**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут ім. Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота № 7**

з дисципліни

«Системне програмування»

на тему

**«Модульне програмування. Використання процедур.»**

Виконав: Перевірив:

Студент 2 курсу ФІОТ доц. Павлов В.Г.

Групи ІМ-13

Ситник Денис Олександрович

Номер у списку групи: 22

Київ – 2023

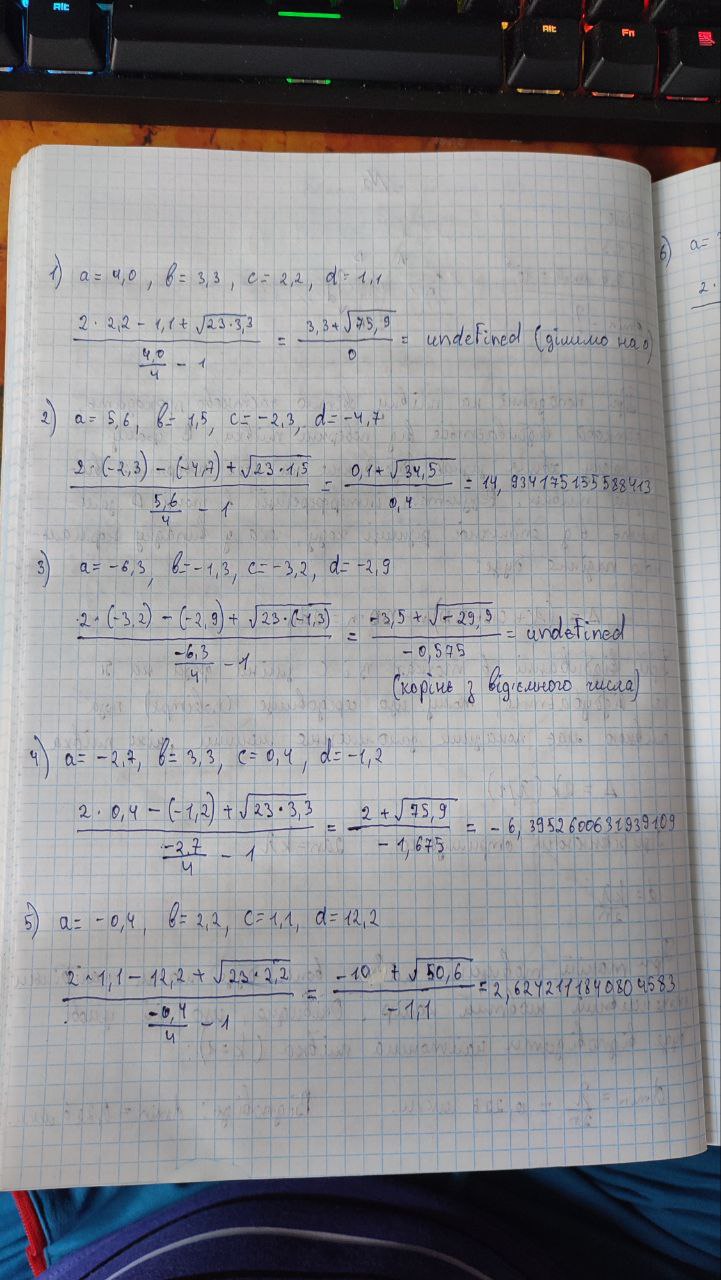
**Мета роботи**

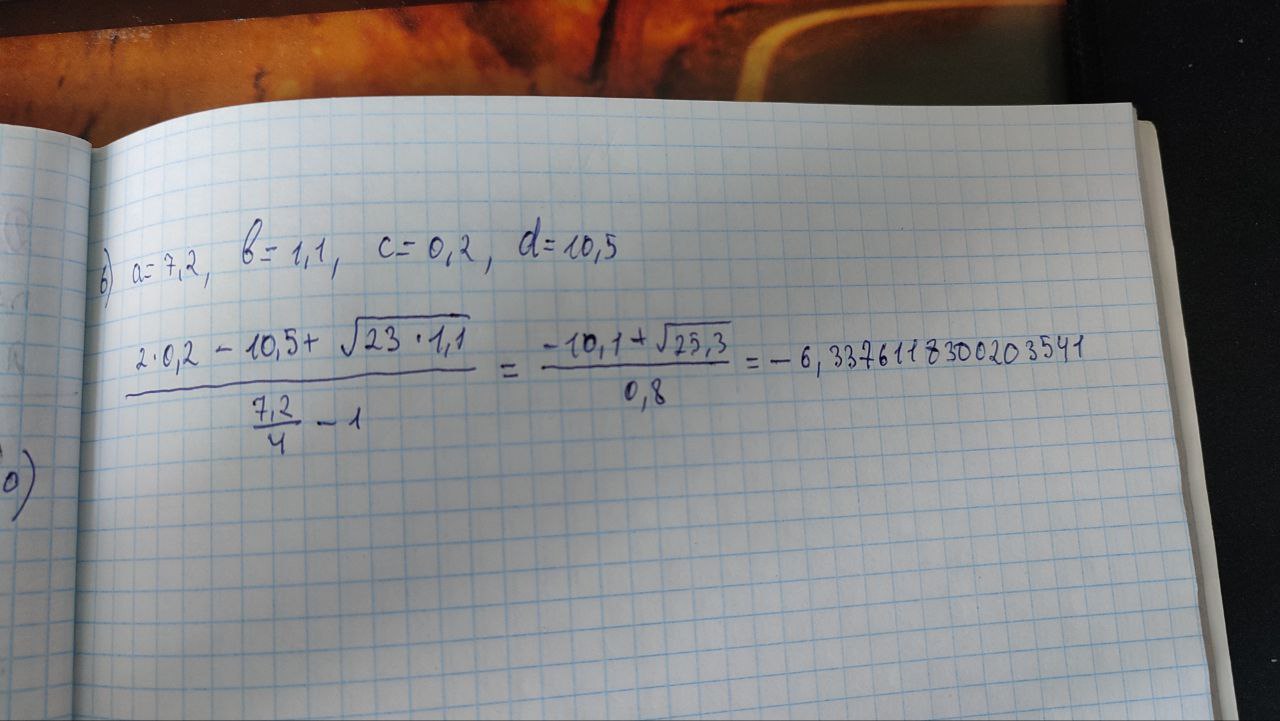
Вивчення прийомів модульного програмування, методів звернення до процедур і передачі в них параметрів.

Номер у списку – 22, варіант 1:



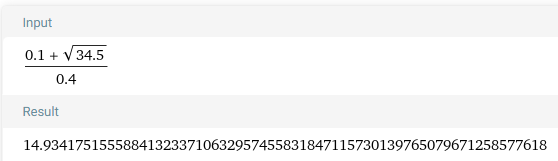
**Контрольні розрахунки:**

****

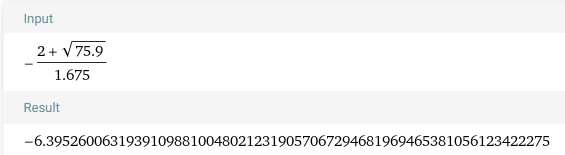
****

**Результати обчислень в інтернет-калькуляторі:**

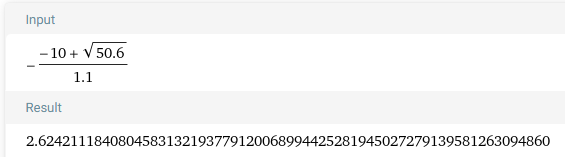
**2)**

****

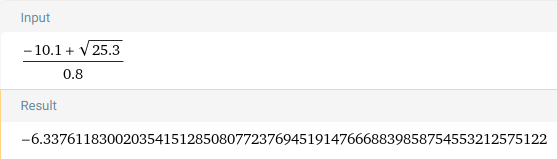
**4)**

****

**5)**

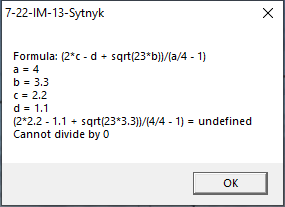
****

**6)**

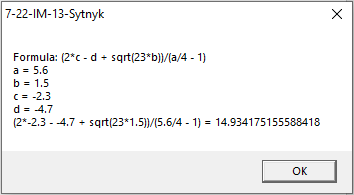
****

**Скріншоти виконання програми:**

Нуль у знаменнику:

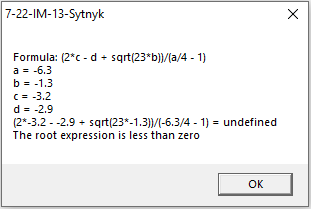


Додатнє на додатнє:

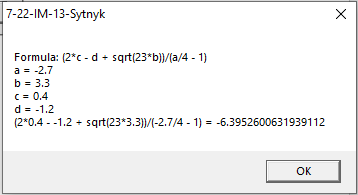


14,934175155588413 у контрольних розрахунках (відрізняється остання цифра)

