**LINUX TOOLS**

**DAY 04 (25-10)**

* **GCC –** C compiler
  + - * -wall to address all the warnings.
      * w --- warning
      * Debugging options usually use -g and there are other options.
      * -pthread used as a preprocessor options usually.
      * -I to include directory.
* **Commands:**

-E preprocessed state

-c compile only

-s preprocessed state to assembly file

-o assembly to object file

-I to include directory

-m to enter a message

-l to include libraries

Cd – to come to the root directory

Cd .. – to move to the previous directory

tree – to get tree structure of directory.

Command to get intermediate files(.s,.o,.): -save-temps (temporary all the files to be saved)

\*---Remove all eg: - rm ./bin/\*(removes all the files from bin directory)

Cd **../**src – to change into parent directory and then to the required directory.

Cp -rf source destination --- to copy file/directory from source to destination

**DAY 05 (28-10)**

* **Compiling the code without using make command**

Mkdir mainprj

Cd mainprj

Mainprj>mkdir inc obj

Created header file

Cd inc

Mainprj/inc/>vi calc.h

cd../

cd src

mainprj/src/> vi calc.c

mainprj/src/> vi main.c

cd ../

mainprj/>gcc -c ./src/calc.c -I./inc/ -o ./obj/calc.o

mainprj/>gcc -c ./src/main.c -I./inc/ -o ./obj/main.o

* **Errors**
* **implicit declaration:** this error occurs when we used the interface, but interface not declared.
* **Segmentation fault** (core dumped): trying to assess address beyond the scope.
* **Stack smashing detected:**
* **Exited normally:**
* Invalid use of void expression – it is difficult for a void pointer to dereference so we explicitly typecast it
* Double free detected (aborted (core dumped)) –
* To copy files/directories from one repository to another repository
* First clone to the repository from which files are to be copied (sir’s github repository: -git clone https://github.com/bhimatak/CGBatch17Oct2024.git)
* **Make file:** It will search/fetch for only source files which are latest modified and compiles accordingly.
* To create makefile : - vi Makefile --- to create makefile
* Make -f ./scripts/Makefile --- to run a makefile written in the shell language
* **Code coverage tool (gcov)**: we get to know the no.of lines executed in a given code
* To check the coverage of the program lines:
* gcc -fprofile-arcs -ftest-coverage file1.c file2.c …….
* After compiling above command we get file.*gcno (this file contains binary data)*
* After executing (./a.out) we get file.gcda (it is again a binary file)
* gcov file.c ---- to get the percentage of the code executed and is useful in unit test case
* **git hub:**
* Create and clone the github repository, git clone https:…. --- command to clone the repository
* Create a directory using repository in command prompt
* git add . --- to add all the edited files into the local repository, git add filename --- to add specified file into the local repository.
* git commit -m “ “ --- to pass a message and commit the changes
* git push origin main --- to push the local added files to the github
* **splint**: it does static analysis of source code. To check memory leakages
* check for bugs

**DAY 06 (29-10)**

* **ctags:** find out where the functions are declared
* sudo apt install universal-ctags to install the ctags tool
* ctags -R . --- command used to generate the index data file i.e tags file
* set tags path of tag file --- in command mode in the vi file
* **navigating:**
* cltr[] --- to navigate (to search for the function declared in the current function)
* cltr+t --- to move backward
* **cscope**:
* sudo apt install cscope – to install the cscope tool
* rebuild the data base
* cscope.out file formed or produced
* cscope find . -name ‘\*.[ch]’
* cltr+d coming out of that csope
* **gprof**: to check amount of time consumed in different functions
* gcc -o application -pg filename
* gmon.out file is obtained after compiling above command
* executing gmon.out file we get g.profile file
* **gdb**: The GNU debugger - check the flow
* debugger used to check run time errors if any are present
* break – debugger utility
* watch – to keep an eye on the variable change or we can use the print to know how the variables are changing.
* gdb a.out command used for execution after compiling the file i.e gcc -g filename.
* Options used are break (b), run(r), continue(c), list (l), next (n), step (s) – to run the program inside the function

**C**

* Dennis Ritchie developed c language in 1972. It took him 3 years 8 months to develop this language.
* Business package- notepad, excel, IRCTC

System software- device drivers, software commands like date, time

C is suitable for both

* C is **faster and efficient** because other languages need virtual box unlike C and file size is lesser
* C is suited for **structured programming**
* Highly portable
* **Structure of c program**

