C/C++ profiler

Dr. Yihsiang Liow (February 13, 2024)

Contents

1 Profilers

In software engineering, profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization, and more specifically, performance engineering. - *Wikipedia*

2 gprof

To install, as root, do

```
dnf install -y binutils
```

Create directory `gprof/` and in this directory write

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <vector>

void randvector(std::vector< int > & v)
{
    for (auto && e: v) e = rand();
}

void swap(int & a, int & b)
{
    int t = a;
    a = b;
    b = t;
}

void bubblesort(std::vector< int > & v)
{
    int n = v.size();
    for (int i = n - 2; i >= 0; --i)
    {
        for (int j = 0; j <= i; ++j)
        {
            if (v[j] > v[j + 1])
            {
                swap(v[j], v[j + 1]);
            }
        }
    }
}

int main()
{
    srand((unsigned int) time(NULL));
    std::vector< int > v(10000);
    randvector(v);
    bubblesort(v);
    return 0;
}
```

Compile and link using g++ but remember to include -pg for both:

```
g++ main.cpp -pg -c -o main.o
g++ -pg main.o -o main.exe
```

or just

```
g++ main.cpp -pg -o main.exe
```

To run gprof, do

```
gprof main.exe gmon.out > gprof.txt
```

You will get the file grof.txt which looks something like this (truncated on the right):

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 53.41      0.17     0.17 150436406     0.00     0.00  std::vector<int, std::allocator<int
 22.66      0.24     0.07        1    70.24   291.01  bubblesort(std::vector<int, std::all
 12.95      0.28     0.04 25223203     0.00     0.00  swap(int&, int&)
  6.47      0.30     0.02        1    20.07    20.07  randvector(std::vector<int, std::all
  4.86      0.31     0.02        1    15.05    15.05  std::vector<int, std::allocator<int>
  0.00      0.31     0.00    20002     0.00     0.00  __gnu_cxx::__normal_iterator<int*, s
  0.00      0.31     0.00    10001     0.00     0.00  bool __gnu_cxx::operator!=<int*, std
  0.00      0.31     0.00    10000     0.00     0.00  __gnu_cxx::__normal_iterator<int*, s
  0.00      0.31     0.00    10000     0.00     0.00  __gnu_cxx::__normal_iterator<int*, s
  0.00      0.31     0.00        3     0.00     0.00  __gnu_cxx::new_allocator<int>::~new_
  0.00      0.31     0.00        3     0.00     0.00  std::allocator<int>::~allocator()
  0.00      0.31     0.00        2     0.00     0.00  __gnu_cxx::new_allocator<int>::new_a
  0.00      0.31     0.00        2     0.00     0.00  __gnu_cxx::__normal_iterator<int*, s
  0.00      0.31     0.00        2     0.00     0.00  __gnu_cxx::new_allocator<int>::max_s
  0.00      0.31     0.00        2     0.00     0.00  std::allocator<int>::allocator(std::
  0.00      0.31     0.00        2     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z10randvectorRSt6vec
  0.00      0.31     0.00        1     0.00     0.00  __static_initialization_and_destruct
  0.00      0.31     0.00        1     0.00     0.00  __gnu_cxx::new_allocator<int>::deall
  0.00      0.31     0.00        1     0.00     0.00  __gnu_cxx::new_allocator<int>::alloc
  0.00      0.31     0.00        1     0.00     0.00  __gnu_cxx::new_allocator<int>::new_a
  0.00      0.31     0.00        1     0.00     0.00  std::allocator<int>::allocator()
  0.00      0.31     0.00        1     0.00     0.00  void std::_Destroy_aux<true>::__dest
  0.00      0.31     0.00        1     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  std::_Vector_base<int, std::allocato
  0.00      0.31     0.00        1     0.00     0.00  std::allocator_traits<std::allocator
  0.00      0.31     0.00        1     0.00     0.00  std::allocator_traits<std::allocator
  0.00      0.31     0.00        1     0.00     0.00  std::allocator_traits<std::allocator
  0.00      0.31     0.00        1     0.00     0.00  int* std::__uninitialized_default_n_
  0.00      0.31     0.00        1     0.00     0.00  std::vector<int, std::allocator<int>
  0.00      0.31     0.00        1     0.00     0.00  std::vector<int, std::allocator<int>
  0.00      0.31     0.00        1     0.00     0.00  std::vector<int, std::allocator<int>
  0.00      0.31     0.00        1     0.00     0.00  std::vector<int, std::allocator<int>
```

```
  0.00      0.31     0.00        1     0.00     0.00  std::vector<int, std::allocator<int>
  0.00      0.31     0.00        1     0.00     0.00  std::vector<int, std::allocator<int>
  0.00      0.31     0.00        1     0.00     0.00  std::vector<int, std::allocator<int>
  0.00      0.31     0.00        1     0.00     0.00  __gnu_cxx::__enable_if<std::__is_sca
  0.00      0.31     0.00        1     0.00     0.00  int* std::__niter_base<int*>(int*)
  0.00      0.31     0.00        1     0.00     0.00  int* std::__niter_wrap<int*>(int* co
  0.00      0.31     0.00        1     0.00     0.00  int* std::__uninitialized_default_n<
  0.00      0.31     0.00        1     0.00     0.00  int* std::__uninitialized_default_n_
  0.00      0.31     0.00        1     0.00     0.00  unsigned long const& std::min<unsign
  0.00      0.31     0.00        1     0.00     0.00  int* std::fill_n<int*, unsigned long
  0.00      0.31     0.00        1     0.00     0.00  void std::_Destroy<int*>(int*, int*)
  0.00      0.31     0.00        1     0.00     0.00  void std::_Destroy<int*, int>(int*,

 %         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) 2012-2019 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 2 byte(s) for 3.21% of 0.31 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.31                 main [1]
                0.07    0.22       1/1           bubblesort(std::vector<int, std::allocato
                0.02    0.00       1/1           randvector(std::vector<int, std::allocato
