded and literally included in our source file.

3. Compiling

Compilation refers to the process of converting a program from the textual source code 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. C programs can be compiled from a single source file or from multiple source files, and may use system libraries and header files. Compiler processes statements written in a particular programming language and converts them into machine language or "code" that a computer's processor uses. 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. 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

gcc -c first.c

This command does pre-processing and compiling. Output of this command is first.o -> **Object file.**

4. Linking

It is a computer program that takes one or more object files generated by a compiler and combine them into one, executable program. Computer programs are usually made up of multiple modules that span separate object files, each being a compiled computer program. This is the stage at which all the linking of function calls with their definitions are done. Till stage, gcc doesn't know about the definition of functions like printf(). Until the compiler knows exactly where all of these functions are implemented, it simply uses a place-holder for the function call. The definition of printf() is resolved and the actual address of the function printf() is plugged in. We put together all these object files along with the predefined library routines by **linking**. The output is called as **image or a loadable image** with the name a.out [linux based] or a.exe[windows]

gcc first.o // Linking to itself

// If more than one object files are created after compilation, all must be mentioned with gcc and linked here to get one executable.

If a file with the same name[a.out/a,exe] as the executable file already exists in the current directory it will be overwritten. This can be avoided by using the gcc option -o. This is usually given as the last argument on the command line.

gcc first.o -o PESU // loadable image - Output file name is PESU

5. Executing the loadable image

Loader loads the image into the memory of the computer. This creates a **process** when command is issued to execute the code. We **execute or run** the process. We get some results. In 'C', we get what we deserve!

In windows,

a.exe and press enter

OR

PESU and press enter

In Linux based systems,

./aout and press enter

OR

./PESU and press enter

For interested students:

The following options of gcc are a good choice for finding problems in C and C++ programs.

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

Happy Coding!