1. Documentation section: user purpose

/\*

Description: program details, algorithms, libraries used are mentioned

Author: CSP

Date of creation (DOC)/ Date of modification (DOM):

Version: 0.1v- new creation

Usually, 1.0 is used as stable version with no bugs and errors

\*/

1. Link section: including header files

* Libraries references are linked in link session

1. Definition: prototype of function, macros definition
2. Global declaration section: variables are written and accessed

* Variable and function names should be meaningful to maintain coding standards
* Ideally function names camel cases

1. Main () function section:
2. Declaration part: all variables should be declared at first
3. Executable part: executing expression, function (anything performing task)

* Every function has its own address same goes with main function also
* We can have more than one main function
* commands
* Carriage return \r – cursor returns to the start of the same line we are working

New line \n - cursor points to the start of the next line

\b – backspace the values

* int x=10, y=20;

printf(“%d”, x, y); ---- o/p: 10

* C tokens: keywords, identifiers, constants, strings, special symbols, operators
* Naming convention shouldn’t use numeric in the start of the variable
* anything in “ “ is string, ‘ ‘ is a single character
* mod operator can’t be operated on float/double
* operators
* datatypes – fundamental, derived (array, function, strings), user defined
* modifiers – signed, unsigned, long, short
* identifiers – identify the variables, functions and other user defined variables
* the variables cannot be start with numeric, $ but can be started \_, letters.
* int a=b=c=10;

a=b=c=50;

leads to the compile error we can’t assign values like this is C

**DAY-7 (4-11)**

* **Managing input output operations**: reading a single character can be done by using the function getchar, this can also be done by scanf().

Eg: printf(), scanf(), getchar(), gets(), putchar(), puts()

* Two types of formatted I/O statements:

1. Formatted I/O – this enables the user specify the type of the data or the way it is read or written. Eg: scanf(), printf()
2. Unformatted I/O – it does not specify the type of the data or the way it is read or written. Eg: getchar(), gets(), putchar(), puts() etc

* %lf – for double

%u – usigned value

* Int num = 12345;

Scanf(“%3d”, &num); - o/p 123

3 is the field width of the input number

Here 45 will be in the temporary buffer

* Printf(“%05d”, 678);

0 0 6 7 8 – o/p

* Printf(“% 5d”, 678);

6 7 8 – o/p

If you give more than 5 digits of number, we get the output printed.

* To clear/flush out the buffer – fflush(stdin) function is used ( it will not work in the linux). It should be used before the scanf statement.

To clear buffer we can give scanf(“ “) – it is useful for the single character constant

To clear buffer numeric constant, we use

* Space or new line character is an end of string in scanf

Eg: name = pravallika c – o/p: Pravallika

To overcome this we give scanf(“%[^/n]s”, name); (it means it will scan all characters until new line)

* All format specifiers in scanf should not have any spaces it will slightly affect the output

Eg : scanf(“%d %d”,&n1,&n2); - not a problem

scanf(“%d%d”, &n1, &n2); - not a problem

scanf(“ %d%d”,&n1,&n2); - not a problem only for integers if the immediate one is character it will take as 3 inputs

scanf(“%d %d ”,&n1,&n2); - slight affect

* **\t, \n should not be used in scanf (**avoid unprintable characters)

We can use anything in the printf

* sprintf is used to convert integer to string in linux

itoa in unix environment (p4.c)

* atoi possible in linux used to change a string to integer (only possible for numerical string)
* sprintf – to convert the integer into string. Should include library function when using sprintf

Eg: #include <stdio.h>

2 #include <stdlib.h>

3

4 int main()

5 {

6 int i,j;

7 char w[5] = "2002";

8 char buff[1024];

9 char buff1[1024];

10

11 int res=0;

12 scanf("%d%d",&i,&j);

13 printf("\ni=%d\tj=%d",i,j);

14 sprintf(buff,"\ni=%d\tj=%d",i,j);

15 printf("\nBuffer value: %s",buff);

16 sprintf(buff1,"%d",i);

17 printf("\nBuff1: %s",buff1);

18 res = atoi("2002")+2;

19 printf("\nResult = %d",res);

20 printf("\n\n");

21 return 0;

22 }

**O/p:** 10 20

i=10 j=20

Buffer value:

i=10 j=20

Buff1: 10

Result = 2004

* **decision making and branching**
* when we have more than one conditions and to test/ check, we use branching
* if statement, if else, nested if else, switch statement, else if ladder
* nested if:

if(cond1)