                0.00    0.00       1/1           std::allocator<int>::allocator() [29]
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::v
                0.00    0.00       1/3           std::allocator<int>::~allocator() [18]
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::~
-----------------------------------------------
                0.07    0.22       1/1           main [1]
[2]     93.5    0.07    0.22       1         bubblesort(std::vector<int, std::allocator<in
                0.17    0.00 150436406/150436406     std::vector<int, std::allocator<int>
```

```
                0.04    0.00 25223203/25223203     swap(int&, int&) [4]
                0.02    0.00       1/1           std::vector<int, std::allocator<int> >::s
-----------------------------------------------
                0.17    0.00 150436406/150436406     bubblesort(std::vector<int, std::allo
[3]     53.2    0.17    0.00 150436406         std::vector<int, std::allocator<int> >::ope
-----------------------------------------------
                0.04    0.00 25223203/25223203     bubblesort(std::vector<int, std::alloca
[4]     12.9    0.04    0.00 25223203         swap(int&, int&) [4]
-----------------------------------------------
                0.02    0.00       1/1           main [1]
[5]      6.5    0.02    0.00       1         randvector(std::vector<int, std::allocator<in
                0.00    0.00   10001/10001       bool __gnu_cxx::operator!=<int*, std::vec
                0.00    0.00   10000/10000       __gnu_cxx::__normal_iterator<int*, std::v
                0.00    0.00   10000/10000       __gnu_cxx::__normal_iterator<int*, std::v
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::b
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::e
-----------------------------------------------
                0.02    0.00       1/1           bubblesort(std::vector<int, std::allocato
[6]      4.8    0.02    0.00       1         std::vector<int, std::allocator<int> >::size(
-----------------------------------------------
                0.00    0.00   20002/20002       bool __gnu_cxx::operator!=<int*, std::vec
[13]     0.0    0.00    0.00   20002         __gnu_cxx::__normal_iterator<int*, std::vecto
-----------------------------------------------
                0.00    0.00   10001/10001       randvector(std::vector<int, std::allocato
[14]     0.0    0.00    0.00   10001         bool __gnu_cxx::operator!=<int*, std::vector<
                0.00    0.00   20002/20002       __gnu_cxx::__normal_iterator<int*, std::v
-----------------------------------------------
                0.00    0.00   10000/10000       randvector(std::vector<int, std::allocato
[15]     0.0    0.00    0.00   10000         __gnu_cxx::__normal_iterator<int*, std::vecto
-----------------------------------------------
                0.00    0.00   10000/10000       randvector(std::vector<int, std::allocato
[16]     0.0    0.00    0.00   10000         __gnu_cxx::__normal_iterator<int*, std::vecto
-----------------------------------------------
                0.00    0.00       3/3           std::allocator<int>::~allocator() [18]
[17]     0.0    0.00    0.00       3         __gnu_cxx::new_allocator<int>::~new_allocator
-----------------------------------------------
                0.00    0.00       1/3           main [1]
                0.00    0.00       1/3           std::vector<int, std::allocator<int> >::_
                0.00    0.00       1/3           std::_Vector_base<int, std::allocator<int
[18]     0.0    0.00    0.00       3         std::allocator<int>::~allocator() [18]
                0.00    0.00       3/3           __gnu_cxx::new_allocator<int>::~new_alloc
-----------------------------------------------
                0.00    0.00       2/2           std::allocator<int>::allocator(std::alloc
[19]     0.0    0.00    0.00       2         __gnu_cxx::new_allocator<int>::new_allocator(
-----------------------------------------------
                0.00    0.00       1/2           std::vector<int, std::allocator<int> >::b
                0.00    0.00       1/2           std::vector<int, std::allocator<int> >::e
[20]     0.0    0.00    0.00       2         __gnu_cxx::__normal_iterator<int*, std::vecto
-----------------------------------------------
                0.00    0.00       1/2           std::allocator_traits<std::allocator<int>
                0.00    0.00       1/2           __gnu_cxx::new_allocator<int>::allocate(u
[21]     0.0    0.00    0.00       2         __gnu_cxx::new_allocator<int>::max_size() con
-----------------------------------------------
                0.00    0.00       1/2           std::vector<int, std::allocator<int> >::_
                0.00    0.00       1/2           std::_Vector_base<int, std::allocator<int
[22]     0.0    0.00    0.00       2         std::allocator<int>::allocator(std::allocator
                0.00    0.00       2/2           __gnu_cxx::new_allocator<int>::new_alloca
-----------------------------------------------
                0.00    0.00       1/2           std::vector<int, std::allocator<int> >::~
                0.00    0.00       1/2           std::vector<int, std::allocator<int> >::_
[23]     0.0    0.00    0.00       2         std::_Vector_base<int, std::allocator<int> >:
-----------------------------------------------
                0.00    0.00       1/1           __libc_csu_init [64]
[24]     0.0    0.00    0.00       1         _GLOBAL__sub_I__Z10randvectorRSt6vectorIiSaIi
```

```
                0.00    0.00       1/1           __static_initialization_and_destruction_0
-----------------------------------------------
                0.00    0.00       1/1           _GLOBAL__sub_I__Z10randvectorRSt6vectorIi
