* **GCC**

It stands for **GNU Compiler Collection.** It is a set of compilers and development tools for for various operating systems including Linux, Windows and BSDs. It is primarily used for compiling C & C++ programs.

* **GNU**

It stands for **GNU’s Not Unix** and it is pronounced as **“g-noo”**. It is an extensive collection of free software which can be used as an operating system or can be used in parts with other operating system.

* **Compilation**

|  |  |  |
| --- | --- | --- |
| **COMMANDS** | **OUTPUTS** | **STEPS** |
|  | Sample.c |  |
| "gcc -E Sample.c” then |  | Pre-Processor |
| “gcc -E Sample.c > Sample.I” |  |  |
|  | Sample.i |  |
| “gcc -S Sample.i” |  | Compiler |
|  | Sample.s |  |
| “gcc -c Sample.s” |  | Assembler |
|  | Sample.o |  |
| “gcc -O Sample.o” |  | Linker |
|  | a.out |  |
| “gcc -o Sample\_out Sample.c” |  | Creates an output file name “Sample\_out” for Sample.c |
|  | Sample\_out |  |

Sample.i --> It is preprocessor output file

Sample.s --> It is an assembly language file

Sample.o --> It is an object file that is created when the c program is compiled by a compiler

a.out --> It is a file that contains the executable code

* **GDB**

It stands for **GNU Project Debugger** and it is a powerful debugger for C.

GDB allows you to run the program upto a certain point, then stop and print out the values of certain variables at that point, or step through the program one line at a time and print out the values for each variable after executing each line.

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATION** | **SHORT FORMS** | **COMMANDS** |  |
|  |  | gcc -g <c\_filename> -o <output\_filename> | This will compile the c file and will tell the compiler that the executionable file that is being generated should contain the debugging informations. |
|  |  | gdb <output\_filename> | This will open the output file in debugging mode. |
| breakpoint | b | b <line\_no/function \_name> | Puts breakpoints on the line no. or the function name |
| list | l |  | Shows a lines of source code in the debugger. |
| frame | f |  | Shows the current(line) frame you are in. |
| next | n |  | Moves to the next line. |
| step | s |  |  |
| backtrace | b |  | Backtraces to the steps we have come through |
| print | p |  | Print values of some variables at that point of time |
| info | i |  |  |

* **GCOV**

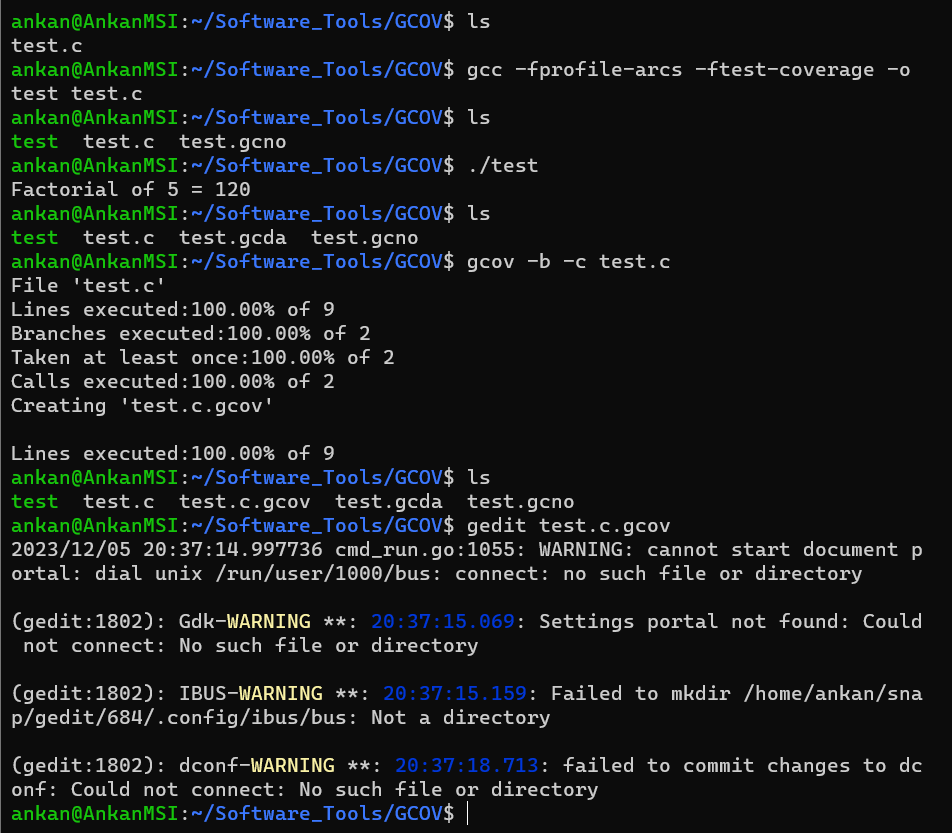
GCOV is a source code coverage analysis tool that generates counts of how many times each statement in a program is executed. It is a standard utility in the GNU Compiler Collection (GCC) Suite.

