Topics To be Covered

Week	Broader Topic	Lecture	Topics	Tools to be covered
1	GCC Compiler	81-90	1. GCC Compiler 1.2 Introduction 2. Compilation of program, Execution of Program 3. Installing the build essential packages 4. Compiling a C program 5. Basic Shell Programs 5.1 Write shell script to print your full name 5.2 Linux shell script program to create and print the value of variables 5.3 Linux shell script program to add two numbers 5.4 Linux shell script program to swap two numbers 6. Time Stamping	Linux

1. Introduction to GCC compiler: Basics of GCC

GCC (GNU Compiler Collection) is a popular and widely used compiler system that supports multiple programming languages, including C, C++, Objective-C, Fortran, Ada, and others. It is a free and open-source compiler suite developed by the GNU Project.
Here are some key points and basics of the GCC compiler:

- **Compilation Process:** The GCC compiler translates source code written in a programming language into executable machine code that a computer can understand and execute. The compilation process involves several stages, including preprocessing, assembly, and linking.

- **Language Support:** GCC supports multiple programming languages, with C and C++ being the most commonly used. It provides comprehensive language features, optimizations, and compatibility with various language standards, such as ISO C, ISO C++, and POSIX.

- **Preprocessing:** Before the actual compilation, the preprocessor stage processes the source code files. It handles directives starting with a hash symbol (#), such as include statements, macro definitions, and conditional compilation. The preprocessor expands macros and includes the necessary header files.

- **Compilation:** In the compilation stage, GCC translates the preprocessed source code into assembly code specific to the target architecture. It performs language-specific analyses, optimizations, and generates intermediate representations, such as GIMPLE or GENERIC.

- **Assembly:** The assembly stage converts the generated assembly code into machine code specific to the target platform. It handles low-level instructions, registers, and memory operations. The resulting object code is represented in relocatable format.

- **Linking:** In the final stage, the linker combines the object code files generated by the compiler along with any required libraries to create an executable binary. It resolves symbols and references, performs address relocation, and produces a standalone executable file.

- **Optimization:** GCC provides a wide range of optimization options to improve the performance and efficiency of the generated code. It includes optimizations for code size, speed, and memory usage. Optimization levels can be specified during compilation to balance between compilation time and the resulting code's performance.

- **Platform Support:** GCC is known for its portability and supports a variety of platforms, including different processor architectures, operating systems, and embedded systems. It can generate code for x86, ARM, PowerPC, MIPS, and other architectures.

- **Extensions and Features:** GCC offers various language extensions and features beyond standard specifications. These extensions allow programmers to leverage additional functionality provided by GCC, but they might reduce the portability of the code to other compilers.

- **Toolchain:** GCC is a part of a larger toolchain that includes additional tools like GNU Binutils (assembler, linker), GNU Debugger (GDB), and runtime libraries. Together, these tools form a comprehensive development environment for building and debugging software.

GCC is widely adopted in the open-source community and is the default compiler on many Unix-like operating systems, including Linux. It provides a robust and efficient compilation system for a wide range of programming languages, making it a popular choice for software development projects.

2. Compilation of program, Execution of program

When working with a program, there are two main steps involved: compilation and execution.

- **Compilation:** Compilation is the process of translating the human-readable source code written in a programming language into machine-readable instructions that can be executed by the computer. The compilation process typically involves the following steps:

- **Preprocessing:** In this step, the preprocessor (part of the compiler) handles directives such as #include, #define, and #ifdef. It expands macros and includes necessary header files, preparing the source code for the next stage.

- **Parsing:** The compiler parses the preprocessed source code, analyzing its structure and syntax according to the rules of the programming language. This step creates an intermediate representation of the code.

- **Semantic Analysis:** The compiler performs semantic analysis, which involves checking the correctness of the code in terms of types, variable declarations, function calls, and other language-specific rules. It ensures that the code follows the language's grammar and semantics.

- **Code Generation:** Based on the intermediate representation, the compiler generates machine code specific to the target platform. This code consists of low-level instructions understood by the computer's hardware.

The output of the compilation process is often an object file or executable file. The object file contains compiled machine code but may not be directly executable. It needs to be linked with other object files and libraries to create a complete executable program.

- **Execution:** Once the compilation process is complete and an executable program is generated, the program can be executed, which means running the program on the computer. The execution process involves the following steps:

- **Loading:** The operating system loads the executable program into memory, allocating the necessary resources to execute it.

- **Startup:** The program's entry point is typically a function called "main." The execution starts at this point, and any initialization code specified in the program is executed.

- **Runtime:** The program performs its intended operations, which can involve reading input, processing data, performing calculations, interacting with external resources, and producing output.

- **Termination:** The program reaches its end or encounters an exit condition. At this point, it releases any resources it acquired during runtime and terminates its execution.

- **Output:** The program may produce output, such as displaying information on the screen, writing to files, or communicating with other programs or devices.

During program execution, the operating system manages various aspects, including memory allocation, process scheduling, input/output operations, and handling system calls. It provides an execution environment for the program to run efficiently and securely.
By following the steps of compilation and execution, programmers can develop and run software applications on computers, enabling them to perform various tasks and solve specific problems.

3. Installing the Build-Essential Package

To install the build-essential package on a Debian-based Linux distribution (such as Ubuntu), you can follow these steps:

- **Open a terminal:** Launch the terminal application on your Linux system. You can usually find it in the Applications menu or by searching for "Terminal" in the system's launcher.
- **Update package lists (optional):** It's a good practice to update the package lists to ensure you have the latest information about available packages. Run the following command in the terminal:

```
sudo apt update
```

Enter your password when prompted. This command updates the local package lists from the repositories.