[25]     0.0    0.00    0.00       1         __static_initialization_and_destruction_0(int
-----------------------------------------------
                0.00    0.00       1/1           std::allocator_traits<std::allocator<int>
[26]     0.0    0.00    0.00       1         __gnu_cxx::new_allocator<int>::deallocate(int
-----------------------------------------------
                0.00    0.00       1/1           std::allocator_traits<std::allocator<int>
[27]     0.0    0.00    0.00       1         __gnu_cxx::new_allocator<int>::allocate(unsig
                0.00    0.00       1/2           __gnu_cxx::new_allocator<int>::max_size()
-----------------------------------------------
                0.00    0.00       1/1           std::allocator<int>::allocator() [29]
[28]     0.0    0.00    0.00       1         __gnu_cxx::new_allocator<int>::new_allocator(
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[29]     0.0    0.00    0.00       1         std::allocator<int>::allocator() [29]
                0.00    0.00       1/1           __gnu_cxx::new_allocator<int>::new_alloca
-----------------------------------------------
                0.00    0.00       1/1           void std::_Destroy<int*>(int*, int*) [57]
[30]     0.0    0.00    0.00       1         void std::_Destroy_aux<true>::__destroy<int*>
-----------------------------------------------
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
[31]     0.0    0.00    0.00       1         std::_Vector_base<int, std::allocator<int> >:
                0.00    0.00       1/1           std::allocator_traits<std::allocator<int>
-----------------------------------------------
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
[32]     0.0    0.00    0.00       1         std::_Vector_base<int, std::allocator<int> >:
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
                0.00    0.00       1/2           std::allocator<int>::allocator(std::alloc
-----------------------------------------------
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
[33]     0.0    0.00    0.00       1         std::_Vector_base<int, std::allocator<int> >:
                0.00    0.00       1/3           std::allocator<int>::~allocator() [18]
-----------------------------------------------
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
[34]     0.0    0.00    0.00       1         std::_Vector_base<int, std::allocator<int> >:
                0.00    0.00       1/1           std::allocator_traits<std::allocator<int>
-----------------------------------------------
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
[35]     0.0    0.00    0.00       1         std::_Vector_base<int, std::allocator<int> >:
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
-----------------------------------------------
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
[36]     0.0    0.00    0.00       1         std::_Vector_base<int, std::allocator<int> >:
-----------------------------------------------
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::v
[37]     0.0    0.00    0.00       1         std::_Vector_base<int, std::allocator<int> >:
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
-----------------------------------------------
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::~
[38]     0.0    0.00    0.00       1         std::_Vector_base<int, std::allocator<int> >:
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
-----------------------------------------------
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
[39]     0.0    0.00    0.00       1         std::allocator_traits<std::allocator<int> >::
                0.00    0.00       1/1           __gnu_cxx::new_allocator<int>::deallocate
-----------------------------------------------
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
[40]     0.0    0.00    0.00       1         std::allocator_traits<std::allocator<int> >::
                0.00    0.00       1/1           __gnu_cxx::new_allocator<int>::allocate(u
-----------------------------------------------
```

```
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::_
[41]     0.0    0.00    0.00       1         std::allocator_traits<std::allocator<int> >::
                0.00    0.00       1/2           __gnu_cxx::new_allocator<int>::max_size()
-----------------------------------------------
                0.00    0.00       1/1           int* std::__uninitialized_default_n<int*,
[42]     0.0    0.00    0.00       1         int* std::__uninitialized_default_n_1<true>::
                0.00    0.00       1/1           int* std::fill_n<int*, unsigned long, int
-----------------------------------------------
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::_
[43]     0.0    0.00    0.00       1         std::vector<int, std::allocator<int> >::_S_ma
                0.00    0.00       1/1           std::allocator_traits<std::allocator<int>
                0.00    0.00       1/1           unsigned long const& std::min<unsigned lo
-----------------------------------------------
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::v
[44]     0.0    0.00    0.00       1         std::vector<int, std::allocator<int> >::_S_ch
                0.00    0.00       1/2           std::allocator<int>::allocator(std::alloc
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::_
                0.00    0.00       1/3           std::allocator<int>::~allocator() [18]
-----------------------------------------------
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::v
[45]     0.0    0.00    0.00       1         std::vector<int, std::allocator<int> >::_M_de
                0.00    0.00       1/2           std::_Vector_base<int, std::allocator<int
