Programming in Linux

AY 2023-24

GITAM UNIVERSITY

v0.1, 05th Feb 2024

A University should be a place of light, of liberty, and of learning.

Rajesh Sola

www.gitamedu.com

GITAM
(DEEMED TO BE UNIVERSITY)

Outline

Installing & Preparing Linux

Building simple programs - GNU Tools, GCC options

Building Multi File programs, header files

Smart Building with make (Makefiles)

Code Visualization - Pythontutor

Memory Leak & Heap Error detection

Source Formatting

Static Analysis

GITAM
(DEEMED TO BE UNIVERSITY)

Linux Installation

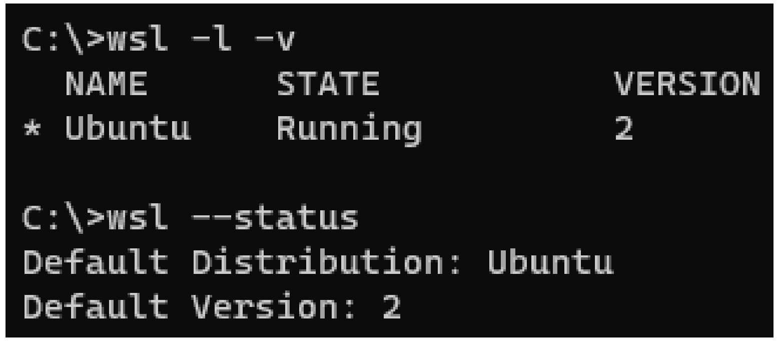
- Dual Boot Linux
- Linux over VirtualBox/VMWare
- Linux over WSL

Preferred Linux Distribution – Ubuntu 22.04 / LTS version

WSL How to

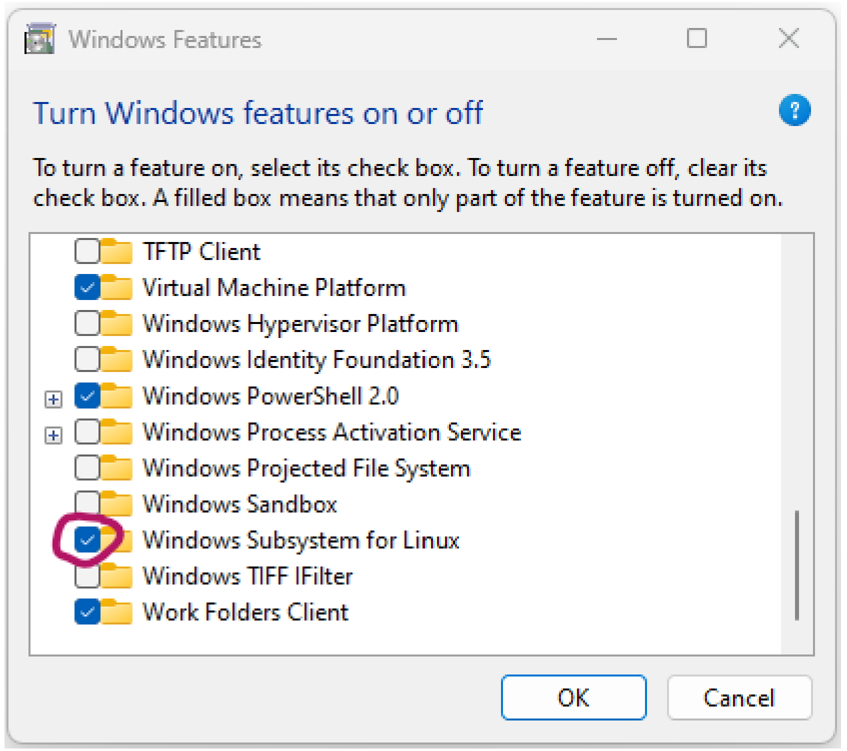
- Enable in WSL in "Turn Windows Features On or Off"
- In Windows Command Prompt

```
wsl --update
wsl --install -d Ubuntu
wsl --status
wsl -l -v
wsl
```



```
C:\>wsl -l -v
  NAME      STATE           VERSION
* Ubuntu    Running         2

C:\>wsl --status
Default Distribution: Ubuntu
Default Version: 2
```



Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ☐ TFTP Client
- ☑ Virtual Machine Platform
- ☐ Windows Hypervisor Platform
- ☐ Windows Identity Foundation 3.5
- ☑ Windows PowerShell 2.0
- ☐ Windows Process Activation Service
- ☐ Windows Projected File System
- ☐ Windows Sandbox
- ☑ Windows Subsystem for Linux
- ☐ Windows TIFF IFilter
- ☑ Work Folders Client

Enjoy the Linux enabled on top of WSL

Simple Commands

ls	• List files & directories in current/specified dir
pwd	• Prints path of present working directory
mkdir	• Create a new directory
cd	• Change the directory
cp	• Copy 1 or more files
mv	• Move/Rename the files
rm	• Remove the files & directories

Simple Tutorial:- https://linuxjourney.com/lesson/the-shell

Programming in Linux

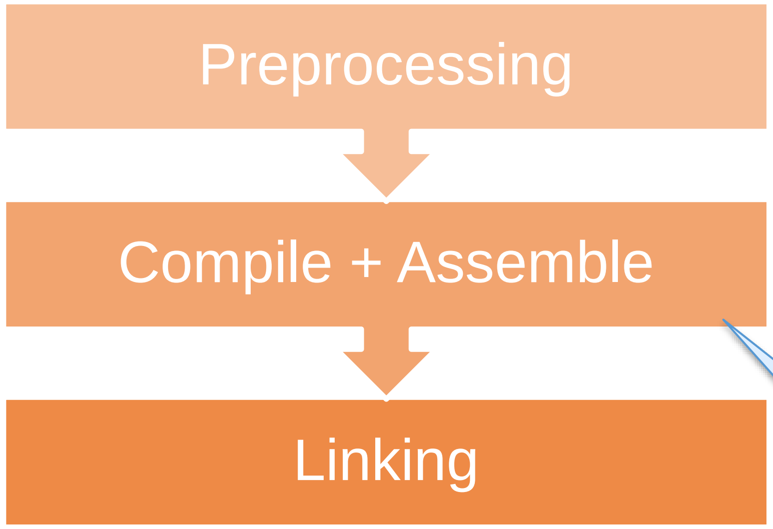
- Open Integrated Terminal in VS Code and Switch to WSL
- Install gcc and other build essentials

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

- Write a simple program, say hello.c
- Build using gcc, with one of the following methods

```
    gcc hello.c -o hello       # hello.out, h.out
    ./hello
    gcc hello.c
    ./a.out
```

Build Phases & GNU Tools, GCC options



```
gcc options & GNU Tools for these stage
gcc -E sample.c     # stop with preprocessing, tool : cpp
gcc sample.c -c     # stop with compilation, generates sample.o
gcc sample.o        # linking, one or more obj files + std libs
                    # ld is the actual tool for linking

gcc -S sample.c     # generates equivalent assembly, sample.s
gcc sample.s -o sample.o # assemble the code, generated obj file,
                    # as is the actual tool for assembling
```

Compile only, generates object file

Where is the definition of printf?

std C library (libc.a/libc.so)
stdio.h just provides prototype

GITAM (DEEMED TO BE UNIVERSITY)

Multi File Programming

test.c

```
#include<sdio,h>

int main()
{
    int a,b,c,d;
    a=10;
    b=20;
    c=sum(a,b);
    d=squre(a);
    printf("c=%d,d=%d\n",c,d);
    return 0;
}
```

sum.c

```
int sum(int x,int y)
{
  int res = x + y;
  return res;
}
```

sqr.c

```
int square(int x)
{
  int res = x * x;
  return res;
}
```

```
gcc test.c -c
gcc sum.c -c
gcc sqr.c -c
gcc test.o sum.o sqr.o -o all.out
```

Observe errors with these kind of commands

```
gcc test.c (or) gcc test.o
gcc sum.c (or) gcc sum.o
```

GITAM
(DEEMED TO BE UNIVERSITY)

Adding Prototype & Header Files

test.c

```c
#include<sdio,h>

int sum(int,int);
int square(int);

int main()
{
     int a,b,c,d;
    a=10;
    b=20;
    c=sum(a,b);
    d=squre(a);
    printf("c=%d,d=%d\n",c,d);
    return 0;
}
```

fun.h

```c
#ifndef __FUN_H
#define __FUN_H

int sum(int,int);
int square(int);

#endif
```

test.c

```c
#include<sdio,h>

#include "fun.h"

int main()
{
    int a,b,c,d;
    a=10;
    b=20;
    c=sum(a,b);
    d=squre(a);
    printf("c=%d,d=%d\n",c,d);
    return 0;
}
```

