

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

**ПРОГРАММА ДЛЯ ВЫЧИСЛЕНИЯ КВАДРАТНОГО КОРНЯ ПО
ИТЕРАЦИОННОЙ ФОРМУЛЕ ГЕРОНА АЛЕКСАНДРИЙСКОГО**

Пояснительная записка

Исполнитель
студент группы **БПИ196**
Шилова Ксения Алексеевна

30 октября 2020 г.

Формулировка задачи

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

Итерационная формула Герона Александрийского

Итерационная формула Герона Александрийского – это представление члена X_n бесконечной последовательности:

$$X_{n+1} = \frac{1}{2} (X_n + N/X_n), \quad \text{где } X_1 - \text{любое положительное число} \quad [1]$$

Причем, $\lim_{n \rightarrow \infty} X_n = \sqrt{N}$.

Таким образом, чтобы вычислить квадратный корень, можно поитерационно вычислять члены последовательности, запоминая только предыдущее значение на каждом шаге.

Но для того, чтобы вычислить значение в программе, нужно задать некоторую точность, которая определяет условие выхода из цикла подсчета членов последовательности. В данной задаче точность задана как 0,05%. Это означает, что нужно выходить из цикла в том случае, если значение за одну итерацию изменилось не больше, чем на 0,05%.

Алгоритм вычисления квадратного корня

(Все инструкции FPU взяты из источников [2], [3])

1. Общая структура программы:

```
start:
    FINIT ;инициализируем сопроцессор
    call inputNumber
    call iterations

displayResult:
    invoke printf, resultNumb, dword[currentNumber], dword[currentNumber+4]

    call compareToFuncSQRT

finish:
    invoke getch
    invoke ExitProcess, 0 ; выход из программы с кодом возврата 0
```

1.1 inputNumber – ввод числа и проверка на корректность и ноль

- 1.2 iterations – произведение итераций, то есть вычисление очередного члена последовательности, и проверка достижения точности
- 1.3 compareToFuncFSQRT – проверка результата с помощью команды FSQRT от введенного числа

1. Считывание числа N с консоли

- 1.1 Проверка числа на отрицательность (если число отрицательное, то запрос повторного ввода)
- 1.2 Если число неотрицательное, то выполняется проверка на 0. Если число < 0.000 000 000 000 1, то выводим 0, так как по итерационной формуле будет выводиться ноль и не нужно вычислять члены последовательности.
- 1.3 Если число положительное, то запускаем цикл подсчета членов последовательности. В этом случае берем первый член последовательности равным N.

```
-----  
inputNumber:  
    mov [tmp], esp; запоминаем указатель на стек  
  
    invoke printf, mesN      ;запрос ввода числа  
    invoke scanf, formatF, N ;считывание N  
    FLD [zero]              ;загружаем ноль, чтобы с ним сравнивать  
    FLD [N] ;помещаем число в стек ;st(0) = N  
    FST [prevNumber] ;начальное значение (x_1).  
  
    ;если число отрицательное, то не берем корень  
    FCOMPP ;сравнение с выталкиванием из стека  
    fstsw ax  
    sahf  
    jnb start ;если число отрицательное, то повторный ввод  
  
checkZero: ;проверяем число на равенство нулю  
; (точнее, на то, меньше ли число 0.000 000 000 000 1)  
; так как если оно меньше, то программа выводит в результат ноль и нет смысла проводить все итерации  
    FLD [zero] ;загрузили ноль  
    FST [currentNumber] ;пусть текущее будет ноль (если его понадобится вывести)  
  
    FLD [nearZero] ;загружаем число, близкое к нулю  
    FSUB [N] ;вычли N  
    FCOMPP ;сравниваем N и почти 0  
    fstsw ax  
    sahf  
    ja displayResult ; печатаем ноль, если N = 0  
  
    mov esp, [tmp] ;восстанавливаем указатель на стек  
ret  
-----
```