                0.00    0.00       1/1           int* std::__uninitialized_default_n_a<int
-----------------------------------------------
                0.00    0.00       1/1           randvector(std::vector<int, std::allocato
[46]     0.0    0.00    0.00       1         std::vector<int, std::allocator<int> >::end()
                0.00    0.00       1/2           __gnu_cxx::__normal_iterator<int*, std::v
-----------------------------------------------
                0.00    0.00       1/1           randvector(std::vector<int, std::allocato
[47]     0.0    0.00    0.00       1         std::vector<int, std::allocator<int> >::begin
                0.00    0.00       1/2           __gnu_cxx::__normal_iterator<int*, std::v
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[48]     0.0    0.00    0.00       1         std::vector<int, std::allocator<int> >::vecto
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::_
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::_
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[49]     0.0    0.00    0.00       1         std::vector<int, std::allocator<int> >::~vect
                0.00    0.00       1/2           std::_Vector_base<int, std::allocator<int
                0.00    0.00       1/1           void std::_Destroy<int*, int>(int*, int*,
                0.00    0.00       1/1           std::_Vector_base<int, std::allocator<int
-----------------------------------------------
                0.00    0.00       1/1           int* std::fill_n<int*, unsigned long, int
[50]     0.0    0.00    0.00       1         __gnu_cxx::__enable_if<std::__is_scalar<int>:
-----------------------------------------------
                0.00    0.00       1/1           int* std::fill_n<int*, unsigned long, int
[51]     0.0    0.00    0.00       1         int* std::__niter_base<int*>(int*) [51]
-----------------------------------------------
                0.00    0.00       1/1           int* std::fill_n<int*, unsigned long, int
[52]     0.0    0.00    0.00       1         int* std::__niter_wrap<int*>(int* const&, int
-----------------------------------------------
                0.00    0.00       1/1           int* std::__uninitialized_default_n_a<int
[53]     0.0    0.00    0.00       1         int* std::__uninitialized_default_n<int*, uns
                0.00    0.00       1/1           int* std::__uninitialized_default_n_1<tru
-----------------------------------------------
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::_
[54]     0.0    0.00    0.00       1         int* std::__uninitialized_default_n_a<int*, u
                0.00    0.00       1/1           int* std::__uninitialized_default_n<int*,
-----------------------------------------------
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::_
[55]     0.0    0.00    0.00       1         unsigned long const& std::min<unsigned long>(
-----------------------------------------------
```

```
                0.00    0.00       1/1           int* std::__uninitialized_default_n_1<tru
[56]     0.0    0.00    0.00       1         int* std::fill_n<int*, unsigned long, int>(in
                0.00    0.00       1/1           int* std::__niter_base<int*>(int*) [51]
                0.00    0.00       1/1           __gnu_cxx::__enable_if<std::__is_scalar<i
                0.00    0.00       1/1           int* std::__niter_wrap<int*>(int* const&,
-----------------------------------------------
                0.00    0.00       1/1           void std::_Destroy<int*, int>(int*, int*,
[57]     0.0    0.00    0.00       1         void std::_Destroy<int*>(int*, int*) [57]
                0.00    0.00       1/1           void std::_Destroy_aux<true>::__destroy<i
-----------------------------------------------
                0.00    0.00       1/1           std::vector<int, std::allocator<int> >::~
[58]     0.0    0.00    0.00       1         void std::_Destroy<int*, int>(int*, int*, std
                0.00    0.00       1/1           void std::_Destroy<int*>(int*, int*) [57]
-----------------------------------------------

 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) 2012-2019 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

  [24] _GLOBAL__sub_I__Z10randvectorRSt6vectorIiSaIiEE [22] std::allocator<int>::allocator
   [2] bubblesort(std::vector<int, std::allocator<int> >&) [29] std::allocator<int>::alloc
   [5] randvector(std::vector<int, std::allocator<int> >&) [18] std::allocator<int>::~allo
  [25] __static_initialization_and_destruction_0(int, int) [30] void std::_Destroy_aux<tru
   [4] swap(int&, int&)       [31] std::_Vector_base<int, std::allocator<int> >::_M_alloca
  [26] __gnu_cxx::new_allocator<int>::deallocate(int*, unsigned long) [32] std::_Vector_ba
  [27] __gnu_cxx::new_allocator<int>::allocate(unsigned long, void const*) [33] std::_Vect
  [28] __gnu_cxx::new_allocator<int>::new_allocator() [34] std::_Vector_base<int, std::all
  [19] __gnu_cxx::new_allocator<int>::new_allocator(__gnu_cxx::new_allocator<int> const&)
  [17] __gnu_cxx::new_allocator<int>::~new_allocator() [36] std::_Vector_base<int, std::al
  [20] __gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::__norm
  [15] __gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::operat
  [14] bool __gnu_cxx::operator!=<int*, std::vector<int, std::allocator<int> > >(__gnu_cxx
  [21] __gnu_cxx::new_allocator<int>::max_size() const [39] std::allocator_traits<std::all
  [13] __gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::base()
  [16] __gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::operat
   [6] std::vector<int, std::allocator<int> >::size() const [42] int* std::__uninitialized
```