Uses of gcov: -

1. It is used to analyze programs to create faster and more efficient codes.
2. Discover untested parts of the program.
3. Identify where optimization efforts will have the most impact on the code.

**Command: -**

1. “gcc -fprofile-arcs -ftest-coverage -o <filename> <filename.c>” = This tells the compiler to generate additional information needed by the gcov(basically a flow graph of the program) and also include additional code in the object file for generating the extra profiling information needed by gcov. It creates a output file in the name of <filename>.
2. “run the output file”
3. “gcov -b -c <filename.c>” = This show info about the **lines executed, branches executed, no.of lines taken atleast once, calls executed and creates a <filename>.c.gcov file.**



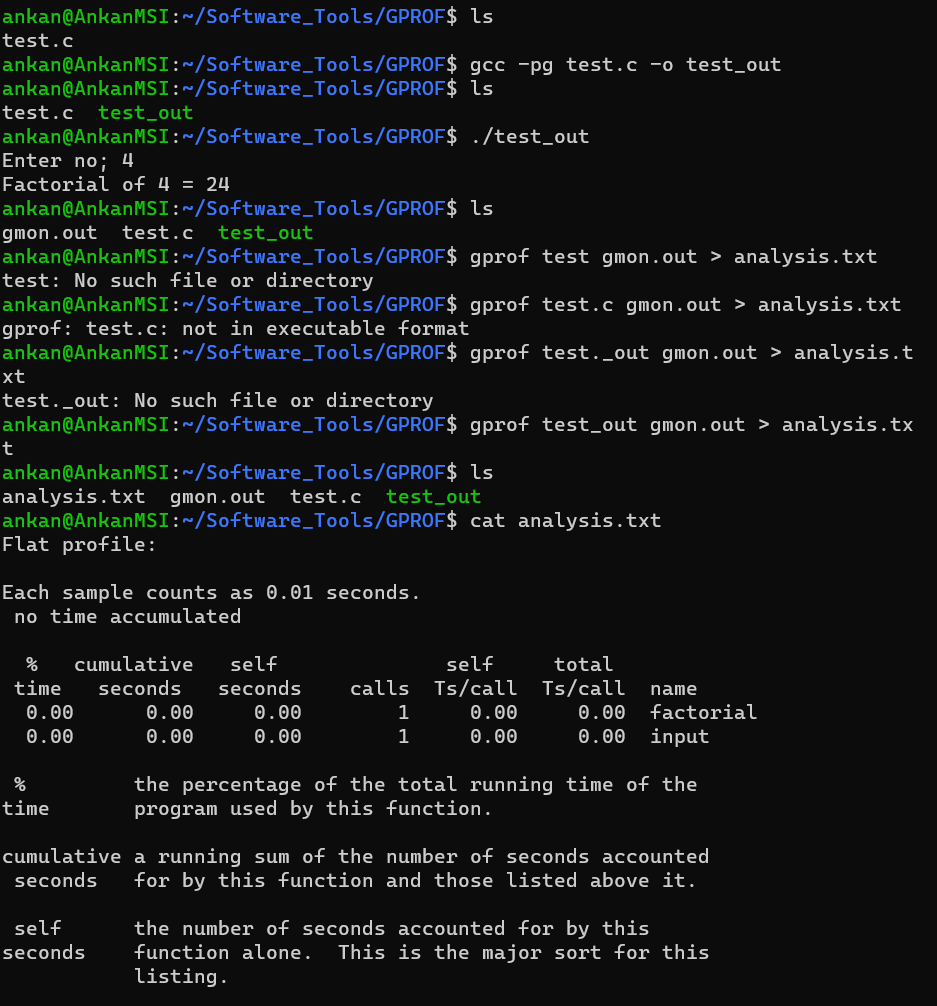
* **GPROF**

It allows you to learn which function taking how much time

Calling or called by relation among functions.

It can show you frequency of function calls.

These may help you to spot unnoticed bugs in the code.



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 factorial

0.00 0.00 0.00 1 0.00 0.00 input

% the percentage of the total running time of the

time program used by this function.

cumulative a running sum of the number of seconds accounted

seconds for by this function and those listed above it.

self the number of seconds accounted for by this

seconds function alone. This is the major sort for this

listing.

calls the number of times this function was invoked, if

this function is profiled, else blank.

self the average number of milliseconds spent in this

ms/call function per call, if this function is profiled,

else blank.

total the average number of milliseconds spent in this

ms/call function and its descendents per call, if this

function is profiled, else blank.

name the name of the function. This is the minor sort

for this listing. The index shows the location of

the function in the gprof listing. If the index is

in parenthesis it shows where it would appear in

the gprof listing if it were to be printed.

Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,

are permitted in any medium without royalty provided the copyright

notice and this notice are preserved.

Call graph (explanation follows)

granularity: each sample hit covers 4 byte(s) no time propagated

index % time self children called name

0.00 0.00 1/1 main [8]

[1] 0.0 0.00 0.00 1 factorial [1]

-----------------------------------------------

0.00 0.00 1/1 main [8]

[2] 0.0 0.00 0.00 1 input [2]

-----------------------------------------------

This table describes the call tree of the program, and was sorted by

the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the

index number at the left hand margin lists the current function.

The lines above it list the functions that called this function,