{

If(cond2)

{

}

Else

{

}

}

Else {

If(cond3)

{

}

Else

{

}

}

**DAY-8 (5-11)**

* else if ladder
* it tries to check all the if else if conditions unless the condition satisfies or then goes to else condition, so it takes time.
* if condition holds strings, float values we use else if ladder
* can test relational conditions
* switch – to check single characters, ascii, integer conditions
* switch(cond)

{

case ‘1’:

…

break;

case ‘2’:

…

break;

default:

…

break; - it is not mandatory because this is the final label

}

* should not have duplicate cases in switch

1. case 66:

case ‘B’:

1. case ‘B’:

case ‘B’:

gives an error as duplicate cases are not allowed

* **looping:** do while, while, for
* initialization

condition check

statements to be executed

counter

* two types of loops: entry controlled (while, for loop), exit controlled (do while)
* do {

sts

} while(cond);

* ;(semi colon) is used after while condition in do while because
* For(initialization section ;condition section ;counter section){}
* Initializing and counter we can use more than one variable in for loop so it is powerful than while loop
* We can vomit one or more section in **for loop**

**DAY-9 (6-11)**

* For loop being an entry-controlled loop change to exit controlled loop (m1.c)
* Goto command
* Syntax: label name: ----- declaration

Sts;

goto label name; -----calling

* Label should be declared from start of the line i.e root
* **Arrays**
* Arrays are homogeneous which can contain only similar data type of elements
* Basic syntax: datatype array name[size];

Eg: int arr[10];

int b[2] = {1,2} ---- initialized array

* Index values of arrays should always be a whole number but not float
* To calculate the size of an array=sizeof(arr)/sizeof(arr[0])
* y array indexing starts with 0
* formula for accessing array content : [base address +(index value \* size of data type)]
* types of arrays:

1. **static array:** the size of array is allocated before to the compilation time

defined in stack segment

eg: int arr[5];

1. **dynamic array:** the size of array is allocated at run time

defined in heap segment

eg: defined using malloc, calloc, realloc ---- stdlib.h

1. **stretchable array:** size of the array is increased or decreased on the need for dynamic array

malloc, calloc, realloc

1. **mutable array:** size of the array is allocated at the time of linking and before execution

* 2D array:
* datatype array name[row][col]
* array rule
* arr[2] is equal to 2[arr]
* arr[arr[2]] is also possible
* arr[2.5] imples arr[2], arr[10.9] imples arr[10]
* **Functions:**
* Functions are two types:
  1. Std library function:

Printf, sqrt, abs, pow

* 1. User defined: user is defining their own task to be performed
* Void disp() function without input args without return datatype

int add(int, int) function with input args with return datatype

void changename(char\*) function with input args without return datatype

**DAY-10 (7-11)**

* Recursive function
* **Pointers**
* pointers hold the address not to a value, points to a particular address
* syntax: datatype \*pointer name;

eg: int \*ptr ---- ptr is a pointer that is pointing to the integer datatype

* pointer doesn’t belong to any datatype
* types of pointer:
* NULL pointer – pointer pointing to nothing but NULL

Eg: int \*ptr= NULL

* Void pointer/ generic pointer – it can hold address of any datatype

Eg: void \*ptr

* Wild pointer – pointer without initialization

It should not be the case every pointer should be initialized otherwise it can point to anything

* Dangling situation – pointer pointing to a address where in the address reference is lost

Static key word declaration to overcome dangling situation

* Every pointer has a same size that is 4/8 bytes depending on the bitness of the system
* it is difficult for a void pointer to dereference, so we explicitly typecast it. Only generic pointers require typecasting
* refer notes about pointers
* rules of pointer:
* &\* ---- will nullify each other
* Operand[] -> \*operand similarly \*op -> op[]

Eg: ptr = &b[0];

ptr = &\*(b+0);

ptr = (b+0) -> b;

* int \*ptr=NULL;

\*ptr=101;

printf("value stored at ptr: %d\n",\*ptr);

gives Segmentation fault because we did not give correct address

* when a member of a structure is a pointer which is pointing to itself is called self-referencing pointer
* storage classes:
* **static -** the variable will be there throughout the program (till the end)
* **extern -**
* **auto –** local variable declaration **(**block scope**)**
* **register –** variable stored in cpu

time taken to access the variable stored in register is faster than variable stored in stack frame

* malloc and calloc have same characteristics except malloc has one argument, and calloc has two

memset, memcmp, memcpy