Профилирование - сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш-промахов и т.д. Инструмент, используемый для анализа работы, называют профилировщиком или профайлером (англ. profiler). [wiki]

С помощью профилирования вы можете узнать, сколько времени ваша программа тратит на выполнение каждой из своих функций, а также в каком порядке и как часто эти функции вызывают друг друга. Проанализировав эту информацию, можно понять, что какая-то часть программы работает медленнее, чем вы ожидали, или что какая-то из функций вызывается чаще или режем, чем вы предполагали.

Чтобы выполнить профилирование необходимо

1. Собрать программу с включенным профилированием.
2. Выполнить программу, чтобы получить данные профилирования.
3. Запустить утилиту *gprof* для анализа этих данных.

Сборка программы для профилирования

Для сборки программы с профилированием необходимо указать ключ “-pg”

```
gcc -std=c99 -Wall -Werror -g -pg bubble.c -o bubble.exe
```

Встретив ключ “-pg”, компилятор добавит специальные функции в исполняемый код вашей программы, которые будут собирать разную информацию о выполнении ее функций (сколько раз функция была вызвана, откуда она вызывалась и т.п.).

Замечание
В случай раздельной компиляции проекта этот ключ указывается как на этапе компиляции, так и на этапе компоновки.

Выполнение программы

После компиляции программы ее необходимо запустить, чтобы получить данные профилирования, которые будет анализировать утилита *gprof*. Помните, что входные данные, которые обрабатывает ваша программа, существенным образом влияют на результаты профилирования, потому что данные профилирования описывают лишь ту часть вашей программы, которая обрабатывала эти входные данные. Поэтому если во время выполнения программы вы не используете какие-то возможности программы, информация о функционировании этого функционала получена не будет.

Во время выполнения ваша программа записывает данные профилирования в файл, который называется gmon.out. Если файл с таким именем уже существует, он будет перезаписан.

Замечание
Если вам необходимо анализировать данные разных запусков вашей программы, то перед выполнением очередного запуска файл gmon.out нужно переименовать.

Чтобы файл gmon.out успешно сохранился, программа должна завершится корректным образом: нельзя прерывать ее выполнения с помощью сочетания клавиш “Ctrl-C” или сняв соответствующий процесс.

Анализ результатов профилирования

После того как вы получили результаты профилирования, необходимо запустить утилиту *gprof* для интерпретации результатов. Запуск утилиты *gprof* выглядит следующим образом

```
gprof app-name [data-file] [> out-file]
```

Если вы не указываете имя файла с данными (параметр **data-file**), подразумевается файл gmon.out. При запуске утилиты *gprof* можно указать имена нескольких файлов с данными, и утилита проанализирует их все.

Запуск утилиты *gprof* с параметром "**> out-file**" подразумевает сохранение вывода утилиты в файл **out-file**. В этом случае вы можете проанализировать его позже в удобном для вас текстовом редакторе. Если вывод не перенаправить в файл, он окажется на экране.

Запустим утилиту *gprof* следующим образом

```
gprof bouble.exe > report.txt
```

Рассмотрим содержимое файла report.txt. Утилита *gprof* предоставляет разные форматы для представления выходных данных.

Плоский профиль (flat profile)

Плоский профиль показывает время, затраченное на выполнение каждой функции программы. Ниже приведена часть плоского профиля для анализируемой программы.

```
Плоский профиль:

Все образцы считаются как 0.01 seconds.
  %    суммарное    сама              сама      всего
время   seconds    seconds   вызовы ms/вызов ms/вызов  имя
 60.93      0.46      0.46        1   460.00   490.00  arr_sort
 34.44      0.72      0.26                             _mcount_private
  3.97      0.75      0.03 24895632     0.00     0.00  swap
  0.66      0.76      0.01                             _fentry__
  0.00      0.76      0.00        1     0.00     0.00  arr_rand
```

Замечание
_mcount_private, _fentry__ - это специальные функции, которые добавил компилятор для сбора данных профилирования. Время выполнения этих функция позволяет измерить накладные расходы от профилирования.

Функции упорядочены по убыванию времени выполнения, затем по убыванию количества вызовов и, наконец, по имени функции.

Перед строкой с заголовками указан период времени, через который собираются данные профилирования.

% time	% время	Процент от общего времени выполнения программы, проведенного в этой функции.
cumulative seconds	суммарное seconds	Суммарное время выполнения этой функции и функций, расположенных выше в таблице.
self seconds	сама seconds	Время выполнения этой функции.
calls	вызовы	Количество вызовов этой функции.
self	сама	Среднее время выполнения одного вызова этой функции.

ms/call	ms/вызовы	
total ms/call	всего ms/вызовы	Среднее время выполнения одного вызова этой функции и всех функций, которые вызываются из нее.
name	имя	Имя функции.

Граф вызовов (call graph)

Граф вызовов показывает сколько времени программа проводит в функции и во всех дочерних вызовах из этой функции. Ниже приведена часть графа вызовов для анализируемой программы.

```
индекс % время    сама  потомок    вызван     имя
                 0.09    0.02       1/1          main [2]
[1]     66.7     0.09    0.02       1         arr_sort [1]
                 0.02    0.00 6174046/6174046     swap [4]
-----------------------------------------------
                                                  <самопроизвольно>
[2]     66.7     0.00    0.11                  main [2]
                 0.09    0.02       1/1          arr_sort [1]
                 0.00    0.00       1/1          arr_rand [7]
-----------------------------------------------
                 0.02    0.00 6174046/6174046     arr_sort [1]
[4]     12.1     0.02    0.00 6174046          swap [4]
-----------------------------------------------
```

Линии из черточек делят этот отчет на так называемые *записи*. В каждой записи есть строка, которая начинается с индекса, указанного в квадратных скобках. Эта строка называется *главной*. В конце главной строки приведено имя функции, для которой сформирована эта запись. Строки, которые расположены выше этой, описывают последовательность вызовов других функций, в результате которых была вызвана анализируемая функция. Строки, которые расположены ниже этой, описывают функции, вызываемые анализируемой. Эти функции называются *дочерними*.

Записи отсортированы по времени, затраченном на выполнение анализируемой функции.