Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Задание 2 - Алгоритмы поиска значений

тема

|  |  |  |
| --- | --- | --- |
| Преподаватель |  | Р.Ю. Царев |
|  | подпись, дата | инициалы, фамилия |
| Студент КИ18-17/1б 031831229 |  | В.А. Прекель |
| номер группы, зачетной книжки | подпись, дата | инициалы, фамилия |

Красноярск 2020

# 1 Цель работы с постановкой задачи

## 1.1 Цель работы

Алгоритмы поиска значений.

## 1.2 Задача работы

Реализовать в программе два алгоритма (по выбору студента) из указанных ниже:

а) алгоритм поиска минимального (или максимального) значения,

б) алгоритм поиска моды,

в) алгоритм поиска медианы,

г) алгоритм поиска среднего значения,

д) алгоритм поиска значения, равного заданному.

Сравнить эффективность реализованных алгоритмов по времени их выполнения.

Требования к выполнению лабораторной работы:

1. Строгое соответствие программы и результатов ее работы с полученным заданием.

2. Самостоятельные тестирование и отладка программы.

3. Устойчивость работы программы при любых воздействиях, задаваемых пользователем через интерфейс программы.

4. Предоставление демонстрационного примера и исходного текста программы для защиты.

5. Предоставление отчета по практическому заданию, содержащего описание реализованных алгоритмов, программы, результатов работы программы (отчет необходимо загрузить на сайт курса).

# 2 Описание реализованного алгоритма

Реализован алгоритм поиска максимального значения, алгоритм поиска среднего значения. Алгоритмическая часть реализована на языке ассемблера (GNU Assembler для Linux x64), часть взаимодействия с пользователем и юнит-тесты на Си.

# 3 Описание программы (листинги кода)

Листинг 1 – Lab\_02/Lab\_02\_Lib/Lib.s

.text  
  
# int MaxInArray(const int\* array, int count);  
 .globl MaxInArray  
MaxInArray:  
 # пролог  
 pushq %rbp  
 movq %rsp, %rbp  
 # запись в стек адреса первого элемента массива -24(%rbp)  
 # он будет смещатся чтобы получить нужный элемент  
 movq %rdi, -24(%rbp) # -24(%rbp) <- array  
 # запись в стек кол-ва элементов -28(%rbp)  
 movl %esi, -28(%rbp) # -28(%rbp) <- count  
 # максимум (в начале первый элемент массива) -4(%rbp)  
 movq -24(%rbp), %rax # %rax <- array  
 movl (%rax), %eax # %eax <- (%rax)[0]  
 movl %eax, -4(%rbp) # -4(%rbp) <- %eax  
 # номер текущего элемента -8(%rbp)  
 movl $0, -8(%rbp) # -8(%rbp) <- 0  
Loop1\_Start:  
 # сравнение номера текущего элемента с кол-вом элементов  
 # если номер текущего равен или больше, то конец цикла  
 movl -8(%rbp), %eax # %eax <- -8(%rbp)  
 cmpl -28(%rbp), %eax # compare -28(%rbp) <= %eax  
 jge Loop1\_End # if greater or equal go to Loop1\_End  
 # запись текущего элемента в %eax  
 movq -24(%rbp), %rax # %rax <- -24(%rbp)  
 movl (%rax), %eax # %eax <- %rax[0]  
 # сравнение текущего и максимального  
 # если максимальный больше, цикл идёт с начала  
 cmpl %eax, -4(%rbp) # compare %eax <= -4(%rbp)  
 jge Loop1\_Continue # if greater or equal to to Loop1\_Continue  
 # иначе максимальному присваивается текущий  
 movl %eax, -4(%rbp) # -4(%rbp) <- %eax  
Loop1\_Continue:  
 # номер текущего увеличивается на 1  
 addl $1, -8(%rbp) # -8(%rbp) += 1  
 # адрес текущего увеличивается на 4 (байта)  
 movq -24(%rbp), %rax # %rax <- -24(%rbp)  
 addq $4, %rax # %rax += 4  
 movq %rax, -24(%rbp) # -24(%rbp) <- %rax  
 # цикл идёт с начала  
 jmp Loop1\_Start  
Loop1\_End:  
 # возврат значения  
 # return -4(%rbp)  
 movl -4(%rbp), %eax # %eax <- -4(%rbp)  
 # эпилог  
 popq %rbp  
 ret  
  
