Experiment 1 **Date: 16/10/2024**

Advanced Use of GCC

Aim:

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

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,c;

printf("Enter two number:");
scanf("%d%d",&a,&b);

c=a+b;

printf("Sum is %d",c);

}
```

Output

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc sum.c
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ ./a.out sum.c
Enter two number: 10 20
Sum is 30
```

Important Options in GCC

Option: -o

To write and build output to output file.

Output

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$gcc sum.c -o sum_out
```

Here, GCC compiles the sum.c file and generates an executable named sum_out.

Option: -save-temps

To save temporary files generated during program execution.

Output

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc -save-temps -o my_pgm sum.c
```

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

Output

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc -pg -o my_pgm sum.c
```

Option: -g

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

Output

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc -g sum.c -o sum_out
```

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ 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/data/sum_out

Enter two number: 2  4

sum= 6

[Inferior 1 (process 7233) exited normally]

(gdb) quit
```

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

Option: -O

gcc -O sets the compiler's optimization level.

<u>Output</u>

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$  gcc -O3 -o my_pgm sum.c
```

This compiles sum.c with a high level of optimization.

Option: -c

To compile source files to object files without linking. when we compile a program the ‘c’ compiler will generate object file “.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.

<u>Output</u>

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc -c sum.c

mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc sum.o -o a_out

mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ ./a_out

Enter two number:4 2

Sum is 6
```

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 function in object file from different code module.

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>
#define SAMPLE 10
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

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

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc -D SAMPLE myfile.c -o
myfile
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ ./myfile
With preprocessor directive= 1
```

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

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc myfile.c
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ ./a.out
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**

```
// myfile.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 *myfile.c* without include directory *src* :

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc myfile.c
myfile.c:2:18: fatal error: myheader.h: No such file or directory
   2 |        #include "myheader.h"
         compilation terminated.
```

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

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc -I src myfile.c -o myfile_out
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ ./myfile_out
num=5
```

Experiment 2 **Date: 22/10/2024**

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,b,mul;
        printf("Enter two number:");
        scanf("%d%d",&a,&b);
        mul=a*b;
        printf("Multiple of two number is %d",mul);
}
```

Output

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc mul.c -o mul_out
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ ./mul_out
Enter two number : 4  5
Multiple of two number is 20
```

Important Commands in GDB

option -g

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

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc -g mul.c -o mul_out
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gdb mul_out
```

Output

```
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
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 mul_out...
(gdb)
```

a. Command: run

Executes the program from start to end.

<u>Output</u>

(gdb) run

```
Starting program: /home/mits/ds/mul_out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter two number:3 6
Multiple of two number is 18[Inferior 1 (process 5145) exited normally]
```

b. Command: l - for Display the code

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

<u>Output</u>

```
(gdb) l
1       #include<stdio.h>
2       void main()
3       {
4       int a,b,mul;
5       printf("Enter two number:");
6	scanf("%d%d",&a,&b);
7	mul=a*b;
8	printf("Multiple of two number is %d",mul);
9	}
```

c. Command: break

Sets a breakpoint on a particular line.

<u>Output</u>

```
(gdb) break mul.c:6
Breakpoint 1 at 0x5555555551b8: file mul.c, line 6.
(gdb) run
Starting program: /home/mits/data/mul_out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at mul.c:6
6	scanf("%d%d",&a,&b);
```

d. Command: next

Executes the next line of code without diving into functions.

<u>Output</u>

```
(gdb) next
Enter two number:4 5
7	mul=a*b;
(gdb) next
8	printf("Multiple of two number is %d",mul);
```

e. Command: clear

Clears all breakpoints.

<u>Output</u>

```
(gdb) clear
No breakpoint at this line.
```

f. Command: print

Displays the value of a variable.

Output

```
(gdb) print a
$1 = 0
```

g. Command: display

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

Output

```
(gdb) display a
1: a = 0
```

h. Command: quit

Exits out of GDB.

Output

```
(gdb) quit
A debugging session is active.
Inferior 1 [process 5710] will be killed.

Quit anyway? (y or n) y
```

Experiment 3 **Date:23/10/2024**

Familiarisation with gprof

Aim:

Write a program for finding the sum of two numbers using a function. Then profile the executable with gprof.

gprof

Gprof is a profiling program which collects and arranges statistics on our programs.Basically, it looks into each of our functions and inserts code at the head and tail of each one to collect timing information. Then, when we run our program normally, it creates "gmon.out". In order to produce profiling information, the program must be compiled with specific options to tell it to record this information. The important options are: ‘-pg' (profile graph) and '-g' (debug information) must be included.