GITAM (DEEMED TO BE UNIVERSITY)

Makefiles

Makefile

```
all : all.out
all.out : test.o um.o sqr.o
         gcc test.o sum.o sqr.o -o all.out
test.o : test.c fun.h
         gcc test.c -c
sum.o : sum.c fun.h
         gcc sum.c -c
sqr.o : sqr.c fun.h
         gcc sqr.c -c
clean:
         rm -rf (.o all.out
run: all.out
         ./all.out
```

```
all : all.out
all.out : test.o um.o sqr.o
         gcc $^ -o $@
test.o : test.c fun.h
         gcc $< -c
sum.o : sum.c fun.h
         gcc $< -c
sqr.o : sqr.c fun.h
         gcc $< -c
clean:
         rm -rf (.o all.out
run: all.out
         ./all.out
```

Using special variables - $@, $^, $<

```
make clean
make
make run
```

Modify one of the file (test.c or sum.c or sqr.c or fun.h) and re-run make, observe which commands are repeated and which are not

Code Visualization

- Visualizing C Programs using **pythontutor**
  - Visualizing Execution Flow – Step by Step
  - Visualizing Memory Layout (Stack, Heap)
  - Visualizing Stack Frames
- Examples
  - Example-1: Dynamic Memory
  - Example-2: Recursion
  - Example-3: Pass by reference

GITAM
(DEEMED TO BE UNIVERSITY)

Heap Analysis

- Memory Leak and Heap Error Detection
- valgrind tool usage

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main()
{
    int *ptr;
    int n=10;
    ptr = malloc(n * sizeof(int));
    srand(time(0));
    for(int i=0;i<n;i++)
         parr[i]=rand()%100;
    for(int i=0;i<n;i++)
         sum += parr[i];
    //free(parr);
    return 0;
}
```

```
gcc -g dyndemo.c –o dyndemo
valgrind ./dyndemo
```

- ❑ Observe memory leaks in absence of free
- ❑ Observe clean report when dynamic memory is freed properly

Code Style

Source Formatting – Coding Style

Naming Conventions

- Camel Case
- Snake Case
- Pascal Case
- or any other convention followed by project team

Meaningful Names

```
clang-format sample.c
clang-format –i sample.c
```

- Which is the default style followed by clang-format?
- Explore other styles supported by clang-format

Static Analysis

Coding Standards

- MISRA
- SEI CERT
- Custom Standards by Projects/Communities

Free and/or Open-Source Tools

- cppcheck
- clang
- clang-tidy

Proprietary Tools for Static Analysis

- Klockwork
- Polyspace
- Helix QA-C, QAC++
- LDRA Tools
- Sonarlint
- Coverity
- Parasoft

```
TODO:- Analyzing few examples using
cppcheck/clang-tidy
```

GITAM (DEEMED TO BE UNIVERSITY)

Static Analysis

```
#include<stdio.h>

int main()
{
    int *ptr;
    ptr = fetch();
    //do something
    printf("val=%d\n",*ptr);
    return 0;
}
int *fetch()
{
    int x=100;
    return &x;
}
```

```
#include<stdio.h>

int main()
{
    int a=5, b;
    b = a++ * a++ * a++;
    printf("a=%d,b=%d\n",a,b);
}

//Undefined behavior in absence
//of sequence points
```

Unused Variables

Uninitialized Variables

Incompatible pointer assignments/operations

```
cppcheck example.c
clang –analyze example.c
clang-tidy example.c
```

Some rule sets [standards / custom rules]:-

- https://rules.sonarsource.com/c/
- https://wiki.sei.cmu.edu/confluence/display/c
- https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard
- Embedded System development Coding Reference guide

Further Topics

- Basic GIT Familiarity
- Patches - generating & applying, using diff / git
- Static & Dynamic Libraries – creation & linking
- Debugging using gdb/lldb

Stay tuned for further updates!!

GITAM
(DEEMED TO BE UNIVERSITY)

THANK YOU