# double AverageInArray(const int\* array, int count);  
 .globl AverageInArray  
AverageInArray:  
 # пролог  
 pushq %rbp  
 movq %rsp, %rbp  
 # запись в стек адреса первого элемента массива -24(%rbp)  
 # он будет смещатся чтобы получить нужный элемент  
 movq %rdi, -24(%rbp) # -24(%rbp) <- array  
 # запись в стек кол-ва элементов -28(%rbp)  
 movl %esi, -28(%rbp) # -28(%rbp) <- count  
 # запись в стек 0 с плавающей точкой для суммы -12(%rbp)  
 pxor %xmm0, %xmm0 # %xmm0 <- 0  
 movsd %xmm0, -12(%rbp) # -12(%rbp) <- %xmm0  
 # номер текущего элемента -4(%rbp)  
 movl $0, -4(%rbp) # -4(%rbp) <- 0  
Loop2\_Start:  
 # сравнение номера текущего элемента с кол-вом элементов  
 # если номер текущего равен или больше, то конец цикла  
 movl -4(%rbp), %eax # %eax <- -4(%rbp)  
 cmpl -28(%rbp), %eax # compare -28(%rbp) <= %eax  
 jge Loop2\_End # if true go to Loop1\_End  
 # запись текущего элемента в %eax  
 movq -24(%rbp), %rax # %rax <- -24(%rbp)  
 movl (%rax), %eax # %eax <- %rax[0]  
 # записать текущий элемент в регистр с плавающей точкой  
 cvtsi2sdl %eax, %xmm0 # %xmm0 <- %eax  
 # увеличить сумму на текущий элемент  
 movsd -12(%rbp), %xmm1 # %xmm1 <- -12(%rbp)  
 addsd %xmm1, %xmm0 # %xmm0 += %xmm1  
 movsd %xmm0, -12(%rbp) # -12(%rbp) <- %xmm0  
 # номер текущего увеличивается на 1  
 addl $1, -4(%rbp) # -4(%rbp) += 1  
 # адрес текущего увеличивается на 4 (байта)  
 movq -24(%rbp), %rax # %rax <- -24(%rbp)  
 addq $4, %rax # %rax += 4  
 movq %rax, -24(%rbp) # -24(%rbp) <- %rax  
 # цикл идёт с начала  
 jmp Loop2\_Start  
Loop2\_End:  
 # записываем кол-во элементов в регистр с плавающей точкой  
 cvtsi2sdl -28(%rbp), %xmm1 # %xmm1 <- -28(%rbp)  
 # делим сумму на кол-во элементов  
 # результат будет в регистре %xmm0 который возвращается  
 movsd -12(%rbp), %xmm0 # %xmm0 <- -12(%rbp)  
 divsd %xmm1, %xmm0 # %xmm0 /= %xmm1  
 # эпилог  
 popq %rbp  
 ret

Остальной код доступен в репозитории по адресу https://github.com/prekel/AlgorithmsAndDataStructures/tree/develop/Lab\_02.

# 4 Результаты работы программы

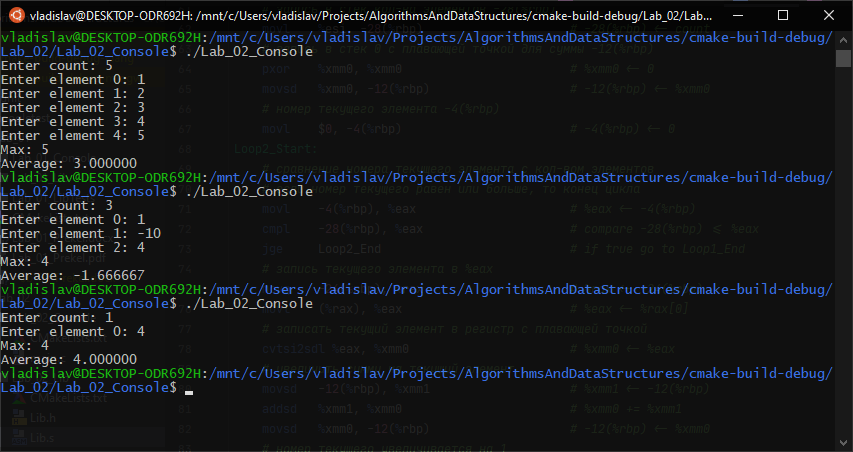


Рисунок 1 – Пример работы программы

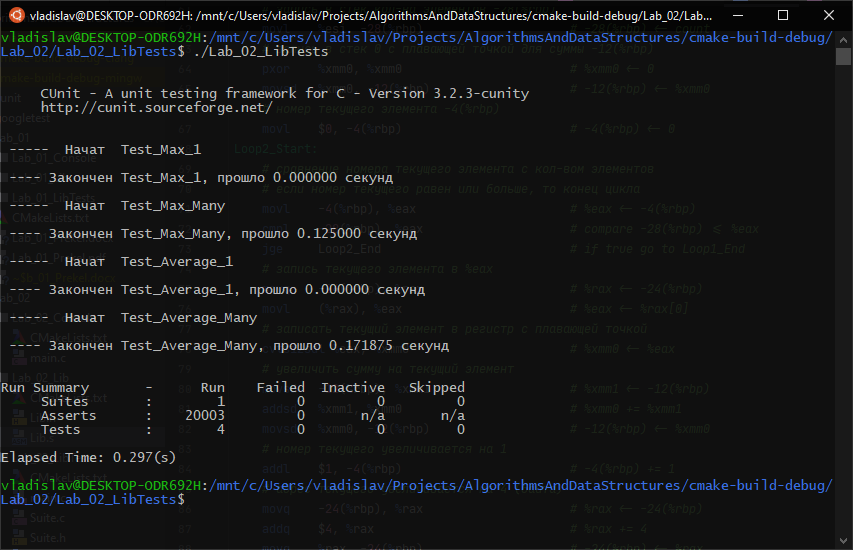


Рисунок 2 – Запуск тестов