# Лабораторная работа № 3 по курсу дискретного анализа: Исследование качества программ

## Условие

Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти.
В случае выявления ошибок или явных недочётов, требуется их исправить.

## Метод решения

Для выявления ошибок и измерения основных параметров программы (временных затрат выполнения и расхода оперативной памяти) будем использовать утилиты *Valgrind*[1] и *gprof*[2] соответственно.

[1]Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail.

[2]Gprof is a performance analysis tool for Unix applications. It used a hybrid of instrumentation and sampling.

## Valgrind

При отправке кода на чекер была ошибка с закрытием файла, которую удалось отловить с использованием утилиты:

```
hplp739@user:~/Desktop/C/DA/lab2$ bash run.sh 1
+ g++ ./testing/genTest.cpp -o ./testing/get.out
+ ./testing/get.out 1
+ gcc -O2 B-tree.c -Wno-unused-result
+ gcc ./testing/check.c -o ./testing/solv.out
+ ./testing/solv.out
+ cat
+ valgrind ./a.out
+ cat
==11918== Invalid read of size 4
==11918==   at 0x48E0DDB: fclose@@GLIBC_2.2.5 (iofclose.c:48)
==11918==   by 0x10966E: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11918== Address 0x0 is not stack'd, malloc'd or (recently) free'd
==11918==
==11918==
==11918== Process terminating with default action of signal 11 (SIGSEGV)
==11918== Access not within mapped region at address 0x0
==11918==   at 0x48E0DDB: fclose@@GLIBC_2.2.5 (iofclose.c:48)
==11918==   by 0x10966E: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11918== If you believe this happened as a result of a stack
==11918== overflow in your program's main thread (unlikely but
==11918== possible), you can try to increase the size of the
==11918== main thread stack using the --main-stacksize= flag.
==11918== The main thread stack size used in this run was 8388608.
==11918==
==11918== HEAP SUMMARY:
==11918==    in use at exit: 4,048 bytes in 2 blocks
==11918==  total heap usage: 5 allocs, 3 frees, 6,568 bytes allocated
==11918==
==11918== LEAK SUMMARY:
==11918==   definitely lost: 0 bytes in 0 blocks
==11918==   indirectly lost: 0 bytes in 0 blocks
==11918==     possibly lost: 0 bytes in 0 blocks
==11918==   still reachable: 4,048 bytes in 2 blocks
==11918==        suppressed: 0 bytes in 0 blocks
==11918== Rerun with --leak-check=full to see details of leaked memory
==11918==
==11918== For lists of detected and suppressed errors, rerun with: -s
==11918== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Первым делом убедимся в отсутствии как-либо ошибок в работе программы. Утечки памяти, выход за границы массива, использование неинициализированных значений — всё это поможет выявить утилита *Valgrind*.

```
hplp739@user:~/Desktop/C/DA/lab2$ bash run.sh 100000
+ g++ ./testing/genTest.cpp -o ./testing/get.out
+ ./testing/get.out 100000
+ gcc -O2 B-tree.c -Wno-unused-result
+ gcc ./testing/check.c -o ./testing/solv.out
+ ./testing/solv.out
+ cat
+ valgrind ./a.out
+ cat
==10574== Memcheck, a memory error detector
==10574== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10574== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10574== Command: ./a.out
==10574==
==10574==
==10574== HEAP SUMMARY:
==10574==    in use at exit: 0 bytes in 0 blocks
==10574==  total heap usage: 19 allocs, 19 frees, 12,293,168 bytes allocated
==10574==
==10574== All heap blocks were freed -- no leaks are possible
==10574==
==10574== For lists of detected and suppressed errors, rerun with: -s
==10574== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
++ diff -u ./testing/my.ans ./testing/correct.ans
+ echo
```

Диагностика показала, что программа верно работает с памятью. Было произведено 19 аллокаций памяти из кучи суммой на 12293168 байт, затем ровно столько же освобождено. Так же не возникло иных ошибок, упомянутых выше.

Для демонстрации того, что они действительно могли быть обработаны, нарочно воспользуемся значением переменной не инициализированной никаким значением:

