

Правительство Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования «Национальный

исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук

Департамент программной инженерии

Отчет к домашнему заданию По дисциплине

«Архитектура вычислительных систем»

Работу выполнил:

Студент группы БПИ-191 Рычков К.П.

Москва 2020

Задача

Разработать программу вычисления корня квадратного по итерационной формуле Герона Александрийского с точностью не хуже 0,05% (использовать FPU)

Решение

Итерационная формула Герона имеет следующий вид:

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{a}{x_n} \right)$$

где a — фиксированное положительное число, а x_1 — любое положительное число.

Итерационная формула задаёт убывающую (начиная со 2-го элемента)

последовательность, которая при любом выборе $\{x_1\}$ быстро сходится к величине \sqrt{a} (квадратный корень из числа), то есть

$$\lim_{n \rightarrow \infty} x_n = \sqrt{a}$$

Для проверки полученного результата следует вычислить точное значение \sqrt{a} при помощи встроенной в FPU команды `fsqrt`

Для лучшего понимания опишем алгоритм в словесной форме. В первую очередь мы считываем вещественное значение a и проверяем его корректность ($a \geq 0$) после чего высчитываем точное значение \sqrt{a} , используя вышеупомянутую команду `fsqrt` и выводим ее в консоль, чтобы точный результат был у пользователя на глазах. Далее реализуем алгоритм вычисления корня по формуле Герона. Присваиваем $x_1 = 1$ и в цикле вычисляем

значение $x_n = \frac{1}{2} \left(x_{n-1} + \frac{a}{x_{n-1}} \right)$ при этом каждый раз сохраняя x_n и x_{n-1} . В конце каждой

итерации цикла проверяем $|x_n - x_{n-1}| < 0,0005$ (указанная в условии задачи погрешность 0,05%), если условие выполняется, то выходим из цикла, иначе реализуем следующую итерацию. После вычисления значения по формуле Герона, выводим его в консоль и находим погрешность, которую потом также выводим на экран, чтобы пользователь мог проверить допустима ли эта погрешность или нет

Теперь реализуем этот алгоритм на языке ассемблера для компилятора FASM.

Для ввода исходных данных будем использовать функцию из стандартной библиотеки `msvcrt.dll` `char *gets(char *str)`. Функция считывает символы из `stdin` и помещает их в массив символов, на который указывает `str`.

Для того, чтобы распарсить строку в действительное значение, применим функцию `int sscanf(char *buf, const char *format, arg-list)`, которая распознает и считывает данные по заданному шаблону из строки.

Аргументы функции:

- buf – указатель на символьный буфер, подлежащий считыванию;
- format – указатель на C-строку, содержащую формат результата;
- остальные аргументы – данные, подлежащие форматированию;

Для вывода в консоль используется функция из той же стандартной библиотеки msvcrt.dll, функция `int printf(const char *format, arg-list)`.

Аргументы функции:

- format – указатель на C-строку, содержащую формат результата;
- остальные аргументы – данные, подлежащие форматированию.

Для завершения работы программы выполним вызов функции `void ExitProcess(uint uExitCode)`.

Аргументы функции:

- uExitCode – код выхода для всех потоков

Функции созданные во время написания программы:

Главная функция программы. В ней считываются входные данные, проверяются на корректность и вызываются функции для выполнения вычисления квадратного корня.

`double geronSqrt(double x, double inxct)` - вычисляет значение `sqrt(x)` с погрешностью `inxct` и сохраняет его в `st(0)`

Параметры функции:

- x – значение, корень которого надо найти
- inxct – допустимая погрешность

`double sqrt(double x)` - вычисляет точное значение `sqrt(x)` и сохраняет его в `st(0)`

Параметры функции:

- x - значение, корень которого надо найти

Текст программы приведен ниже:

```
format PE Console
```

```
entry start
```

```
include 'win32a.inc'
```

```
;-----  
; Выполнил: Рычков Кирилл Павлович  
; Группа: 191  
;  
; Вариант 21:  
; Разработать программу вычисления корня  
; квадратного по итерационной формуле  
; Герона Александрийского с точностью не  
; хуже 0,05% (использовать FPU)
```

```

;-----

section '.data' data readable writeable

x            dq ?;Введённое пользователем значение:
res1         dq ?
res2         dq ?
const2       dq 2.0;константа равная 2
buf          db 256 dup(0);для чтения числа типа double
inxct        dd 0.0005;Точность 0.05%
msg1         db 'f=sqrt(x), x>=0',10,'Enter x: ',0
wrng         db 'Wrong x!',13,10,0
outDouble    db '%lf',0
geronStr     db 'Geron sqrt(x) = %lg',13,10,0
sqrtStr      db 'sqrt(x) = %lg',13,10,0
inxctStr     db 'Inexactness: %lg'

section '.code' code readable executable
start:
    ccall [printf],msg1                ;просим пользователя ввести число
    ccall [gets],buf                  ;считываем строку с числом
    ccall [sscanf],buf,outDouble,x    ;преобразуем строку в double

    ;проверяем удалось ли преобразовать строку в double
    cmp eax,1
    jz main

wrongMsg:;выводим сообщение об ошибке
    ccall [printf],wrng
    jmp start                        ;просим пользователя ввести число еще раз

main:    ;проверяем корректность введенного числа
    fld [x]                          ;введенное пользователем число
    fldz                             ;0
    fcompp                           ;сравниваем 0 с введенным числом
    fstsw ax                         ;получаем флаги
    sahf                             ;переносим их в флаги процессора
    ja wrongMsg                      ;0>x, то просим пользователя ввести x еще раз

    ;передаем в стек x
    fld qword [x]
    sub esp, 8
    fstp qword [esp]
    call sqrt                        ;вычислить точное значение sqrt(x)
    add esp, 8                       ;очищаем стек от переданных параметров

    fst [res1]                       ;сохраняем результат вычислений в res1

    ;выводим результат sqrt(x)
    sub esp, 8
    fstp qword [esp]
    push sqrtStr                     ;Формат сообщения
    call [printf]                    ;выводим результат
    add esp, 12                      ;очищаем стек от переданных параметров

    ;передаем в стек точность вычисления
    fld [inxct]
    sub esp, 8
    fstp qword [esp]

```

```

;передаем в стек значение x
fld qword [x]
sub esp, 8
fstp qword [esp]
call geronSqrt ;вычисляет значение sqrt по формуле Герона
add esp, 16 ;очищаем стек от переданных параметров

fst [res2] ;сохраняем результат вычислений в res2

;выводим результат вычислений в консоль
sub esp, 8
fstp qword [esp]
push geronStr ;формат сообщения
call [printf]
add esp, 12 ;очищаем стек от переданных параметров

;находим погрешность
fld [res1] ;получаем значение res1
fld [res2] ;получаем значение res2
fsubp st1, st ;вычисляем их разность
fabs ;получаем модуль их разности

;выводим погрешность
sub esp, 8
fstp qword [esp]
push inxctStr ;Формат сообщения
call [printf] ;выводим погрешность
add esp, 12 ;очищаем стек от переданных параметров

ccall [_getch] ;ожидание нажатия любой клавиши

stdcall [ExitProcess], 0 ;завершение работы программы

;-----Описание-GeronSqrt-----
; Вычисляет значение sqrt(x) с погрешностью inxct и сохраняет его
; в st(0)
; Параметры функции:
; xl equ ebp+8 ;значение x
; inxctl equ ebp+16 ;погрешность
;
;-----GeronSqrt(double x, double inxct)-----

;Объявление локальных переменных:
xi equ ebp-8 ;значение x_i
xp equ ebp-16 ;значение x_(i-1)

geronSqrt:
push ebp ;сохраняем значение регистра ebp
mov ebp, esp
sub esp, 16 ;создаем место под локальные переменные

;Вычисленное значение
fldz
fstp qword [xp] ;xp = 0
fld1
fstp qword [xi] ;xi = 1

geronLoop:
fld qword [xl] ;грузим x

