**Experiment 1 Date:**

**Advanced Use of GCC**

**Aim:**

Create appropriate C program. Compile and generate output using gcc command and its important options -o, -c, -D, -I, -g, -O, -save-temps

**About GCC Compiler**

GCC is a Linux-based c compiler released by the free software foundation which is usually operated via the command line. It often comes distributed freely with a Linux installation, so if you are running Unix or a Linux variant you will probably have it on your system. You can invoke gcc on a source code file simply by typing:-

**gcc filename**

The default executable output of gcc is "a.out", which can be run by typing"./a.out". It is also possible to specify a name for the executable file at the command line by using the syntax " -o outputfile", as shown in the following example: -

**gcc filename -o outputfile**

Again, you can run your program with "./outputfile". (the ./ is there to ensure to run the program for the current working directory.)

Note: if you need to use functions from the math library (generally functions from math.h" such as sin or sqrt), then you need to explicitly ask it to link with that library with the "-1" flag and the library "m":

**gcc filename -o outputfile -lm**

**Program 1**

**// Write a C program 'sum.c' to add two numbers. Read the input from Standard Input and write output to Standard output**

#include<stdio.h>

void main()

{

int a,b,sum=0;

printf("Enter 2 numbers:");

scanf("%d%d",&a,&b);

sum=a+b;

printf("Sum of %d and %d =%d\n",a,b,sum);

}

**Output**

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc sum.c

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ ./a.out

Enter 2 numbers:12 34

Sum of 12 and 34 =46

**Important Options in GCC**

**Option: -o**

To write and build output to output file.

**Output**

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ ./sum\_out

Enter 2 numbers:12 34

Sum of 12 and 34 =46

This compiles source.c with profiling support, allowing you to use profilers like gprof.

**Option: -save-temps**

To save temporary files generated during program execution.

**Output**

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc -save-temps sum.c

This will generate intermediate files, like sum.i (pre-processed source) and sum.s (assembly code), in addition to the final executable.

**Option: -g**

gcc -g generates debug information to be used by GDB debugger.

**Output**

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc -g sum.c -o sum\_out

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gdb sum\_out

GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1

Reading symbols from sum\_out...

(gdb) run

Starting program: /home/mits/24lab/ds24/sum\_out

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library

"/lib/x86\_64-linux-gnu/libthread\_db.so.1".

Enter 2 numbers: 12 45

Sum of 12 and 45 =57

[Inferior 1 (process 6070) exited normally]

(gdb) quit

This compiles sum.c with debug information, enabling you to debug the resulting executable file.

**Option: -c**

gcc -c compiles source files to object files without linking.When we compile a program the ‘C’ compiler will generate object files “.o” files.After that linker will generate a “.out” file. That means,it is a two steps process; one step is to compile the program and another step is to link the object files and generate the executable file.

**Output**

its@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc -c sum.c

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc sum.o -o sum\_out

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ ./sum\_out

Enter 2 numbers:23 44

Sum of 23 and 44 =67

Here, GCC compiles the sum.c file and generates an executable named sum\_out.This will generate an object file sum.o that can be linked separately. This is the way to create an object file and use the functions in object files from different code modules.

**Program 2**

/\* Write a program with preprocessor directives. #ifdef is used to include a section of code if a certain macro is defined by #define.\*/

// myfile.c  
#include <stdio.h>

void main()

{

#ifdef SAMPLE

printf("With preprocessor directive= %d\n",SAMPLE);

#else

printf("With out #ifdef\n");

#endif

}

**Important Option in GCC**

**Option: -D**

gcc -D defines a macro to be used by the preprocessor.

**Output**

1. **Build *myfile.c* and run it with the maro SAMPLE defined:**

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc -D SAMPLE myfile.c -o myfile

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ ./myfile

With preprocessor directive

1. **Build *myfile.c* and run it without the macro SAMPLE defined:**

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc myfile.c -o myfile

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ ./myfile

With out #ifdef

**Program 3**

/\* Create a program with macro and saved this as header file. Create another C program which include this header file. \*/

// myheader.h  
 #define NUM1 5

save this file as **src/myheader.h**

// myfile1.c  
 #include <stdio.h>  
 #include "myheader.h"  
  
 void main()  
 {  
 int num = NUM1;  
 printf("num=%d\n", num);  
 }

save this file as myfile.c

**Important Option in GCC**

**Option: -I**

gcc -I include directory of header files. This flag is used to specify additional directories where header files are located. It helps the preprocessor find the necessary headers when compiling the code.

