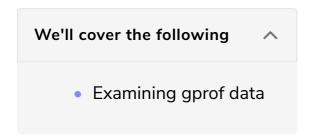
Profiling your code using `gprof`

Another tool available in C is 'gprof'. We'll see its functionality and how it helps us understand the program runtime.



There is a unix utility program called **gprof** (GNU profiler) that can help you determine which parts of your program are taking most execution time.

The basic steps are:

- 1. compile your program with profiling enabled (using the -pg compiler flag)
- 2. execute your program once to generate a profile data file
- 3. run gprof to analyse the profile data

Here is an example program that we wish to profile:

```
#include <stdio.h>
#include <math.h>
#define MAXLOOP 1e7
double myFun1(double x) {
  double a = \sin(x);
  return a;
}
double myFun2(double x) {
  double a = pow(x,3);
  return a;
}
double myFun3(double x) {
 double a = sqrt(x);
  return a;
}
int main(int argc, char *argv[]) {
  int i;
  double x;
  double xsum = 0.0;
  for (i=1; i<MAXLOOP; i++) {</pre>
    x = mvFun1(i) + mvFun2(i) + mvFun3(i);
```

```
xsum += x;
}
printf("xsum = %.6f\n", xsum);
return 0;
}
```





Now let's recompile the program for the profiler, execute it once, and then run gprof to look at the profile data:

```
plg@wildebeest:~/Desktop$ gcc -o go go.c -lm -pg
plg@wildebeest:~/Desktop$ ./go
xsum = 2499999499999934350004060160.000000
plg@wildebeest:~/Desktop$ gprof go -p
Flat profile:
Each sample counts as 0.01 seconds.
     cumulative self
                                   self
                                           total
time seconds seconds calls ns/call ns/call name
40.35
         0.12
                  0.12 9999999 12.11 12.11 myFun3
         0.24
40.35
                0.12
                                                   main
         0.28
                  0.04 9999999
                                    4.04
                                            4.04 myFun2
13.45
         0.29
                0.01 9999999
 3.36
                                     1.01
                                             1.01 myFun1
 3.36
         0.30
                  0.01
                                                   frame_dummy
%
          the percentage of the total running time of the
          program used by this function.
time
cumulative a running sum of the number of seconds accounted
         for by this function and those listed above it.
 seconds
          the number of seconds accounted for by this
self
          function alone. This is the major sort for this
seconds
          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
          function per call, if this function is profiled,
ms/call
      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.
          the name of the function. This is the minor sort
name
```

```
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.
```

Examining gprof data

The very first section of the <code>gprof</code> output gives us important information. It's telling us that we are spending 40.35% of our time in <code>myFun3()</code> compared with only 13.45% and 3.36% in <code>myFun2()</code> and <code>myFun3()</code>. The obvious thing to try to do now is to speed up <code>myFun3()</code>.

Even with all these utilities, there are some methods in C which always slow your program down. We'll discuss these in the next lesson.