Program

```
#include<stdio.h>
int sum(int x, int y){
   return x+y;
}
void main(){
   int a,b;
   printf("Enter 2 numbers : ");
   scanf("%d %d",&a,&b);
   printf("Sum : %d",sum(a,b));
}
```

Output

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc sum.c
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ ./a.out
Enter 2 numbers : 10 20
Sum : 30
```

Option: -pg

To compile a source file for profiling, specify the ‘-pg’ option when we run the compiler. -pg’, alters either compilation or linking to do what is necessary for profiling.

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc -pg -o sum.out sum.c
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ ./sum.out
Enter 2 numbers : 20 30
Sum : 50
```

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gprof ./sum.out gmon.out > pgm3.txt
```

pgm3.txt

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        1     0.00     0.00  sum
```

Experiment 4 Date:29/10/2024

Different types of functions

Aim:

Write a program for finding the sum of two numbers using different types of functions.

a) Function with argument and return type
b) Function with argument and no return type
c) Function without argument and return type
d) Function without argument and no return type

This program should have menu driven options.

Algorithm:

main()

```
1  Start
2  Declare ch, num1,num2
3  Display choices.
4  Read option ch.
       if ch==1 call sum1()
       if ch==2 input num1 and num2 and call sum2(num1,num2).
       if ch==3 call sum3() and print the result from sum3().
       if ch==4 input num1 and num2 and call sum4(num1,num2) and print the
       result from sum4().
5  Repeat steps 3 while ch>0&&ch<4.
6  Stop.
```

void sum1()

```
1  Start
2  Declare a and b.
3  Read a and b.
4  Print a+b.
5  Exit.
```

Void sum2(int a, int b)

```
1  Start
2  Print a+b.
3  Exit.
```

int sum3()

```
1  Start
2  Declare a and b.
3  Read a and b.
4  Return a+b.
5  Exit.
```

int sum4(int a, int b)

```
1  Start
2  Return a+b
3  Exit.
```

Program

```
#include <stdio.h>
int sum1() {
   int a, b;
   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);
   return a + b;
}

void sum2(int a, int b) {
   printf("Sum: %d\n", a + b);
}

int sum3() {
   int a, b;
   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);
   return a + b;
}

void sum4() {
   int a, b;
   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);
   printf("Sum: %d\n", a + b);
}

int main() {
   int ch, num1, num2;

   do {
      printf("Menu:\n");
      printf("1. Function with argument and return type\n");
      printf("2. Function with argument and no return type\n");
      printf("3. Function without argument and return type\n");
      printf("4. Function without argument and no return type\n");
      printf("Enter your choice (0 to exit): ");
      scanf("%d", &ch);

      switch(ch) {
         case 1: {
            int result = sum1();
            printf("Sum: %d\n", result);
            break;
         }
         case 2:
            printf("Enter two numbers: ");
            scanf("%d %d", &num1, &num2);
            sum2(num1, num2);
            break;
            case 3: {
                int result = sum3();
                printf("Sum: %d\n", result);
                break;
            }
            case 4:
                sum4();
                break;
            case 0:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while(ch != 0);

    return 0;
}
```

Output

```
mits@mits-ThinkCentre-neo-50t-Gen-3:~/data$ gcc arrgu.c

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

Menu:

1. Function with argument and return type

2. Function with argument and no return type

3. Function without argument and return type

4. Function without argument and no return type

Enter your choice (0 to exit): 1

Enter two numbers: 2 5

Sum: 7

Menu:

1. Function with argument and return type

2. Function with argument and no return type

3. Function without argument and return type

4. Function without argument and no return type
```

```
Enter your choice (0 to exit): 3

Enter two numbers: 3 4

Sum: 7

Menu:

1. Function with argument and return type

2. Function with argument and no return type

3. Function without argument and return type

4. Function without argument and no return type

Enter your choice (0 to exit): 4

Enter two numbers: 3 6

Sum: 9

Menu:

1. Function with argument and return type

2. Function with argument and no return type

3. Function without argument and return type

4. Function without argument and no return type

Enter your choice (0 to exit): 0

Exiting program.
```