Від’ємний підкореневий вираз:

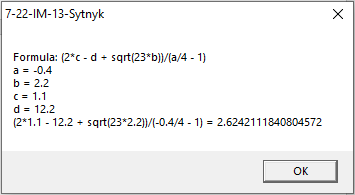


Додатнє на від’ємне:



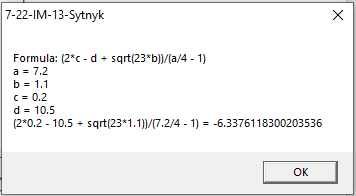
-6,3952600631939109 у контрольних розрахунках (відрізняються дві останні цифри)

Від’ємне на від‘ємне:



2,6242111840804583 у контрольних розрахунках (відрізняються дві останні цифри)

Від’ємне на додатнє



-6,3376118300203541 у контрольних розрахунках (відрізняються дві останні цифри)

**Ці відмінності пояснюються тим, що комп’ютер округлює числа при розрахунках.**

**Фійл лістингу:**

.386

.model flat, stdcall

include \masm32\include\masm32rt.inc

public valuesA, one, four

extern thirdProcedure:proto

coprocessorState macro

ftst

fnstsw ax

sahf

endm

successMessage macro

invoke wsprintf, addr buff, addr dataFormat,

addr valueA, addr valueB, addr valueC, addr valueD,

addr valueC, addr valueD, addr valueB, addr valueA, addr resultStr

invoke szCatStr, addr data, addr buff

invoke MessageBox, 0, offset data, offset labTitle, 0

endm

errorMessage macro messageFormat

invoke wsprintf, addr buff, addr messageFormat,

addr valueA, addr valueB, addr valueC, addr valueD,

addr valueC, addr valueD, addr valueB, addr valueA

invoke szCatStr, addr data, addr buff

invoke MessageBox, 0, offset data, offset labTitle, 0

endm

.data?

numerator dt 32 dup(?) ;; long double

denominator dt 32 dup(?) ;; long double

result dq 32 dup(?)

resultStr db 32 dup(?)

valueA db 32 dup(?)

valueB db 32 dup(?)

valueC db 32 dup(?)

valueD db 32 dup(?)

data db 32 dup(?)

buff db 32 dup(?)

.data

labTitle db "7-22-IM-13-Sytnyk", 0

dataFormat db "Formula: (2\*c - d + sqrt(23\*b))/(a/4 - 1)", 10,

"a = %s", 10,

"b = %s", 10,

"c = %s", 10,

"d = %s", 10,

"(2\*%s - %s + sqrt(23\*%s))/(%s/4 - 1) = %s", 10, 0

zeroDenominatorFormat db "Formula: (2\*c - d + sqrt(23\*b))/(a/4 - 1)", 10,

"a = %s", 10,

"b = %s", 10,

"c = %s", 10,

"d = %s", 10,

"(2\*%s - %s + sqrt(23\*%s))/(%s/4 - 1) = undefined", 10,

"Cannot divide by 0", 10, 0

invalidDefinitionAreaFormat db "Formula: (2\*c - d + sqrt(23\*b))/(a/4 - 1)", 10,

"a = %s", 10,

"b = %s", 10,

"c = %s", 10,

"d = %s", 10,

"(2\*%s - %s + sqrt(23\*%s))/(%s/4 - 1) = undefined", 10,

"The root expression is less than zero", 10, 0

valuesA dq 4.0, 5.6, -6.3, -2.7, -0.4, 7.2

valuesB dq 3.3, 1.5, -1.3, 3.3, 2.2, 1.1

valuesC dq 2.2, -2.3, -3.2, 0.4, 1.1, 0.2

valuesD dq 1.1, -4.7, -2.9, -1.2, 12.2, 10.5

one dq 1.0

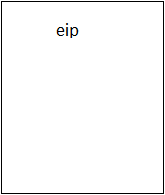
two dq 2.0

four dq 4.0

twentyThree dq 23.0

.code

firstProcedure proc



fld qword ptr[ebx] ;; 2 in stack

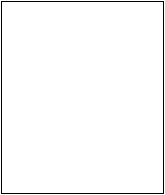
fld qword ptr[ecx + ebp \* 8] ;; c in stack

fmul ;; 2\*c

fld qword ptr[edx + ebp \* 8] ;; d in stack

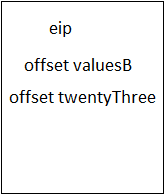
fsub ;; 2\*c - d

ret

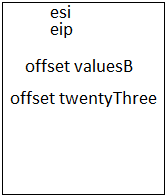


firstProcedure endp

secondProcedure proc



push esi



mov esi, esp

mov ebx, [esi + 8]

