

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

Факультет компьютерных наук  
Департамент программной инженерии

**КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ НАХОЖДЕНИЯ  
ЗНАЧЕНИЯ ФУНКЦИИ ГИПЕРБОЛИЧЕСКОГО КОСИНУСА**  
Пояснительная записка

Исполнитель:  
студент группы БПИ199  
\_\_\_\_\_ К. Н. Борисов  
«\_\_\_\_\_» \_\_\_\_\_ 2020 г.

Москва 2020

# Содержание

<b>1</b>	<b>Текст задания</b>	<b>3</b>
<b>2</b>	<b>Применяемые расчетные методы</b>	<b>4</b>
2.1	Алгоритм вычисления	4
2.2	Считывание числа $x$	4
2.3	Вывод результата в консоль	5
2.4	Вывод промежуточных результатов	5
<b>3</b>	<b>Тестовые примеры</b>	<b>6</b>
<b>4</b>	<b>Список использованной литературы</b>	<b>7</b>
<b>5</b>	<b>Текст программы</b>	<b>8</b>

## 1. Текст задания

Разработать программу, вычисляющую с помощью степенного ряда с точностью не хуже 0.1% значение функции гиперболического косинуса  $\cosh(x) = \frac{e^x + e^{-x}}{2}$  для конкретного параметра  $x$  (использовать FPU) [1]

## 2. Применяемые расчетные методы

### 2.1. Алгоритм вычисления

Гиперболический косинус раскладывается в такой степенной ряд [3]:

$$\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \dots = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$$

Чтобы не вычислять каждый раз факториалы и степени заново мы можем хранить их во временных переменных. В идеале у нас был бы примерно такой код:

```
float cosh(float x) {
    float delta = 1.0;
    float res = 1.0;
    for (int i = 1; i < 10000; i++) {
        delta *= x / i;
        if (i % 2 == 0) res += delta;
        if (abs(delta / res) < 0.001) return;
    }
    return res;
}
```

Но из-за ограничений ассемблера мне пришлось видоизменить алгоритм, чтобы код на ассемблере был более понятным:

```
float64_t x = 0;
float64_t tmpFactorial = 0;
float64_t tmpPow = 0;
float64_t tmpSquare = 0;
float64_t chres = 0;
float64_t prevchres = 0;

void hyperbolicCosine() {
    tmpSquare = tmpPow = x * x;
    chres = 1.0;
    tmpFactorial = 2.0;

    for (int i = 3; i < 10000;) {
        chres += tmpPow / tmpFactorial;

        if (chres == ∞) {
            chres = prevchres;
            return;
        }
        if (chres == prevchres) return;
        prevchres = chres;

        tmpPow *= tmpSquare;
        tmpFactorial *= i++;
        tmpFactorial *= i++;
    }
}
```

Тут, для увеличения производительности, *i* каждый раз увеличивается на 2 и во временных переменных хранится не вся дробь, а отдельно числитель в **tmpPow** и отдельно знаменатель в **tmpFactorial**. Таким образом у нас становится в два раза меньше операций деления, а деление чисел с плавающей точкой очень медленное [4]. Но, из-за нехватки точности, **tmpFactorial** и **tmpPow** быстро уходят в ∞, и поэтому посчитать гиперболический косинус от  $|x| > 100$  получается плохо.

### 2.2. Считывание числа *x*

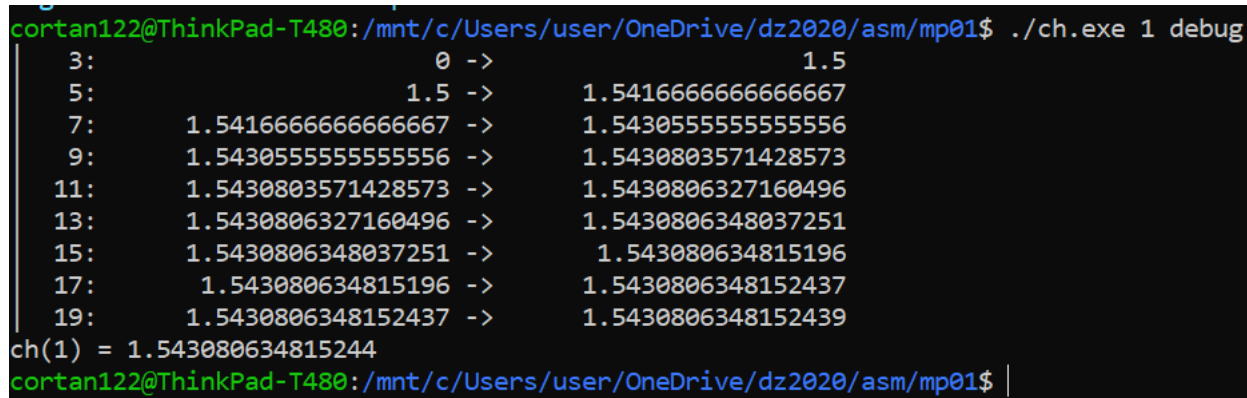
Число *x* считывается с командной строки, или из первого аргумента **argv**. Чтобы функция **scanf** работала с 64-битными числами с плавающей точкой, используется **"%lf"** вместо **"%f"**.

