**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут імені 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 групи ІМ-21 доцент

Лесько Дмитро Миколайович Павлов В.Г.

Номер у списку групи: 14

**Київ 2024**

**Мета роботи:** Вивчення прийомів модульного програмування, методів зве рнення до процедур і передачі в них параметрів.

**Виконання роботи:**

Як і сказано у завданні, я розробив код для обчислення значень рівняння, використовуючи наші змінні. Для передачі параметрів я використав усі методи, перечислені у вимогах до лабораторної роботи.

Макроси, формули і методи розріхунків я взяв зі своєї минулої лабораторної роботи, що пройшла перевірку. Переробив макроси під процедури, бо так легше маніпулювати програмую, де параметри передаються не на пряму і також це було вказано у вимогах. Використавши значення з минулої роботи, я побачив, що вони збігаються як з розрахунками, так і з контрольними прикладами, тому вважаю, що програма вдало рахує і відловлює критичні моменти у програмі, такі як ділення на 0 і порушення області визначення функцій.

**Детальні розрахунки контрольних прикладів:**

**Наша формула: ( 4 \* lg(c) - b/2 + 23)/(d - a + 1)**

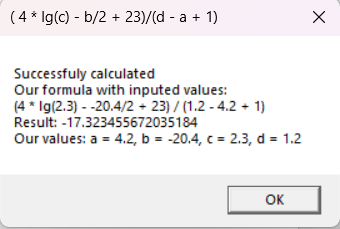
**Перший випадок**:

а = 4.2, б = -20.4, с = 2.3, d = 1.2

Розрахунок:

(4\*lg(2.3)+20.4/2+23)/(1.2-4.2+1)= (4\*0.3617278360175928+10.2+23)/(-2) = 34.64691134407037/-2 = -17.32345567203518

Скріншот з програми:



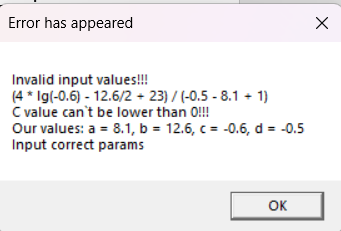
**Другий випадок**:

а = 8.1, б = 12.6, с = -0.6, d = -0.5

Розрахунок:

Цей випадок (приклад) демонструє вдалу валідацію вхідних параметрів нашою програмою. У цьому випадку підставлене число в логарифм менше 0, що порушує його область визначення, тому ми виводимо відповідне повідомлення

Скріншот з програми:



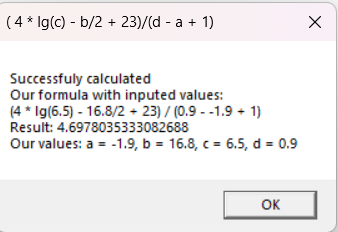
**Третій випадок**:

а = -1.9, б = 16.8, с = 6.5, d = 0.9

Розрахунок:

(4\*lg(6.5)-16.8/2+23)/(0.9+1.9+1)= (4\* 0.8129133566428555-8.4+23)/(3.8) = 17.85165342657142/3.8 = 4.697803533308269

Скріншот з програми:



\*Результат не збігається останніми числами, через округлення калькулятором

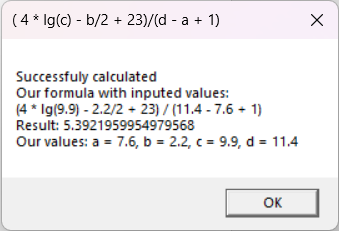
**Четвертий випадок:**

а = 7.6, б = 2.2, с = 9.9, d = 11.4

Розрахунок:

(4\*lg(9.9)-2.2/2+23)/(11.4-7.6+1)= (4\* 0.9956351945975499-1.1+23)/(4.8) = 25.8825407783902/4.8 = 5.392195995497958

Скріншот з програми:



**П’ятий випадок(з нулем у знаменнику):**

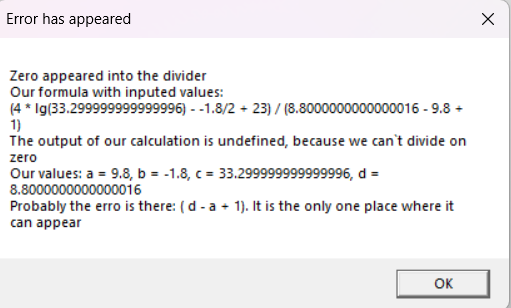
а = 7.6, б = 2.2, с = 9.9, d = 11.4

Розрахунок:

(4\*lg(33.3)+1.8/2+23)/(8.8-9.8+1)= 29.98977693402528/ 0 = Infinity?