and the lines below it list the functions this one called.

This line lists:

index A unique number given to each element of the table.

Index numbers are sorted numerically.

The index number is printed next to every function name so

it is easier to look up where the function is in the table.

% time This is the percentage of the `total' time that was spent

in this function and its children. Note that due to

different viewpoints, functions excluded by options, etc,

these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this

function by its children.

called This is the number of times the function was called.

If the function called itself recursively, the number

only includes non-recursive calls, and is followed by

a `+' and the number of recursive calls.

name The name of the current function. The index number is

printed after it. If the function is a member of a

cycle, the cycle number is printed between the

function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly

from the function into this parent.

children This is the amount of time that was propagated from

the function's children into this parent.

called This is the number of times this parent called the

function `/' the total number of times the function

was called. Recursive calls to the function are not

included in the number after the `/'.

name This is the name of the parent. The parent's index

number is printed after it. If the parent is a

member of a cycle, the cycle number is printed between

the name and the index number.

If the parents of the function cannot be determined, the word

`<spontaneous>' is printed in the `name' field, and all the other

fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly

from the child into the function.

children This is the amount of time that was propagated from the

child's children to the function.

called This is the number of times the function called

this child `/' the total number of times the child

was called. Recursive calls by the child are not

listed in the number after the `/'.

name This is the name of the child. The child's index

number is printed after it. If the child is a

member of a cycle, the cycle number is printed

between the name and the index number.

If there are any cycles (circles) in the call graph, there is an

entry for the cycle-as-a-whole. This entry shows who called the

cycle (as parents) and the members of the cycle (as children.)

The `+' recursive calls entry shows the number of function calls that

were internal to the cycle, and the calls entry for each member shows,

for that member, how many times it was called from other members of

the cycle.

Copyright (C) 2012-2022 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,

are permitted in any medium without royalty provided the copyright

notice and this notice are preserved.

Index by function name

[1] factorial [2] input