2. Цикл подсчета членов последовательности

- 2.1 Загружаем число на вершину стека
- 2.2 На вершине стека производим вычисления нового члена последовательности, зная предыдущий:
 - Делим N на X_n .
 - Прибавляем X_n .
 - Умножаем эту сумму на $\frac{1}{2}$.
 - Записываем результат в соответствующую переменную.

```

iterations:
    mov [tmp], esp ; запоминаем указатель на стек

    FLD [N] ;загружаем в вершину N
oneIteration:
    ;на вершине стека N
    FDIV [prevNumber] ;делим N на x_n
    FADD [prevNumber] ;добавляем к этому x_n => st(0) = (x_n + N/x_n)
    FMUL [half]; умножаем на 1/2
    ;теперь на вершине стека x_n+1
    FST [currentNumber] ;записываем в переменную текущее значение

```

2.3 Проверяем условие выхода из цикла (переменная изменилась за одну итерацию меньше, чем на 0.05 %).

В вершине стека вычитаем из текущего члена последовательности предыдущий и берем модуль этой разности.

Делим модуль разности на предыдущий член последовательности и вычитаем 0.0005.

Если число в вершине стека меньше или равно нулю, то новый за одну итерацию значение изменилось не больше, чем на 0.05%, это означает, что можно вывести результат (уже записан в переменную).

Если число в вершине стека больше нуля, то продолжаем вычислять члены последовательности.

```

;проверка достижения точности
;сейчас на вершине стека текущее число
FSUB [prevNumber] ;вычитаем из текущего предыдущее число
FABS ;берем модуль разности
FDIV [prevNumber] ;делим на предыдущее число -
;получаем НА сколько процентов (в долях) изменилось число при новой итерации
FSUB [precision] ;вычитаем точность

FLD [zero] ;загружаем ноль, чтобы сравнить точность и реальное изменение

FCOMPP ;сравнение с выталкиванием из стека
fstsw ax
sahf
jae displayResult ;если точность оказалась больше, чем реальное изменение

;продолжаем приближение
FLD [currentNumber]
FSTP [prevNumber] ;меняем итерационное значение
FLD [N] ;загружаем наше число на вершину

;выводим очередное число последовательности
invoke printf, currentNumb, dword[currentNumber], dword[currentNumber+4]
jmp oneIteration

mov esp, [tmp] ;восстанавливаем указатель на стек
ret

```

3. Вывод результата в консоль.

```
displayResult:
    invoke printf, resultNumb, dword[currentNumber], dword[currentNumber+4]
```

4. Вывод корня квадратного, полученного с помощью команды FSQRT (для сравнения).

```
;-----
compareToFuncSQRT:
    mov [tmp], esp    ;запоминаем указатель на стек

    FLD [N]           ;загружаем наше число на вершину стека
    FSQRT              ;вызываем функцию корня квадратного
    FSTP [realNumber] ;запоминаем значение
    invoke printf, realNumb, dword[realNumber], dword[realNumber+4]

    mov esp, [tmp]     ;восстанавливаем указатель на стек
ret
;-----
```

Список переменных

```
section '.data' data readable writable

mesN db "Enter a non-negative number: ",10, 0
half dq 0.5 ;коэффициент в формуле
zero dq 0 ;ноль
nearZero dq 0.000000000000001 ;число, при вводе которого или меньше, считаем результат равным нулю
precision dq 0.0005 ;Точность в долях
N dq ? ;число, которое введут
prevNumber dq ?
currentNumber dq ?
realNumber dq ?
formatF db '%lf', 0
currentNumb db 'Current number is %lf',10, 0
resultNumb db 10, "RESULT number is %lf", 10, 0
realNumb db 10, "Command FSQRT outcome is %lf", 10, 0
tmp dd ?
```

- 1) mesN – строка с запросом ввода числа, из которого нужно извлечь корень
- 2) half – коэффициент в формуле Герона Александрийского
- 3) zero – ноль, который нужен для сравнения чисел (то есть сравнения их разности с нулем)
- 4) nearZero – число, близкое к нулю, при вводе которого или меньше, программа должна вывести результат 0
- 5) precision – точность, с которой производятся вычисления
- 6) N – число, из которого нужно взять корень