```
   hplp739@user:~/Desktop/C/DA/lab2$ bash run.sh 1
+ g++ ./testing/genTest.cpp -o ./testing/get.out
+ ./testing/get.out 1
+ gcc -O2 B-tree.c -Wno-unused-result
+ gcc ./testing/check.c -o ./testing/solv.out
+ ./testing/solv.out
+ cat
+ valgrind ./a.out
+ cat
==11238== Memcheck, a memory error detector
==11238== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
```

```
==11238== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==11238== Command: ./a.out
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x1092E1: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x1092EB: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x1092F5: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x1092FF: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x48E3F37: ungetc (ioungetc.c:34)
==11238==   by 0x10930C: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x10B236: _scanSTR (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==   by 0x109314: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x10B23A: _scanSTR (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==   by 0x109314: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x10B241: _scanSTR (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==   by 0x109314: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Use of uninitialised value of size 8
==11238==   at 0x10B243: _scanSTR (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==   by 0xFFF: ???
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x10B247: _scanSTR (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==   by 0x109314: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Conditional jump or move depends on uninitialised value(s)
==11238==   at 0x1095F0: StringToLower (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==   by 0x10931C: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== Use of uninitialised value of size 8
==11238==   at 0x10B243: _scanSTR (in /home/egortest/Desktop/C/DA/lab2/a.out)
```

```
==11238==   by 0xFFF: ???
==11238==   by 0x4A4C69F: ??? (in /usr/lib/x86_64-linux-gnu/libc-2.31.so)
==11238==   by 0x48E0D1C: _IO_file_doallocate (filedoalloc.c:104)
==11238==   by 0x1FFEFFFBBF: ???
==11238==   by 0x1FFEFFFC0F: ???
==11238==   by 0x1FFEFFFBB7: ???
==11238==   by 0x1FFEFFFBBB: ???
==11238==   by 0x48F1945: _IO_sputbackc (genops.c:645)
==11238==   by 0x109314: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238==
==11238== More than 10000000 total errors detected. I'm not reporting any more.
==11238== Final error counts will be inaccurate. Go fix your program!
==11238== Rerun with --error-limit=no to disable this cutoff. Note
==11238== that errors may occur in your program without prior warning from
==11238== Valgrind, because errors are no longer being displayed.
==11238==
^C==11238==
==11238== Process terminating with default action of signal 2 (SIGINT)
==11238==   at 0x496CFD0: read (read.c:26)
==11238==   by 0x48EFB9E: _IO_file_underflow@@GLIBC_2.2.5 (fileops.c:517)
==11238==   by 0x48F0F85: _IO_default_uflow (genops.c:362)
==11238==   by 0x10B28C: _scanSTR (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==   by 0x109314: main (in /home/egortest/Desktop/C/DA/lab2/a.out)
==11238==
==11238== HEAP SUMMARY:
==11238==    in use at exit: 4,048 bytes in 2 blocks
==11238==  total heap usage: 1,666,551 allocs, 1,666,549 frees, 213,330,256 bytes
    allocated
==11238==
==11238== LEAK SUMMARY:
==11238==   definitely lost: 0 bytes in 0 blocks
==11238==   indirectly lost: 0 bytes in 0 blocks
==11238==     possibly lost: 0 bytes in 0 blocks
==11238==   still reachable: 4,048 bytes in 2 blocks
==11238==        suppressed: 0 bytes in 0 blocks
==11238== Rerun with --leak-check=full to see details of leaked memory
==11238==
==11238== Use --track-origins=yes to see where uninitialised values come from
==11238== For lists of detected and suppressed errors, rerun with: -s
==11238== ERROR SUMMARY: 10000000 errors from 12 contexts (suppressed: 0 from 0)
```

Из вывода программы видно, что было обнаружена наша ошибка. Далее программа начала крутиться в бесконечном цикле и считывать входные данные не в том порядке. Обо всём этом нам и доложено в выводе программы.

## Gprof

Профилировщик показывает, сколько процентов от общего времени работы программы работает и сколько раз вызывается каждая функция (и ещё много данных, которые не так важны для нашей задачи). Диагностика проводилась на тесте из 1000000 строк с запросами на поиск, добавление и удаление. Запустим тест, и затем просмотрим вывод программы