## 2.3. Вывод результата в консоль

Для вывода используется функция `printf` и `"%23.20g"` вместо `"%f"`, потому что `"%f"` для больших чисел выводит очень много лишних нулей. Также в `printf` числа с плавающей точкой всегда передаются 64-битные [5], а в x86-32 инструкция `push` может работать только с 32-битными числами [2].

## 2.4. Вывод промежуточных результатов

Если передать `debug` как второй аргумент, программа будет выводить промежуточный результат после каждой итерации (см. рисунок 1). Тут используется U+2502 (Box Drawings Light Vertical) вместо U+007C (Vertical Line) для того, чтобы не было пропусков в левой линии.



```
cortan122@ThinkPad-T480:/mnt/c/Users/user/OneDrive/dz2020/asm/mp01$ ./ch.exe 1 debug
 3:      0 ->      1.5
 5:      1.5 ->    1.5416666666666667
 7:    1.5416666666666667 ->    1.5430555555555556
 9:    1.5430555555555556 ->    1.5430803571428573
11:    1.5430803571428573 ->    1.5430806327160496
13:    1.5430806327160496 ->    1.5430806348037251
15:    1.5430806348037251 ->    1.543080634815196
17:    1.543080634815196 ->    1.5430806348152437
19:    1.5430806348152437 ->    1.5430806348152439
ch(1) = 1.543080634815244
cortan122@ThinkPad-T480:/mnt/c/Users/user/OneDrive/dz2020/asm/mp01$ |
```

Рисунок 1: Вывод промежуточных результатов



## 4. Список использованной литературы

- [1] Инструкция по составлению пояснительной записки [Электронный ресурс]. //URL: <http://softcraft.ru/edu/comparch/tasks/mp01/> (Дата обращения: 10.10.2020, режим доступа: свободный)
- [2] x86 Instruction Set Reference [Электронный ресурс]. //URL: <https://c9x.me/x86/> (Дата обращения: 10.10.2020, режим доступа: свободный)
- [3] Статья «Hyperbolic functions» Wikipedia.org //URL: [https://en.wikipedia.org/wiki/Hyperbolic\\_functions](https://en.wikipedia.org/wiki/Hyperbolic_functions) (Дата обращения: 10.10.2020, режим доступа: свободный)
- [4] Why is float division slow? [Электронный ресурс]. //URL: <https://stackoverflow.com/a/506345> (Дата обращения: 29.10.2020, режим доступа: свободный)
- [5] Default argument promotions [Электронный ресурс]. //URL: [https://en.cppreference.com/w/c/language/conversion#Default\\_argument\\_promotions](https://en.cppreference.com/w/c/language/conversion#Default_argument_promotions) (Дата обращения: 29.10.2020, режим доступа: свободный)

## 5. Текст программы

```
; Микропроект №1
; Автор: Борисов Костя
; Группа: БПИ199
; Дата: 10.10.2020
; Вариант: 3

; Разработать программу, вычисляющую с
; помощью степенного ряда с точностью не
; хуже 0.1% значение функции
; гиперболического косинуса  $ch(x) = (e^x + e^{-x})/2$ 
; для конкретного параметра x
; (использовать FPU)

format PE console
entry start
include 'win32a.inc'

section '.code' code readable executable
; эта функция проверяет запущены ли мы из cmd.exe и можно ли нам закрыться сразу
isGetchNeeded:
    call [GetConsoleWindow]

    push consoleWindowProcessId
    push eax
    call [GetWindowThreadProcessId]

    call [GetCurrentProcessId]
    cmp eax, [consoleWindowProcessId]
    mov eax, 0
    jne isGetchNeeded_ret
    mov eax, 1
isGetchNeeded_ret:
    test eax, eax
    ret

; это функция красиво выходит
gracefulExit:
    call isGetchNeeded
    jz gracefulExit_exit

; если мы выедем сейчас то пользователь ничего не успеет увидеть
    push exitStr
    call [printf]
    add esp, 4
    call [getch]

gracefulExit_exit:
    push 0
    call [exit]
    add esp, 4
    ret

; эта функция меняет кодировку консоли на UTF8 и читает argv
dramaticEntrance:
    push 65001
    call [SetConsoleOutputCP]

    push env
    push 0
    push env
    push argv
    push argc
```



```

    call [getmainargs]
    add esp, 20

    ret

; эта функция выводит два последних значений частичной суммы ряда (только когда надо)
debugLog:
    cmp dword[argc], 3
    jne debugLog_ret
    push ecx
    push dword[chres+4]
    push dword[chres]
    push dword[prevchres+4]
    push dword[prevchres]
    push ecx
    push debugFormat_printf
    call [printf]
    add esp, 24
    pop ecx
debugLog_ret:
    ret

; эта функция проверяет можно ли от этого числа взять адекватный
checkHyperbolicCosineBounds:
    finit
    fld qword[x]
    fabs
    fild dword[hundred]
    fcompp
    fstsw ax
    sahf
    jb checkHyperbolicCosineBounds_fail
    ret

checkHyperbolicCosineBounds_fail:
    push dword[x+4]
    push dword[x]
    push floatWrongFormat_printf
    call [printf]
    add esp, 20
    call gracefulExit
    ret

; эта функция считает chres = ch(x)
hyperbolicCosine:
    ; ряд у нас  $x^{(2n)}/(2n)!$ 
    finit
    fld qword[x]
    fmul st0,st0
    fst qword[tmpPow]
    fstp qword[tmpSquare]

    fld1
    fstp qword[chres]
    fild dword[two]
    fstp qword[tmpFactorial]

    ; начало цикла
    mov ecx, 3
hyperbolicCosine_loop:
    fld qword[tmpPow]
    fdiv qword[tmpFactorial]
    fadd qword[chres]
    fst qword[chres]