У цьому випадку ми отримуємо 0 у знаменнику, тому програма не йде далі, а виводить повідомлення про виниклу помилку.

Скріншот з програми:



У такому випадку після виведення повідомлення ми пропускаємо цей цикл і йдемо далі рахувати числа.

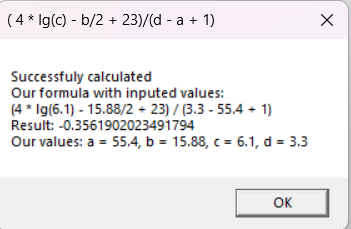
**Шостий випадок:**

а = 55.4, б = 15.88, с = 6.1, d = 3.3

Розрахунок:

(4\*lg(6.1)-15.88/2+23)/(3.3-55.4+1)= (4\* 0.7853298350107669-7.94+23)/( -51.1) = 18.20131934004307/-51.1 = -0.3561902023491794

Скріншот з програми:



**Лістенінг:**

**7-14-IM-21-Lesko.asm**

.386

.model flat, stdcall

option casemap :none

include \masm32\include\windows.inc

include \masm32\macros\macros.asm

include \masm32\include\kernel32.inc

include \masm32\include\masm32.inc

include \masm32\include\user32.inc

includelib \masm32\lib\masm32.lib

includelib \masm32\lib\user32.lib

includelib \masm32\lib\kernel32.lib

public Avalue, Dvalue, one

extern count\_divider: proto

.data?

    equationBuffer dd 512 dup(?)

    answer dq 8 dup(?)

    fix\_buffer dq 8 dup(?) ; buffer user for debugging the program and tracking the numbers

    divider dt 1 dup(?)

    shared dt 1 dup(?)

    sharedFirst dt 1 dup(?)

    sharedSecond dt 1 dup(?)

    Acurrent db 32 dup(?)

    Bcurrent db 32 dup(?)

    Ccurrent db 32 dup(?)

    Dcurrent db 32 dup(?)

    result db 64 dup(?)

.data

    zero\_cause\_msg\_box\_title db "Error has appeared", 0

    msg\_box\_title db "( 4 \* lg(c) - b/2 + 23)/(d - a + 1)", 0

    zero\_cause\_output\_msg\_box db "Zero appeared into the divider", 10, 13,

        "Our formula with inputed values: ", 10, 13,

        "(4 \* lg(%s) - %s/2 + 23) / (%s - %s + 1)", 10, 13,

        "The output of our calculation is undefined, because we can`t divide on zero", 10, 13,

        "Our values: a = %s, b = %s, c = %s, d = %s", 10, 13,

        "Probably the erro is there: ( d - a + 1). It is the only one place where it can appear", 0

    default\_output\_msg db "Successfuly calculated", 10, 13,

        "Our formula with inputed values: ", 10, 13,

        "(4 \* lg(%s) - %s/2 + 23) / (%s - %s + 1)", 10, 13,

        "Result: %s ", 10, 13,

        "Our values: a = %s, b = %s, c = %s, d = %s", 0

    c\_less\_than\_zero\_msg db "Invalid input values!!!", 10, 13,

        "(4 \* lg(%s) - %s/2 + 23) / (%s - %s + 1)", 10, 13,

        "C value can`t be lower than 0!!!", 10, 13,

        "Our values: a = %s, b = %s, c = %s, d = %s", 10, 13,

        "Input correct params", 0

    debug\_message db "Message is there: %s", 0

    Avalue dq 4.2, 8.1, -1.9, 7.6, 9.8, 55.4

    Bvalue dq -20.4, 12.6, 16.8, 2.2, -1.8, 15.88

    Cvalue dq 2.3, -0.6, 6.5, 9.9, 33.3, 6.1

    Dvalue dq 1.2, -0.5, 0.9, 11.4, 8.8, 3.3

    arraySize equ ($ - Avalue) / 32

    twentyThree dq 23.0

    ten dq 10.0

    four dq 4.0

    minus\_two dq -2.0

    one dq 1.0

    zero dq 0.0

.code

call\_msg\_window MACRO title:REQ, msg:REQ

    invoke MessageBox, 0, addr msg, addr title, 0

ENDM

count\_shared\_first PROC

**EIP**

    fld qword ptr [eax]