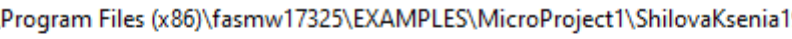
- 7) prevNumber, currentNumber – числа, в которых запоминаются X_n и X_{n+1}
- 8) realNumber – число, полученное с помощью команды FSQRT для сравнения результата
- 9) format – строка, нужна для считывания введенного N
- 10) currentNumb, resultNumb, realNumb – строки для вывода текущей итерации, результирующего числа, и числа, полученного с помощью FSQRT
- 11) tmp – переменная, в которой запоминается указатель на стек, чтобы процедуры работали корректно

Область допустимых значений

Пользователь может ввести любое вещественное число.

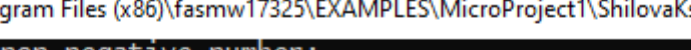
Тестирование

- 1) ввод отрицательного числа



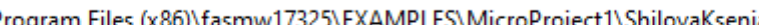
```
C:\Program Files (x86)\fasmw17325\EXAMPLES\MicroProject1\ShilovaKsenia196.exe
Enter a non-negative number:
-5
Enter a non-negative number:
```

- 2) ввод нуля



```
C:\Program Files (x86)\fasmw17325\EXAMPLES\MicroProject1\ShilovaKsenia196.exe
Enter a non-negative number:
0
RESULT number is 0.000000
Command FSQRT outcome is 0.000000
```

- 3) ввод числа, близкого к нулю слева



```
C:\Program Files (x86)\fasmw17325\EXAMPLES\MicroProject1\ShilovaKsenia196.exe
Enter a non-negative number:
-0.000000000000000000000001
Enter a non-negative number:
```

- 4) ввод числа, близкого к нулю справа

C:\Program Files (x86)\fasmw17325\EXAMPLES\MicroProject1\ShilovaKsenia196.exe

```
Enter a non-negative number:
123456789987654321
Current number is 61728394993827160.000000
Current number is 30864197496913580.000000
Current number is 15432098748456792.000000
Current number is 7716049374228400.000000
Current number is 3858024687114208.000000
Current number is 1929012343557120.000000
Current number is 964506171778592.000000
Current number is 482253085889360.000000
Current number is 241126542944808.000000
Current number is 120563271472660.000000
Current number is 60281635736842.000000
Current number is 30140817869445.000000
Current number is 15070408936770.500000
Current number is 7535204472481.250000
Current number is 3767602244432.625000
Current number is 1883801138600.312500
Current number is 941900602068.155880
Current number is 470950366570.074890
Current number is 235475314357.013120
Current number is 117737919322.312010
Current number is 58869483947.599564
Current number is 29435790537.348408
Current number is 14719992321.067463
Current number is 7364189667.816033
Current number is 3690477072.521941
Current number is 1861964936.068040
Current number is 964134754.523464
Current number is 546092032.222270
Current number is 386082631.464465
Current number is 352925210.948349
Current number is 351367636.565482

RESULT number is 351364184.286863

Command FSQRT outcome is 351364184.269903
```

Список используемых источников

- 1) https://ru.wikipedia.org/wiki/Итерационная_формула_Герона
- 2) <http://softcraft.ru/edu/comparch/practice/asm86/05-fpu/> (Презентация к семинару – команды FPU)
- 3) <http://flatassembler.narod.ru/fasm.htm#2-1-13> (Описание данных, Инструкции FPU)