```

```

        fdiv qword [xi]          ;x/xi
        fadd qword [xi]         ;x/xi + xi
        fdiv [const2]          ;(x/xi + xi)/2
        fst qword [xi]          ;xi = (x/xi + xi)/2
        fld qword [xp]
        fsubp st1, st           ;xi - xp
        fabs                    ;|xi - xp|
        fld qword [xi]
        fstp qword [xp]         ;записываем xi в xp
        fcomp qword [inxctl]    ;сравнить |xi - xp| с eps
        fstsw ax;               ;перенести флаги сравнения в ax
        sahf;                   ;занести ax в флаги процессора
        jnb geronLoop;          ;если |xi - xp|>=inxctl, продолжить цикл

        fld qword [xi]
        leave

ret
;-----

;-----Описание-Sqrt-----
; Вычисляет точное значение sqrt(x) и сохраняет его в st(0)
;-----Sqrt(double x)-----
sqrt:
        push ebp                ;сохраняем значение регистра ebp
        mov ebp,esp
        fld qword [ebp+8];x
        fsqrt                   ;sqrt(1-x^2)
        pop ebp                 ;возвращаем значение регистра ebp
ret
;-----

section '.idata' import data readable

library kernel,'kernel32.dll',\
        user,'user32.dll',\
        msvcrt,'msvcrt.dll'

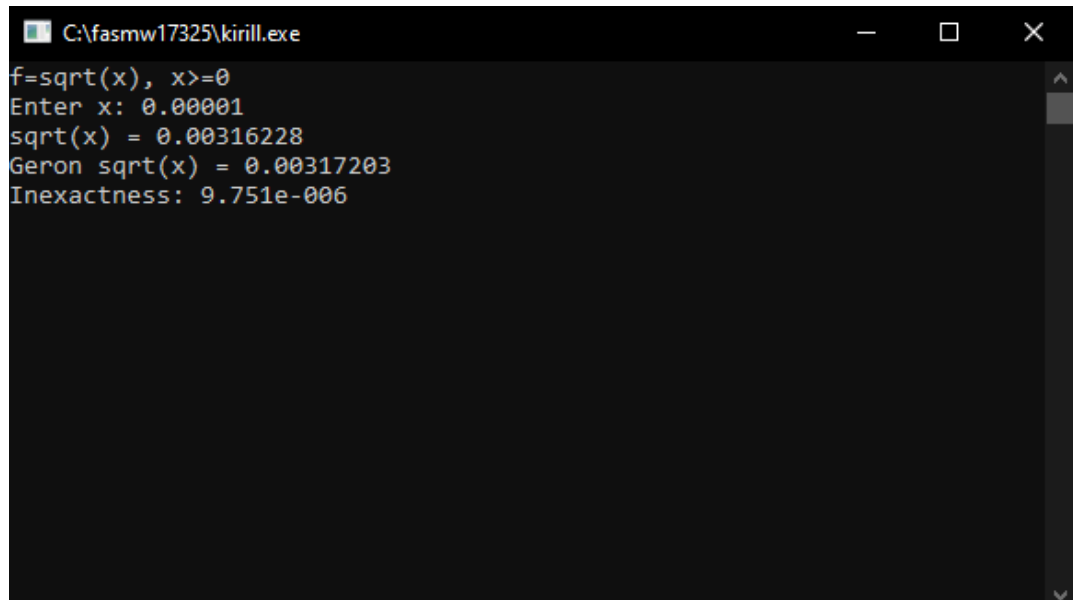
import kernel,\
        ExitProcess,'ExitProcess'

import msvcrt,\
        sscanf,'sscanf',\
        gets,'gets',\
        _getch,'_getch',\
        printf,'printf'

```

Тестирование

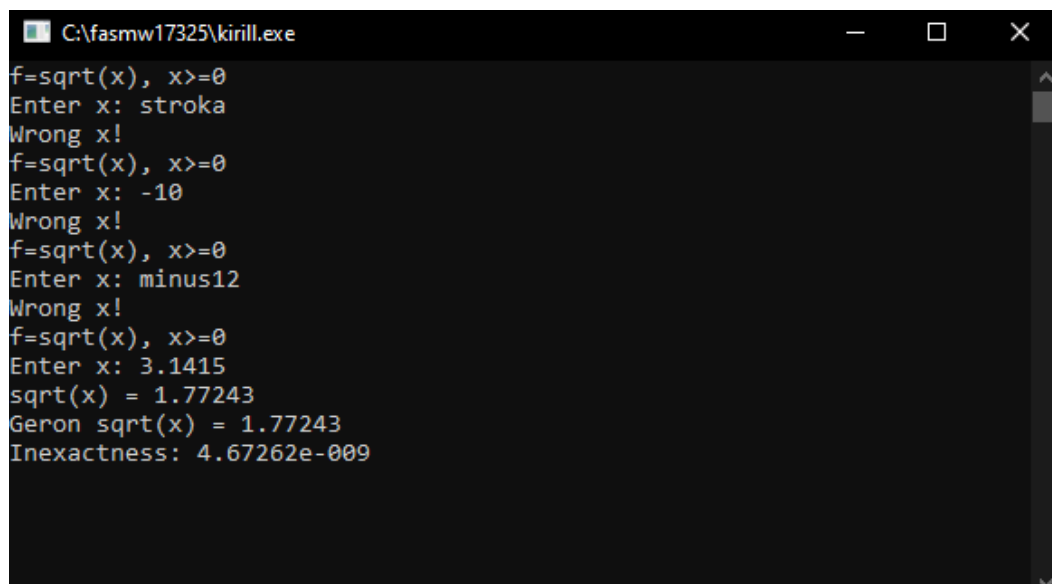
При запуске программы открывается консольное приложение и пользователя просят ввести значение x . После ввода значения x в консоль выводится “ $\text{sqrt}(x) = \{\text{точное значение } \sqrt{x}\}$ ”, “ $\text{Geron sqrt}(x) = \{\text{значение } \sqrt{x} \text{ с точностью не хуже } 0,05\%\}$ ”, “ $\text{Inexactness: \{итоговая погрешность\}}$ ” (см. рис. 1)



```
C:\fasmw17325\kirill.exe
f=sqrt(x), x>=0
Enter x: 0.00001
sqrt(x) = 0.00316228
Geron sqrt(x) = 0.00317203
Inexactness: 9.751e-006
```

