1)Normal Execution of Gcov on the given code

-: 0:Source:que2.c

-: 0:Graph:que2.gcno

-: 0:Data:que2.gcda

-: 0:Runs:1

-: 1:#include <stdio.h>

-: 2:

1: 3:int main (void)

-: 4:{

-: 5:

-: 6: int i, total, num;

-: 7:

1: 8: total = 0;

-: 9:

11: 10: for (i = 0; i < 10; i++)

10: 11: total += i;

-: 12:

1: 13: if (total != 45)

#####: 14: printf ("Failure\n");

-: 15: else

1: 16: printf ("Success\n");

-: 17:

1: 18: num = 0;

1: 19: switch(num) {

#####: 20: case 1: printf("1");

#####: 21: break;

-: 22:

#####: 23: case 2: printf("2");

#####: 24: break;

-: 25:

#####: 26: case 3: printf("3");

#####: 27: break;

-: 28:

#####: 29: case 4: printf("4");

#####: 30: break;

-: 31:

1: 32: default: printf("0");

1: 33: break;

-: 34:

-: 35: }

1: 36: return 0;

-: 37:}

Output -

Success

0

2)Block with numbering and their counts

-: 0:Source:que2.c

-: 0:Graph:que2.gcno

-: 0:Data:que2.gcda

-: 0:Runs:3

-: 1:#include <stdio.h>

-: 2:

3: 3:int main (void)

-: 4:{

-: 5:

-: 6: int i, total, num;

-: 7:

3: 8: total = 0;

-: 9:

33: 10: for (i = 0; i < 10; i++)

3: 10-block 0

33: 10-block 1

30: 11: total += i;

30: 11-block 0

-: 12:

3: 13: if (total != 45)

3: 13-block 0

#####: 14: printf ("Failure\n");

%%%%%: 14-block 0

-: 15: else

3: 16: printf ("Success\n");

3: 16-block 0

-: 17:

3: 18: num = 0;

3: 19: switch(num) {

3: 19-block 0

#####: 20: case 1: printf("1");

%%%%%: 20-block 0

#####: 21: break;

-: 22:

#####: 23: case 2: printf("2");

%%%%%: 23-block 0

#####: 24: break;

-: 25:

#####: 26: case 3: printf("3");

%%%%%: 26-block 0

#####: 27: break;

-: 28:

#####: 29: case 4: printf("4");

%%%%%: 29-block 0

#####: 30: break;

-: 31:

3: 32: default: printf("0");

3: 32-block 0

3: 33: break;

-: 34:

-: 35: }

3: 36: return 0;

3: 36-block 0

-: 37:}

Inference - Baised on the analysis, it can be ststed that the code the first for loop runs and the else part of the another block is run. Printing success in the screen, in the result mentioned above, the program spends most of the time and computation in the loop and optimizing the program can be done by optimization of loop.Further if you run gprof on it we can see it runs within 0.01 sec as sampling is not done.

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

% 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) 2012-2020 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.