The report is made up of the "flat profile" and the "call graph".

The "flat profile" is very easy to read. Explanation is at the end of this section.

The "call graph" part of the report allows you to see the function call graph. The call graph part is made up of sections. Each section looks like this:

```
index % time    self  children    called
                                        f
                                        g
[5]                                 h
                                        i
                                        j
                                        k
```

This is the section for function `h` which is given an index of 5. `h` is called by `f` and `g`, and `h` calls `i`, `j`, `k`. Under the `called` column, if you see

```
index % time    self  children    called
[5]                               5         h
```

this means `h` is called 5 times. If you see

```
index % time    self  children    called
[5]                               1+4       h
```

it means `h` is recursive and there is 1 non-recursive call to itself and there are 4 recursive calls. If you see

```
index % time    self  children    called
                                  2/5             g
[5]                                       h
```

this means `g` calls `h` 2 times out of a total of 5 functions calls that `g` makes. If you see

```
index % time    self  children    called
[5]                                       h
                                  10/20           i
```

this means `h` calls `i` 10 times out of a total of 20 times `i` is called.

Warning: If the total runtime is very small, the report might be empty.

Exercise 2.1.

(a) There's a `std::swap` function from C++ STL. Run `gprof` on your program after changing your `swap` to `std::swap` in the bubblesorting progam. Which swap function is faster, `swap` or `std::swap`?
(b) Write another version of the bubblesort program using arrays instead of `std::vector`. Generate a new `gprof` report. Which version is faster?