```

```

; мы тут сравниваем флоты как инты
mov eax, [chres+4]
cmp eax, 0x7ff00000
je hyperbolicCosine_fixInf
cmp eax, [prevchres+4]
jne hyperbolicCosine_cmpend
mov eax, [prevchres]
cmp eax, [chres]
je hyperbolicCosine_end
hyperbolicCosine_cmpend:
call debugLog
fstp qword[prevchres]

; обновляем tmpPow
fld qword[tmpPow]
fmul qword[tmpSquare]
fstp qword[tmpPow]

; обновляем tmpFactorial
fld qword[tmpFactorial]
mov [i], ecx
fimul dword[i]
inc ecx
mov [i], ecx
fimul dword[i]
inc ecx
fstp qword[tmpFactorial]

cmp ecx, 10000
jl hyperbolicCosine_loop
hyperbolicCosine_end:

ret

hyperbolicCosine_fixInf:
call debugLog
fld qword[prevchres]
fstp qword[chres]
ret

start:
call dramaticEntrance

cmp dword[argc], 1
jne start_sscanf

; тут мы для scanf-а написали что он создаёт float64
push askForX
call [printf]
add esp, 4
push x
push floatFormat_scanf
call [scanf]
add esp, 8
cmp eax, 1
jne start_wrongInput

jmp start_scanfEnd
start_sscanf:
push x
push floatFormat_scanf
mov eax, [argv]
mov eax, [eax+4]

```

```

push eax
call [sscanf]
add esp, 12
cmp eax, 1
jne start_wrongInput

start_scanfEnd:
call checkHyperbolicCosineBounds
call hyperbolicCosine

; printf умеет работать только с float64
; а push неумеет
push dword[chres+4]
push dword[chres]
push dword[x+4]
push dword[x]
push floatFormat_printf
call [printf]
add esp, 20

start_exit:
call gracefulExit
ret

start_wrongInput:
push wrongFromat
call [printf]
add esp, 4
jmp start_exit

section '.data' data readable writable
; тут всякие служебные переменные и строки
consoleWindowProcessId: dd 0
argv: dd 0
argc: dd 0
env: dd 0
exitStr: db 'ты открыли эту программу не из терминала(',10, \
'поэтому для выхода из неё тебе надо нажать Enter',10,0

; тут полезные переменные
floatFormat_scanf: db '%lf',0
floatFormat_printf: db 'ch(%g) = %.16g',10,0
debugFormat_printf: db '| %3i: %23.20g -> %23.20g',10,0
askForX: db 'От какого числа нам надо искать косинус? ',0
wrongFromat: db 'Надо ввести какое-то действительное число (через точку)',10,0
floatWrongFormat_printf: db 'Число %g слишком большое и посчитать косинус от него очень сложно',10,0

align 4
i: dd 0
two: dd 2 ; это буквально 2 потому что FPU не может читать литералы
hundred: dd 100 ; это буквально 100 потому что FPU не может читать литералы

; это всё float64
align 8
x: dq 0
tmpFactorial: dq 0
tmpPow: dq 0
tmpSquare: dq 0
chres: dq 0
prevchres: dq 0

section '.idata' import code readable
library msvcrt, 'msvcrt.dll', kernel32, 'kernel32.dll', user32, 'user32.dll'

```

```
import msvcrt, \
    printf, 'printf', scanf, 'scanf', sscanf, 'sscanf', \
    exit, '_exit', getch, '_getch', getmainargs, '__getmainargs'

import kernel32, \
    SetConsoleOutputCP, 'SetConsoleOutputCP', \
    GetConsoleWindow, 'GetConsoleWindow', \
    GetCurrentProcessId, 'GetCurrentProcessId'

import user32, GetWindowThreadProcessId, 'GetWindowThreadProcessId'
```