```
hplp739@user:~/Desktop/C/DA/lab2$ bash run.sh 1000000
+ g++ ./testing/genTest.cpp -o ./testing/get.out
+ ./testing/get.out 1000000
+ gcc -O2 -pg B-tree.c -Wno-unused-result
+ gcc ./testing/check.c -o ./testing/solv.out
+ ./testing/solv.out
+ cat
+ cat
+ ./a.out
++ diff -u ./testing/my.ans ./testing/correct.ans
+ echo

+ gprof a.out
+ cat
```

Оформим вывод программы в виде таблицы

| % time | total seconds | self seconds | cals | self ms/cals | total ms/cals | name |
|---|---|---|---|---|---|---|
| 35.76 | 0.05 | 0.05 | 669803 | 0.00 | 0.00 | TreeSearch |
| 14.30 | 0.07 | 0.02 | 667968 | 0.00 | 0.00 | SearchInNode |
| 14.30 | 0.09 | 0.02 | 14 | 1.43 | 1.43 | ElListExpand |
| 14.30 | 0.11 | 0.02 | | | | _printSTR |
| 7.15 | 0.12 | 0.01 | 167898 | 0.00 | 0.00 | Insert |
| 7.15 | 0.13 | 0.01 | | | | _scanSTR |
| 3.58 | 0.14 | 0.01 | | | | NodListDestroy |
| 3.58 | 0.14 | 0.01 | | | | _scanCHR |
| 0.00 | 0.14 | 0.00 | 183341 | 0.00 | 0.00 | RemFromNode |
| 0.00 | 0.14 | 0.00 | 167898 | 0.00 | 0.00 | ElListInsert |
| 0.00 | 0.14 | 0.00 | 167296 | 0.00 | 0.00 | RemoveNode |
| 0.00 | 0.14 | 0.00 | 32232 | 0.00 | 0.00 | Rebalance |
| 0.00 | 0.14 | 0.00 | 6535 | 0.00 | 0.00 | NodListInsert |
| 0.00 | 0.14 | 0.00 | 6382 | 0.00 | 0.00 | TreeSplitChild |
| 0.00 | 0.14 | 0.00 | 6280 | 0.00 | 0.00 | MergeNodes |
| 0.00 | 0.14 | 0.00 | 4 | 0.00 | 0.00 | NodListExpand |
| 0.00 | 0.14 | 0.00 | 1 | 0.00 | 0.00 | ElListCreate |
| 0.00 | 0.14 | 0.00 | 1 | 0.00 | 0.00 | NodListCreate |
| 0.00 | 0.14 | 0.00 | 1 | 0.00 | 0.00 | TreeDelete |

Видно, что самые большие затраты времени идут на поиск в дереве. Это достаточно очевидно, так как функция поиска вызывается и при удалении, и при вставке. Далее идёт функция *ElListExpand* которая отвечает за расширения памяти для хранения элементов. И заключает четвёрку самых долгих мест программы функция вывода строки. Исходя из полученных данных, можно сделать вывод, что для ускорения всей работы будет наиболее эффективно оптимизировать именно функции поиска в B-дереве, ведь $\frac{1}{3}$ времени исполняется именно она.

## Выводы

Выполнив третью лабораторную работу по курсу Дискретный анализ, изучил и применил средства диагностики и профилирования, в частности Valgrind и gprof. Исходя из их популярности, понятно, что они часто используются на практике. Valgrind помог мне найти неправильное обращение с файлом в программе и исправить эту ошибку. Gprof показал, какие функции дольше всего выполняет моя программа: поиск в дереве и ввод/вывод данных. Эти утилиты помогают оптимизировать код с целью уменьшения потребляемой памяти и времени работы программы. Учитывая, что это основные ресурсы компьютера, то таким образом можно в целом улучшить качество работы и быстродействие программы.