Рисунок 1 – Работа программы при вводе корректных входных данных

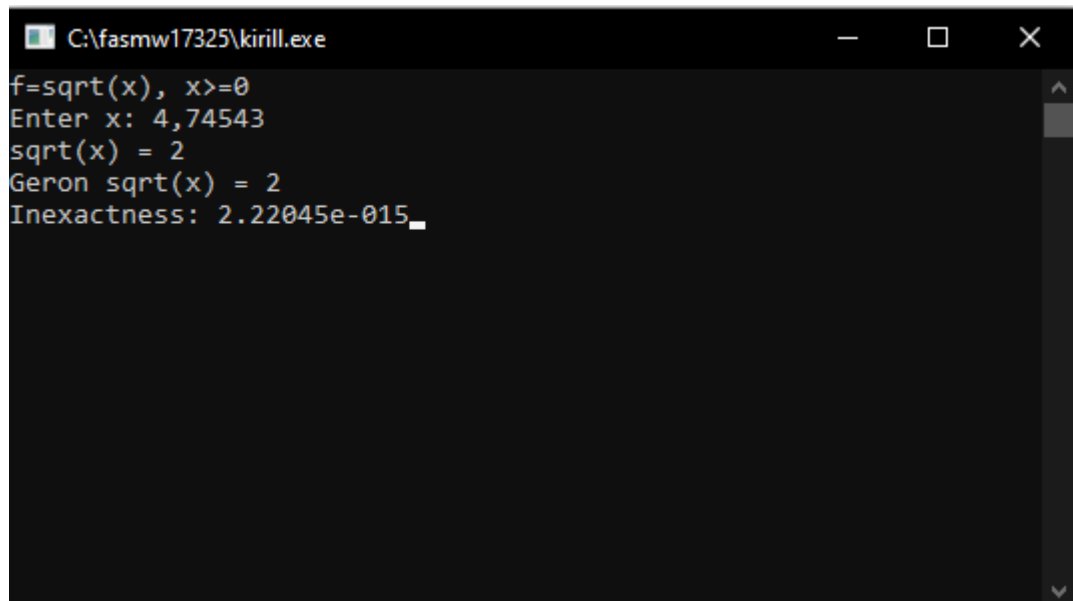
В случае если вводятся некорректные данные, то выводится сообщение “Wrong x!” и пользователя просят ввести x повторно до тех пор, пока он не введет корректные данные (см. рис. 2)



```
C:\fasmw17325\kirill.exe
f=sqrt(x), x>=0
Enter x: stroka
Wrong x!
f=sqrt(x), x>=0
Enter x: -10
Wrong x!
f=sqrt(x), x>=0
Enter x: minus12
Wrong x!
f=sqrt(x), x>=0
Enter x: 3.1415
sqrt(x) = 1.77243
Geron sqrt(x) = 1.77243
Inexactness: 4.67262e-009
```