    fld qword ptr [ebx + esi \* 8]    ; Load c

    fyl2x                                 ; Calculate logarithm base 2 of c: log2(c)

    fldl2t                                ; log2(10)

    fdiv                                  ; Divide log2(c) by log2(10): log2(c) / log2(10) = lg(c)

    fld qword ptr [ecx]                  ; Load 4

    fmul                                  ; Multiply: 4 \* log10(c)

    fstp sharedFirst

    ret

count\_shared\_first ENDP

count\_shared\_second PROC

**EIP**

**Offset Bvalue**

**Offset minus\_two**

**Offset twentyThree**

    push ebp

**EBP**

**EIP**

**Offset Bvalue**

**Offset minus\_two**

**Offset twentyThree**

mov ebp, esp

    mov ebx, [ebp + 16]   ; 23

    mov ecx, [ebp + 12]   ; отримати адресу двійки (-2)

    mov edx, [ebp + 8]    ; barray

    fld qword ptr [edx + esi \* 8]   ; завантажити значення b

    fdiv qword ptr [ecx]  ; поділити b на -2

    fadd qword ptr [ebx]  ; відняти від результату значення twentyThree

    fstp sharedSecond ; зберегти результат в sharedSecond

    pop ebp

**EIP**

**Offset Bvalue**

**Offset minus\_two**

**Offset twentyThree**

    ret

count\_shared\_second ENDP

main:

    mov esi, 0

valueLoop:

    cmp esi, arraySize

    jge endLoop

    finit

    invoke FloatToStr2, [Avalue + esi \* 8], addr Acurrent

    invoke FloatToStr2, [Bvalue + esi \* 8], addr Bcurrent

    invoke FloatToStr2, [Cvalue + esi \* 8], addr Ccurrent

    invoke FloatToStr2, [Dvalue + esi \* 8], addr Dcurrent

    fld qword ptr [Cvalue + esi \* 8] ; Load the current value of C

    fldz                              ; Load 0

    fcom                               ; Compare C with 0

    fstsw ax                          ; Store the FPU status word in AX

    sahf                              ; Set the flags in EFLAGS register

    ; Test if the C0 (greater than) flag is set

    test ah, 00000001b               ; Check if bit 0 (C0 flag) is set

    jnz continueWithCode             ; If the C0 flag is set, jump to continueWithCode

    ; If C is less than or equal to 0, jump to errorNegativeC

    jmp errorNegativeC

    continueWithCode:

    call count\_divider    ; Calculate the denominator

    fstp divider

    mov eax, offset one

    mov ebx, offset Cvalue

    mov ecx, offset four

    call count\_shared\_first     ; Calculate the frist part: 4 \* log10(c)

    push offset twentyThree

**Offset twentyThree**

push offset minus\_two

**Offset minus\_two**

**Offset twentyThree**

    push offset Bvalue

**Offset Bvalue**

**Offset minus\_two**

**Offset twentyThree**

call count\_shared\_second ; calculate b/-2 + 23 (-2 because of later we add second and first, instead of substricting)

    fld sharedFirst

    fld sharedSecond

    fadd

    fstp shared

    fld qword ptr [divider]   ; Load the divider value

    ftst                      ; Compare the divider value with 0

    fnstsw ax                 ; Store the FPU status word in AX

    sahf                      ; Set the flags in EFLAGS register

    test ah, 01000000b       ; Check if bit 7 (C3 flag) is set

    jnz zeroDenominatorError ; If the C3 flag is not set, jump to error handling

    ; Perform the division and store the result

    fld shared

    fld divider

    fdiv

    fstp answer

    ; Format the output message

    invoke FloatToStr2, answer, addr result

    invoke wsprintf, addr equationBuffer, addr default\_output\_msg,

        addr Ccurrent, addr Bcurrent, addr Dcurrent, addr Acurrent,

        addr result,

        addr Acurrent, addr Bcurrent, addr Ccurrent, addr Dcurrent

    call\_msg\_window msg\_box\_title, equationBuffer

    jmp loopEnd

    errorNegativeC:

    ; Display error message for C, when it is lower than 0

    invoke wsprintf, addr equationBuffer, addr c\_less\_than\_zero\_msg,

        addr Ccurrent, addr Bcurrent, addr Dcurrent, addr Acurrent,

        addr Acurrent, addr Bcurrent, addr Ccurrent, addr Dcurrent

    call\_msg\_window zero\_cause\_msg\_box\_title, equationBuffer

    jmp loopEnd

    zeroDenominatorError:

    ; Display error message for division by zero

    invoke wsprintf, addr equationBuffer, addr zero\_cause\_output\_msg\_box,

        addr Ccurrent, addr Bcurrent, addr Dcurrent, addr Acurrent,

        addr Acurrent, addr Bcurrent, addr Ccurrent, addr Dcurrent

    call\_msg\_window zero\_cause\_msg\_box\_title, equationBuffer

loopEnd:

    add esi, 1

    jmp valueLoop

endLoop:

    invoke ExitProcess, NULL

end main

**7-14-IM-21-Lesko-thridpart.asm**

.386

.model flat, stdcall

option casemap :none

public count\_divider

extern Avalue:qword, Dvalue:qword, one:qword

.code

  count\_divider proc

      ; Calculate the denominator (d - a + 1)

      fld [Avalue + esi \* 8] ; a

      fld [Dvalue + esi \* 8] ; d

      fsub st(0), st(1) ; d - a

      fld one

      fxch st(1)

      fadd st(0), st(1) ; (d - a) + 1

      ret

  count\_divider endp

end

**7-14-IM-21-Lesko-thridpart.bat**

@echo off

ml /c /coff "7-14-IM-21-Lesko.asm"

ml /c /coff "7-14-IM-21-Lesko-thirdpart.asm"

link /subsystem:WINDOWS "7-14-IM-21-Lesko.obj" "7-14-IM-21-Lesko-thirdpart.obj"

7-14-IM-21-Lesko.exe

Як бачимо, у своєму коді я використав один макрос з минулої лабораторної роботи (№4), та переробив 2 макроси з минулої (№6), які я до цього взяв у лабораторній роботі №5. Тобто ідея програми залишається такою самою, змінюється цілі та реалізація. Для більш вдалої роботи програми я переписав макроси на proc функції, адже тепер нам не потрібно передавати параметри, як це зручно робиться у макросах, тому робимо це через стек, регістри та public-extern. Функцію, яка рахує чисельник я розділив на дві частини, щоб рахувати різними методами. Програма працює, адже результати з контрольними розрахунками співпадають. Виводяться помилки, при недозволених випадках (ділення на нуль і область визначення функції).

Для зручності я помітив коментарями відповідні етапи обрахунку по формулі.

Також стан стеку я позначав такими тепло жовтими квадратами:

**Висновок**

Виконавши цю лабораторну роботу я вивчив команди для обчислень з числами з плаваючою формулою. Розглянув різні методи обчислень з рівзних сторін (виведення формул логарифму) мови Assembler, та розробив власну програму, яка їх використовую. Також вивчив декілька методів передачі параметрів. Провівши контрольні розрахунки і порівнявши їх з отриманими результатами з програми, Я впевнився, що вона працює правильно та вдало рахує результат по вказаній у варіанті формулі. Також, протягом розробки, у мене виникали проблеми з точністю та вірністю розрахунків, тому я додатково «дебажив» і впевнювався у вірності значень чисельника та знаменника, то впевнений, що там встановлюються правильні значення. Всі результати співпадають. Була також проблема зі стеком та обчислень логарифму за основою десяти. Для цього я вивів формулу та додатково розібрався у правильності його роботи.

Загалом, завдяки вимогам я декомпозував формулу, що дозволило мені легше відслідковувати помилки в розрахунках та роботі програми. Досвід я вважаю позитивним і корисним.