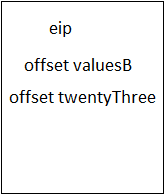
mov eax, [esi + 12]

fld qword ptr [ebx + ebp \* 8] ;; b in stack

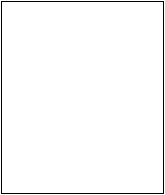
fmul qword ptr [eax] ;; b\*23

fsqrt ;; sqrt(23\*b)

pop esi



ret 12



secondProcedure endp

main:

mov ebp, 0

.while ebp < 6

;; Converting numbers to strings

invoke FloatToStr2, valuesA[ebp \* 8], addr valueA

invoke FloatToStr2, valuesB[ebp \* 8], addr valueB

invoke FloatToStr2, valuesC[ebp \* 8], addr valueC

invoke FloatToStr2, valuesD[ebp \* 8], addr valueD

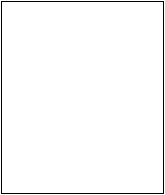
finit

mov eax, offset four

mov ebx, offset valuesA

;; Calculating denominator

call thirdProcedure



coprocessorState

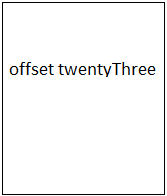
jz zeroDenominator

fstp denominator ;; saving denominator

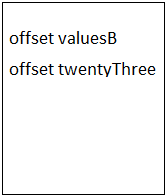
;; Calculating numerator

mov ebx, offset valuesB

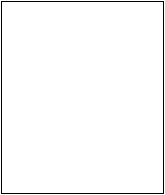
push offset twentyThree



push offset valuesB



call secondProcedure



coprocessorState

test ah, 01000000b

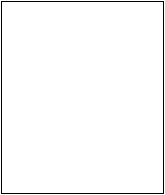
jnz invalidDefinitionArea

mov ebx, offset two

mov ecx, offset valuesC

mov edx, offset valuesD

call firstProcedure



fadd st(0), st(1)

fstp numerator ;; saving numerator

;; Preparing for division

fld tbyte ptr [numerator] ;; long double

fld tbyte ptr [denominator] ;; long double

fdivp st(1), st(0) ;; numerator/denominator

fstp qword ptr [result] ;; saving result in double format

invoke FloatToStr2, result, addr resultStr ;; converting result into string

successMessage

jmp next

zeroDenominator:

errorMessage zeroDenominatorFormat

jmp next

invalidDefinitionArea:

errorMessage invalidDefinitionAreaFormat

jmp next

next:

inc ebp

mov data, 0h

.endw

invoke ExitProcess, 0

end main

**Лістинг модуля:**

.386

.model flat, stdcall

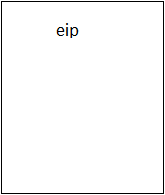
option casemap:none

extern valuesA:qword, one:qword, four:qword

public thirdProcedure

.code

thirdProcedure proc



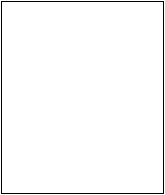
fld valuesA[ebp \* 8] ;; a in stack

fdiv four ;; a/4

fld one ;; 1 in stack

fsub ;; a/4 - 1

ret



thirdProcedure endp

end

**Висновок:**

Виконавши лабораторну роботу номер 7 я вивчив прийоми модульного програмування, методи звернення до процедур і передачі в них параметрів.   
Я використав на практиці три методи:   
- передача параметрів через регістри (перша процедура)

- передача параметрів через стек (друга процедура)

- передача параметрів через **public** та **extern**.

У кожному способі є як свої плюси, так і мінуси. Наприклад, хоча перший метод є найефективнішим у використанні ресурсів, він є обмеженим кількістю регістрів, на відміну від другого. У другому використовується стек – структура, яка працює за принципом FIFO (first in – first out), куди можна класти знатно більше елементів, ніж існує регістрів.

Особисто мені більше всього подобається третій, бо він декомпозує код, тобто розбиває його на маленькі шматочки, які можна використовувати будь-де у своїй програмі. Можна написати процедуру у окремому файлі, дати можливість за допомогою директиви **public** використовувати її у інших, і у тих файлах, куди її хочемо “імпортувати”, зробити це через **extern**.

Отже, усі три варіанти є дієвими, і використовувати їх треба залежно від ситуації.