Exercise 2.2. Write a program where `main()` calls a recursive function `f()`. To make sure `f()` actually takes some time to execute, insert some redundant

time intensive loop inside `f()` (otherwise the total time taken is too small and no `gprof` report is generated). Generate a `gprof` report and make sure you can read it. □

Exercise 2.3. Write a quicksort program, writing as many functions as you can. You should of course have a function to select the pivot (use median-of-three) and a function to perform partitioning. Generate a `gprof` report and read it carefully.

If you want to draw a graph representation of the function call information in a `gprof` report, you can use gprof2dot which uses `graphviz` (see CISS350, etc.) To install `grof2dot`, as root, do

```
dnf install -y gprof2dot
```

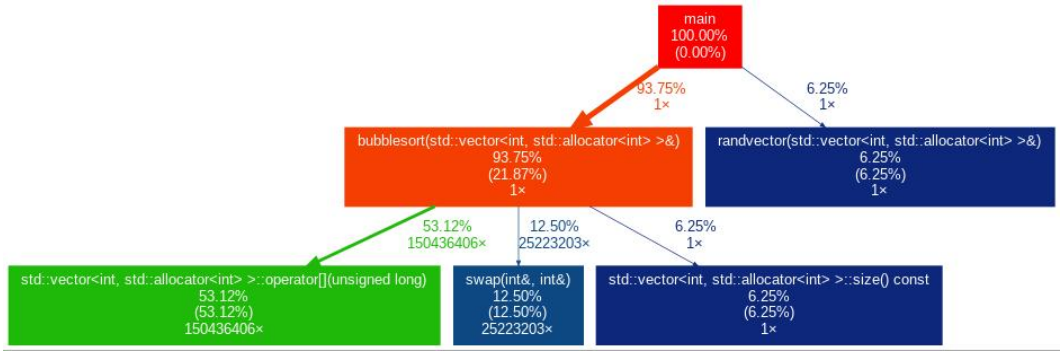
Exit root. To generate a dot file do

```
gprof2dot gprof.txt > gprof.dot
```

You'll get a `gprof.dot` file. Next generate a jpg file:

```
dot -Tjpg gprof.dot -o gprof.jpg
```

You'll get `gprof.jpg`:



Exercise 2.4. Generate function call graph diagrams of your earlier sorting programs, one for the program that uses arrays and another one that uses `std::vector`. □

3 perf

To install perf, as root, do

```
dnf install -y perf
```

Create a directory `perf/` and write the following file in `perf/`:

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <vector>

void randvector(std::vector< int > & v)
{
    for (auto && e: v) e = rand();
}

void swap(int & a, int & b)
{
    int t = a;
    a = b;
    b = t;
}

void bubblesort(std::vector< int > & v)
{
    int n = v.size();
    for (int i = n - 2; i >= 0; --i)
    {
        for (int j = 0; j <= i; ++j)
        {
            if (v[j] > v[j + 1])
            {
                swap(v[j], v[j + 1]);
            }
        }
    }
}

int main()
{
    srand((unsigned int) time(NULL));
    std::vector< int > v(10000);
    randvector(v);
    bubblesort(v);
    return 0;
}
```

Compile it using g++. Then run perf:

```
perf record ./main.exe
```

You might get an output something like this:

```
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.027 MB perf.data (457 samples) ]
```

You'll also get the file `perf.data`. Run

```
perf report --stdio > perf.txt
```

Here's my `perf.txt`:

```
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 94K of event 'cpu-clock:uhpppH'
# Event count (approx.): 23697500000
#
# Overhead  Command   Shared Object        Symbol
# ........  ........  ...................  ..................................................
#
    50.49%  main.exe  main.exe             [.] bubblesort
    33.00%  main.exe  main.exe             [.] std::vector<int, std::allocator<int> >::operator[]
    16.51%  main.exe  main.exe             [.] swap
     0.00%  main.exe  ld-2.30.so           [.] _dl_relocate_object
     0.00%  main.exe  ld-2.30.so           [.] do_lookup_x
     0.00%  main.exe  libc-2.30.so         [.] __random_r
     0.00%  main.exe  libstdc++.so.6.0.28  [.] _GLOBAL__sub_I_compatibility_c__0x.cc
     0.00%  main.exe  libstdc++.so.6.0.28  [.] 0x000000000009c6b0


#
# (Tip: Customize output of perf script with: perf script -F event,ip,sym)
#
```