**Национальный исследовательский университет**

**«Высшая школа экономики»**

**Факультет компьютерных наук, ОП Программная инженерия (Б)**

Пояснительная записка

Наименование разработки: **программа для расчета приближенного значения функции cos(x) с помощью степенного ряда**

Работу выполнил студент группы **БПИ193(1) Андриевский Роман Дмитриевич**

Москва, 2020

1. Текст задания

Разработать программу, вычисляющую с помощью степенного ряда с точностью не хуже 0,1% значение функции cos(х) для заданного параметра x (использовать FPU)

1. Применяемые расчетные методы

При расчетах использовалось разложение функции в ряд Тейлора. Ряд Тейлора для cos(x):

Cos(x) = 1 – (x^2)/2! + (x^4)/4! + … + (-1)^n \* (x^(2n))/(2n)! + …

(Источник: курс Математического анализа)

1. Результаты тестирования на некоторых исходных данных:
2. X = 0

Cos(X) = 1.000000

Exact cos(X) = 1.000000

1. X = 1000000

Cos(X) = 0.936747

Exact cos(X) = 0.936752

1. X = -10000000000

Cos(X) = 0.872796

Exact cos(X) = 0.872798

1. X = 3.14159265

Cos(X) = -1.000000

Exact cos(X) = -1.000000

**Приложение 1. Текст программы**

format PE console

entry start

include 'win32ax.inc'

section '.code' code readable executable

start:

cinvoke printf, queryX

cinvoke scanf, floating, x

stdcall cos

cinvoke printf, outCos, dword[res], dword[res+4]

cinvoke printf, outExactCos, dword[t1], dword[t1+4]

call [getch]

push 0

call [ExitProcess]

proc cos

; Формула: cos x = 1 - x^2/(2)! + x^4/(4)! + ... + (-1)^n \* x^(2n)/(2n)! + ...

startcos:

finit

; чтобы ряд быстрее сходился, приведем x к диапазону [0;2pi]

; сначала "нацело" поделим x на 2pi

fld [x]

fdiv [pi2]

fstp [t]

movq xmm0, [t]

cvttpd2dq xmm1, xmm0

movq [t], xmm1 ; t = x // pi

; Приводим x к диапазону [0;2pi], x -= t\*2pi

fld [pi2]

fimul dword[t]

fld [x]

fsub st0, st1

fstp [x]

; цикл пока погрешность выше eps

lp:

finit

; Сохраняем текущее значение res в t, чтобы отслеживать погрешность

mov eax, dword[res]

mov dword [t], eax

mov eax, dword[res+4]

mov dword[t+4], eax

; res += (x^2n/2n!)\*(-1)^n

finit

fld [xn]

fdiv [fact]

fmul [mult]

fadd [res]

fstp [res]

finit

; проверяем, выполнено ли abs(res-t) < eps

; если выполнено - завершаем цикл, в res - приближенное значение косинуса

fld [t]

fld [res]

fsub st0, st1

fstp [t1]

fld [t1]

fld [eps]

fcomi st1

jb continue

fld [t1]

fld [negeps]

fcomi st1

ja continue

jmp endloop

continue:

finit

; вычисляем следующее значение x^2(n+1)

; xn = xn\*x\*x

fld [xn]

fmul [x]

fmul [x]

fstp [xn]

; вычисляем fact = (2(n+1))!

fld[fact]

fmul[factcnt]

fstp[fact]

fld[factcnt]

fld1

fadd st0, st1

fstp[factcnt]

fld [fact]

fmul[factcnt]

fstp[fact]

fld[factcnt]

fld1

fadd st0, st1

fstp[factcnt]

; меняем знак у множителя

fld[mult]

fmul[ng]

fstp [mult]

; идем на следующую итерацию

jmp lp

endloop:

finit

fld[x]

fsincos

fstp[t1]

endcos:

ret

endp

section '.data' data readable writeable

queryX db 'X = ', 0

outCos db 'Cos(X) = %lf',10, 0

outExactCos db 'Exact cos(X) = %lf',10, 0

integer db '%d', 0

floating db '%lf', 0

newline db 10,0

x dq ?

t dq 0.0

t1 dq 0.0

xn dq 1.0

fact dq 1.0

factcnt dq 1.0

mult dd 1.0

ng dq -1.0

res dq 0

pi dq 3.14159265358979323846

pi2 dq 6.28318530718

; eps и negeps нужны для оценки погресшности

eps dq 0.0001

negeps dq -0.0001

section '.idata' import data readable

library kernel, 'kernel32.dll',\

msvcrt, 'msvcrt.dll'

import kernel,\

ExitProcess, 'ExitProcess'

import msvcrt,\

printf, 'printf',\

scanf, 'scanf',\

getch, '\_getch'