Рисунок 2 – Работа программы при вводе некорректных данных

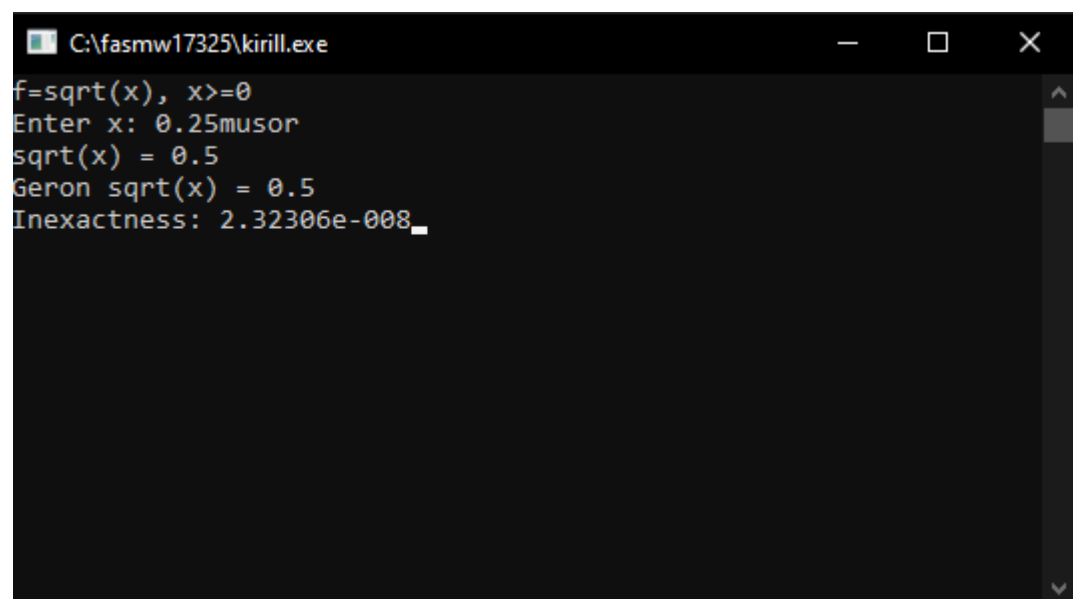
В случае если пользователь ввел вещественное число, но в качестве разделителя была написана “,” , то программа считает только целую часть, а вещественную посчитает некорректной (см. рис. 3)



```
C:\fasmw17325\kirill.exe
f=sqrt(x), x>=0
Enter x: 4,74543
sqrt(x) = 2
Geron sqrt(x) = 2
Inexactness: 2.22045e-015_
```

Рисунок 3 – Вещественное число с “,” в качестве разделителя

В случае если в начале строки находится корректное число, а в конце расположен различный “мусор”, то программа считает корректное число в начале и выведет результат для него (см. рис. 4)



```
C:\fasmw17325\kirill.exe
f=sqrt(x), x>=0
Enter x: 0.25muser
sqrt(x) = 0.5
Geron sqrt(x) = 0.5
Inexactness: 2.32306e-008_
```

Рисунок 4 – Некорректные символы после введенного числа

Ниже представлено еще несколько тестов программы (см. рис. 5-7)


```
C:\fasmw17325\kirill.exe
f=sqrt(x), x>=0
Enter x: 123456
sqrt(x) = 351.363
Geron sqrt(x) = 351.363
Inexactness: 0_
```

Рисунок 5 – Работа при вводе “123456”

```
C:\fasmw17325\kirill.exe
f=sqrt(x), x>=0
Enter x: 0.000000001
sqrt(x) = 3.16228e-005
Geron sqrt(x) = 0.000488964
Inexactness: 0.000457341
```

Рисунок 6 – Работа программы при вводе маленького числа

```
C:\fasmw17325\kirill.exe
f=sqrt(x), x>=0
Enter x: 0
sqrt(x) = 0
Geron sqrt(x) = 0.000488281
Inexactness: 0.000488281_
```

Рисунок 7 – Работа программы при вводе 0

Список используемых источников

1. Википедия (2020) «Итерационная формула Герона»
(https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%93%D0%B5%D1%80%D0%BE%D0%BD%D0%B0)
2. Легалов А.И.(2020) «Разработка программ на ассемблере. Использование макроопределений» (<http://softcraft.ru/edu/comparch/practice/asm86/04-macro/>)
3. Легалов А.И.(2020) «Разработка программ на ассемблере. Использование сопроцессора с плавающей точкой»
(<http://softcraft.ru/edu/comparch/practice/asm86/05-fpu/>)
4. osinavi «FPU (Floating Point Unit)» (<http://osinavi.ru/asm/FPUexpansion/1.html>)
5. osinavi « Основные команды загрузки и сохранения»
(<http://osinavi.ru/asm/FPUexpansion/2.html>)
6. osinavi « Арифметические команды сопроцессора»
(<http://osinavi.ru/asm/FPUexpansion/3.html>)
7. osinavi «Математические команды сопроцессора»
(<http://osinavi.ru/asm/FPUexpansion/4.html>)
8. osinavi « Команды сравнения FPU» (<http://osinavi.ru/asm/FPUexpansion/5.html>)
9. osinavi «Команды управления FPU» (<http://osinavi.ru/asm/FPUexpansion/6.html>)
10. osinavi « Дополнительные возможности сопроцессора»
(<http://osinavi.ru/asm/FPUexpansion/7.html>)