**Output**

**a) Build *myfile1.c* without include directory *src* :**

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc myfile1.c

myfile1.c:2:10: fatal error: myheader.h: No such file or directory

2 | #include "myheader.h"

| ^~~~~~~~~~~~

compilation terminated.

**b) Build *myfile1.c* with include directory *src* :**

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ gcc -I src myfile1.c -o myfile1\_out

mits@mits-ThinkCentre-neo-50t-Gen-3:~/24lab/ds24$ ./myfile1\_out

num=5

**Experiment 2 Date:**

**Familiarisation with GDB**

**Aim:**

Write a C program ‘mul.c’ to multipy two numbers. Read the input from Standard Input and write output to Standard output. Compile and generate output which is then debug with gdb and use the important commands break, run, next, print, display

**Program**

#include<stdio.h>

void main()

{

int a=5;

int b=10;

int mul=a\*b;

printf("\nProduct=%d\n",mul);

}

**Output**

mits@mits:~/Desktop/S1MCA/ADS\_lab$ gcc mul.c -o mul\_out

mits@mits:~/Desktop/S1MCA/ADS\_lab$ ./mul\_out

**Important Commands in GDB**

**option -g**

gcc -g generates debug information to be used by GDB debugger.

acer@acer-Aspire-E5-575:~/clab$ gcc -g mul.c -o mul\_out

acer@acer-Aspire-E5-575:~/clab$ gdb mul\_out

**Output**

GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90

Copyright (C) 2022 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86\_64-linux-gnu".

Type "show configuration" for configuration details.

For bug reporting instructions, please see:

<https://www.gnu.org/software/gdb/bugs/>.

Find the GDB manual and other documentation resources online at:

http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from sum1...

(gdb)

* 1. **Command: run**

Executes the program from start to end.

**Output**

**(gdb) run**

Starting program: /home/mits/Desktop/Poojas1MCA/sum1

[Thread debugging using libthread\_db enabled]

Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".

Enter 2 numbers : 10 20

Product : 200 [Inferior 1 (process 23588) exited normally]

* 1. **Command: l - for Display the code**

Type “l” at gdb prompt to display the code.

**Output**

(gdb) l

1 #include <stdio.h>

2 void main()

3 {

4 int a,b,p;

5 printf("Enter two numbers=");

6 scanf("%d %d",&a,&b);

7 p=a\*b;

8 int x=10;

9 printf("\nsum= %d\n",p);

10 }

**c. Command: break**

Sets a breakpoint on a particular line.

**Output**

**(gdb) break mul.c:4**

Breakpoint 1 at 0x11a4: file mul.c, line 5.

**(gdb) run**

Starting program: /home/acer/clab/mul\_out

Breakpoint 1, main () at mul.c:5

1. printf("Enter two numbers=");
2. **Command: next**

Executes the next line of code without diving into functions.

**Output**

**(**gdb) next

6 scanf("%d %d",&a,&b);

(gdb) next

Enter two numbers=3 4

7 p=a\*b;

1. **Command: clear**

Clears all breakpoints.

**Output** 9 printf("\nsum= %d\n",p);

10 }

**c. Command: break**

Sets a breakpoint on a particular line.

**Output** 9 printf("\nsum= %d\n",p);

10 }

**c. Command: break**

Sets a breakpoint on a particular line.

**Output**

**(gdb) break mul.c:4**

Breakpoint 1 at 0x11a4: file mul.c, line 5.

**(gdb) run**

Starting program: /home/acer/clab/mul\_out

Breakpoint 1, main () at mul.c:5

1. printf("Enter two numbers=");
2. **Command: next**

Executes the next line of code without diving into functions.

**Output**

**(**gdb) next

6 scanf("%d %d",&a,&b);

(gdb) next

Enter two numbers=3 4

7 p=a\*b;

1. **Command: clear**

Clears all breakpoints.

**Output**

**(gdb) break mul.c:4**

Breakpoint 1 at 0x11a4: file mul.c, line 5.

**(gdb) run**

Starting program: /home/acer/clab/mul\_out

Breakpoint 1, main () at mul.c:5

1. printf("Enter two numbers=");
2. **Command: next**

Executes the next line of code without diving into functions.

**Output**

**(**gdb) next

6 scanf("%d %d",&a,&b);

(gdb) next

Enter two numbers=3 4

7 p=a\*b;

1. **Command: clear**

Clears all breakpoints.

**Output**

(gdb) clear

Aborted (core dumped)

1. **Command: print**

Displays the value of a variable.

**Output**

(gdb) print a

(gdb) a 10

1. **Command: display**

Displays the current values of the specified variable after every step.

**Output**

(gdb) display a

1: a=10

1. **Command: quit**

Exits out of GDB.

**(gdb) quit**