Приложение

1. Код программы (FASM)

```
format PE console
entry start

include 'win32a.inc'

section '.code' code readable executable

start:
FINIT ;инициализируем сопроцессор
call inputNumber
call iterations

displayResult:
    invoke printf, resultNumb, dword[currentNumber], dword[currentNumber+4]

call compareToFuncSQRT

finish:
    invoke getch
    invoke ExitProcess, 0 ; выход из программы с кодом возврата 0

;-----
inputNumber:
    mov [tmp], esp; запоминаем указатель на стек

    invoke printf, mesN ;запрос ввода числа
    invoke scanf, formatF, N ;считывание N
    FLD [zero] ;загружаем ноль, чтобы с ним сравнивать
    FLD [N] ;помещаем число в стек ;st(0) = N
    FST [prevNumber] ;начальное значение (x_1).

    ;если число отрицательное, то не берем корень
    FCOMPP ;сравнение с выталкиванием из стека
    fstsw ax
    sahf
    jb start ;если число отрицательное, то повторный ввод

checkZero: ;проверяем число на равенство нулю
; (точнее, на то, меньше ли число 0.000 000 000 000 1)
; так как если оно меньше, то программа выводит в результат ноль и нет смысла проводить все итерации
    FLD [zero] ;загрузили ноль
    FST [currentNumber] ;пусть текущее будет ноль (если его понадобится вывести)

    FLD [nearZero] ;загружаем число, близкое к нулю
    FSUB [N] ;вычли N
    FCOMPP ;сравниваем N и почти 0
    fstsw ax
    sahf
    ja displayResult ; печатаем ноль, если N = 0

    mov esp, [tmp] ;восстанавливаем указатель на стек
ret
;-----
```

```

iterations:
    mov [tmp], esp ; запоминаем указатель на стек

    FLD [N] ;загружаем в вершину N
oneIteration:
    ;на вершине стека N
    FDIV [prevNumber] ;делим N на x_n
    FADD [prevNumber] ;добавляем к этому x_n => st(0) = (x_n + N/x_n)
    FMUL [half]; умножаем на 1/2
    ;теперь на вершине стека x_n+1
    FST [currentNumber] ;записываем в переменную текущее значение

    ;проверка достижения точности
    ;сейчас на вершине стека текущее число
    FSUB [prevNumber] ;вычитаем из текущего предыдущее число
    FABS ;берем модуль разности
    FDIV [prevNumber] ;делим на предыдущее число -
    ;получаем НА сколько процентов (в долях) изменилось число при новой итерации
    FSUB [precision] ;вычитаем точность

    FLD [zero] ;загружаем ноль, чтобы сравнить точность и реальное изменение

    FCOMPP ;сравнение с выталкиванием из стека
    fstsw ax
    sahf
    jae displayResult ;если точность оказалась больше, чем реальное изменение

    ;продолжаем приближение
    FLD [currentNumber]
    FSTP [prevNumber] ;меняем итерационное значение
    FLD [N] ;загружаем наше число на вершину

    ;выводим очередное число последовательности
    invoke printf, currentNumb, dword[currentNumber], dword[currentNumber+4]
    jmp oneIteration

    mov esp, [tmp] ;восстанавливаем указатель на стек
ret

;-----

;-----
compareToFuncSQRT:
    mov [tmp], esp ;запоминаем указатель на стек

    FLD [N] ;загружаем наше число на вершину стека
    FSQRT ;вызываем функцию корня квадратного
    FSTP [realNumber] ;запоминаем значение
    invoke printf, realNumb, dword[realNumber], dword[realNumber+4]

    mov esp, [tmp] ;восстанавливаем указатель на стек
ret
;-----

```

```

section '.data' data readable writable

    mesN db "Enter a non-negative number: ",10, 0
    half dq 0.5 ;коэффициент в формуле
    zero dq 0 ;ноль
    nearZero dq 0.00000000000001 ;число, при вводе которого или меньше, считаем результат равным нулю
    precision dq 0.0005 ;Точность в долях
    N dq ? ;число, которое введут
    prevNumber dq ?
    currentNumber dq ?
    realNumber dq ?
    formatF db '%lf', 0
    currentNumb db 'Current number is %lf',10, 0
    resultNumb db 10, "RESULT number is %lf", 10, 0
    realNumb db 10, "Command FSQRT outcome is %lf", 10, 0
    pointerSt dw ?
    tmp dd ?

section '.idata' import data readable
    library kernel, 'kernel32.dll',\
        msvcrt, 'msvcrt.dll'

    import kernel,\
        ExitProcess, 'ExitProcess'
    import msvcrt,\
        printf, 'printf',\
        sprintf, 'sprintf',\
        scanf, 'scanf',\
        getch, '_getch'

```