- I**nstall the build-essential package:** Use the following command to install the build-essential package:

```
sudo apt install build-essential
```

This command installs the build-essential package and any additional dependencies required for building software, such as the GNU C Compiler (gcc), GNU C++ Compiler (g++), make utility, and others.

- **Authenticate and confirm:** The package manager may ask for your password to authenticate the installation. Enter your password and press Enter to proceed. Additionally, if the installation requires a significant amount of disk space, the package manager may prompt you to confirm the installation. Type "Y" and press Enter to continue.
- **Wait for the installation to complete:** The package manager will download and install the build-essential package along with its dependencies. The process may take some time depending on your internet connection speed and system performance. Once the installation is finished, you'll see a message indicating the successful installation.

After completing these steps, the build-essential package will be installed on your system. You can now use it for building software, compiling programs, and performing other development tasks that require essential build tools and libraries.

4. Compiling a C program

To compile a simple C program, you can follow these steps:

1. **Create a C source file:**
2. Use a text editor to create a new file and save it with a .c extension, for example, hello.c.
3. Open the file and enter the C code for your program. As an example, let's create a simple "Hello, World!" program:

```
include <stdio.h>

 int main()

{

printf("Hello, World!\n");

return 0;

}
```

4. **Open a terminal:**
5. Open a terminal on your Linux system. This can usually be done through the Applications menu or by searching for "Terminal" in the system's launcher.

6. **Navigate to the directory:**
7. Use the cd command to navigate to the directory where your C source file is located. For example, if the file is located in the home directory, use:

```
cd ~
```

8. **Compile the program:**
9. To compile the C program, use the gcc compiler (GNU Compiler Collection) followed by the source file name:

gcc hello.c -o hello

The -o option specifies the name of the output executable file. In this example, it's named hello. If you don't specify an output file name, the default output file name will be a.out.

10. **Execute the program:**

After successful compilation, you can run the program by executing the generated executable file:

./hello

This command executes the hello program, and you should see the output "Hello, World!" displayed in the terminal.

5. Write shell script to print your full name

```
#!/bin/bash
echo "My full name is Linux."
```

1. Save the above code in a file with a **.sh** extension, such as **first_prog.sh.** Make the script executable by running the following command in the terminal:

chmod +x first_prog.sh

2. Then, you can run the script by executing the following command:

./print_name.sh

3. The output will be:
   My full name is Linux.

5.2 Linux shell script program to create and print the value of variables

```
#!/bin/bash

# Variable declaration and assignment
name="Linux"
age=25
city="Chandigarh"

# Printing variable values
echo "Name: $name"
echo "Age: $age"
echo "City: $city"
```

1. Save the above code in a file with a **.sh** extension, such as **print_variables.sh**. Make the script executable by running the following command in the terminal:

   chmod +x print_variables.sh

2. Run the script

In this script, we declare and assign values to three variables: **name, age,** and **city**. Then, we use the **echo** command to print the values of these variables.

5.3 Linux shell script program to add two numbers

```
#!/bin/bash

# Variable declaration and assignment
num1=10
num2=20

# Addition
sum=$((num1 + num2))

# Printing the result
echo "The sum of $num1 and $num2 is: $sum"
```

5.4 Linux shell script program to swap two numbers

```
#!/bin/bash

# Variable declaration and assignment
num1=10
num2=20

echo "Before swapping:"
echo "Num1: $num1"
echo "Num2: $num2"

# Swapping the values
temp=$num1
num1=$num2
num2=$temp

echo "After swapping:"
echo "Num1: $num1"
echo "Num2: $num2"
```

6. Time Stamping

Timestamping refers to assigning a timestamp or a time value to an event, data, or a specific point in time. In computing, timestamping is commonly used for various purposes, such as tracking file modifications, logging events, measuring performance, and synchronization. There are different

ways to obtain and work with timestamps, depending on the specific context and requirements. Here are a few common approaches:

1. **System Time:**

Most operating systems provide a system clock that keeps track of the current time. You can obtain the system time in various programming languages or command-line utilities. The system time is typically represented as the number of seconds or milliseconds elapsed since a specific reference point (e.g., January 1, 1970, 00:00:00 UTC), known as the Unix epoch.

2. **Functions and Libraries:**

Many programming languages provide built-in functions or libraries to work with timestamps. These functions often allow you to obtain the current time, format timestamps, perform calculations, convert between different time zones, and handle date and time values. For example, in C, you can use functions like time(), localtime(), and strftime() from the <time.h> header.

3. **Timestamp Formats:**

Timestamps can be represented in various formats, such as ISO 8601, Unix timestamp, or custom formats. ISO 8601 (e.g., "2023-06-26T15:30:00Z") is a widely used format that represents the date and time in a standardized way. Unix timestamps represent time as a single number representing the number of seconds since the Unix epoch.

4. **File Timestamps:**

File systems often maintain timestamps for files, including the last modification time, last access time, and creation time. These timestamps can be accessed and manipulated using file system APIs or command-line utilities like ls or stat. File timestamps are useful for tracking changes, managing file backups, and maintaining file metadata.

5. **Network Time Protocol (NTP):**

In distributed systems or scenarios requiring accurate and synchronized time across multiple machines, the Network Time Protocol (NTP) can be used. NTP allows systems to synchronize their clocks with highly accurate time sources over the network, ensuring consistency and accuracy in timestamping.

Timestamping is a fundamental aspect of many computer systems and applications, enabling various functionalities and operations that rely on time-related information. The specific implementation and usage of timestamps depend on the programming language, platform, and requirements of the system or application.