



Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

## **Комп’ютерний практикум №3**

### **Системне програмне забезпечення**

**Тема:** Програмування розгалужених алгоритмів

Виконав

студент групи ІІІ-11:

Панченко С. В.

Перевірів:

Лісовиченко О. І

“7” березня 2023 р.

Київ 2023

## ЗМІСТ

1	Мета комп'ютерного практикуму.....	6
2	Завдання.....	7
3	Текст програми.....	8
4	Схема функціонування програми.....	17
5	Приклад виконання.....	24
6	Висновок.....	28
6.1	Команда безумовного переходу та її особливості.....	28
6.2	Команди умовного переходу.....	28
6.3	Команда порівняння СМР.....	28
6.4	Як здійснити умовний перехід на відстань, більшу за 128 байт?.....	29

# 1 МЕТА КОМП'ЮТЕРНОГО ПРАКТИКУМУ

Навчитись програмувати розгалужені алгоритми.

## 2 ЗАВДАННЯ

1. Можливість введення користувачем значень  $x$ ,  $y$ ,  $t$ ,  $a$ ,  $b$  за необхідності.
2. Обчислювати значення функції за введеними значеннями.
3. Виводити на екран результат обчислень.
4. Якщо є ділення, то результат дозволяється виводити:
  - як дійсне число (наприклад:  $= 1,666667$ ) – підвищена складність;
  - окремо цілу частину та остачу (наприклад:  $= 1$  остача  $2$ ) – середня складність;
  - окремо цілу частину та остачу як дріб – середня складність.
5. Програма повинна мати захист від некоректного введення вхідних даних (символи, переповнення, ділення на 0 і т.і.)





```
mov bl, byte [rax]

mov byte [rax], NEW_LINE_CHARACTER
mov rdi, 1

call WriteToConsole

mov byte [rax], bl
pop rbx
pop rdi
ret
;=====
<<
;=====
<<
ClearBuffer:
; Function clearing buffer
; void ClearBuffer(char* buffer, int length);
; Params:
;     rax:    char*   buffer
;     rdi:    int     length
; Returns:
;     void
push rcx
xor rcx, rcx

.loop:
cmp rcx, rdi
jbe .End
mov byte [rax + rcx], 0
jmp .loop

.End:
pop rcx
ret
;=====
<<
;=====
<<
TryConvertNumberToString:
; Function converting integer to string;
; bool TryConvertNumberToString(char* buffer, int bufferSize, int
inputtedLength, int number);
; Params:
;     rax:    char*   buffer
;     rdi:    int     bufferSize
;     rsi:    int&    inputtedLength
;     rdx:    int     number
; Returns:
;     r8:     bool    if true, no error, else thrown error
;
pushf
push rbx
push rcx
push r8
push r9

xor r9, r9
xor r8, r8
; counter = 0
mov rcx, 0
cmp rdx, 0
jge .ReadingNumbersIntoStack

.CheckForNegative:
mov r9, 1
neg rdx
mov byte [rax], MINUS_SIGN
```





```
<<
<<
WriteToConsole:
; Function writing string into STDOUT
; void WriteToConsole(char* buffer, int bufferSize)
; Params:
;     rax:    char*    buffer
;     rdi:    int      bufferSize
; Returns:
;     void
push rax
push rdi
push rsi
push rdx

mov rsi, rax
mov rdx, rdi
mov rax, SYS_WRITE
mov rdi, STDOUT

call DoSystemCallNoModify

pop rdx
pop rsi
pop rdi
pop rax
ret
<<
<<
ReadIntoBuffer:
; Function reading string into buffer;
; void ReadIntoBuffer(char* buffer, int bufferSize, int* inputtedLength);
; Params:
;     rax:    char*    buffer
;     rdi:    int      bufferSize
;     rsi:    int*     inputtedLength
; Returns:
;     void
push rax
push rdi
push rdx
push r8

mov qword [rsi], 0

mov r8, rsi

.loop:
    mov rsi, rax
    mov rdx, rdi
    mov rax, SYS_READ
    mov rdi, STDIN

    call DoSystemCallNoModify

    cmp rax, MAX_LENGTH
    jle .NoError

    mov rax, max_length_error
    mov rdi, max_length_error_length
    call WriteToConsole
    jmp .loop

.NoError:
```



```

jl .NoError

mov bl, byte [rax + r9]

; if(IsNeLineCharacter()) {
;     break;
; }
.IsNewLineCharacter:
    cmp bl, NEW_LINE_CHARACTER
    je .NoError

; if(sign=='-' || sign=='+') {
;     if(index!=0) {
;         return Error;
;     }
;     if(sign=='-') {
;         *number *= -1;
;     }
;
;     --rcx;
;     ++r9;
;     continue;
; }
.CheckForSigns:
    .IsPlusCharacter:
        cmp bl, PLUS_SIGN
        je .CheckForSignBeingFirst
    .IsMinusCharacter:
        cmp bl, MINUS_SIGN
        je .OnEqualMinus
    jmp .CallIsDigit
    .OnEqualMinus:
        mov r10, 1
    .CheckForSignBeingFirst:
        cmp r9, 0
        jne .ErrorSignNotFirst
    jmp .OnIterationEnd

.CallIsDigit
    push rax
    push rdi

    xor rax, rax
    mov al, bl
    call IsDigit
    mov r8, rdi

    pop rdi
    pop rax

cmp r8, 0
je .ErrorIncorrectSymbol
sub bl, DIGIT_ZERO

.CallPow
    push rax
    push rdi
    push rsi

    mov rax, 10
    mov rdi, rcx
    call Pow

    imul rsi, rbx
    mov rdi, rdx
    add rdi, rsi
    mov rdx, rdi

```





## 4 СХЕМА ФУНКЦІОНУВАННЯ ПРОГРАМИ

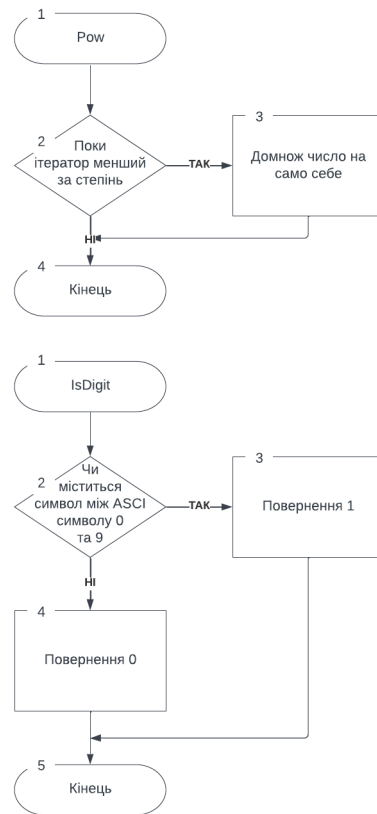


Рисунок 4.1 — схема функцій `Pow` та `IsDigit`

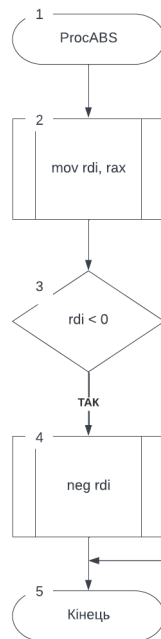


Рисунок 4.2 — схема функції `ProcABS`

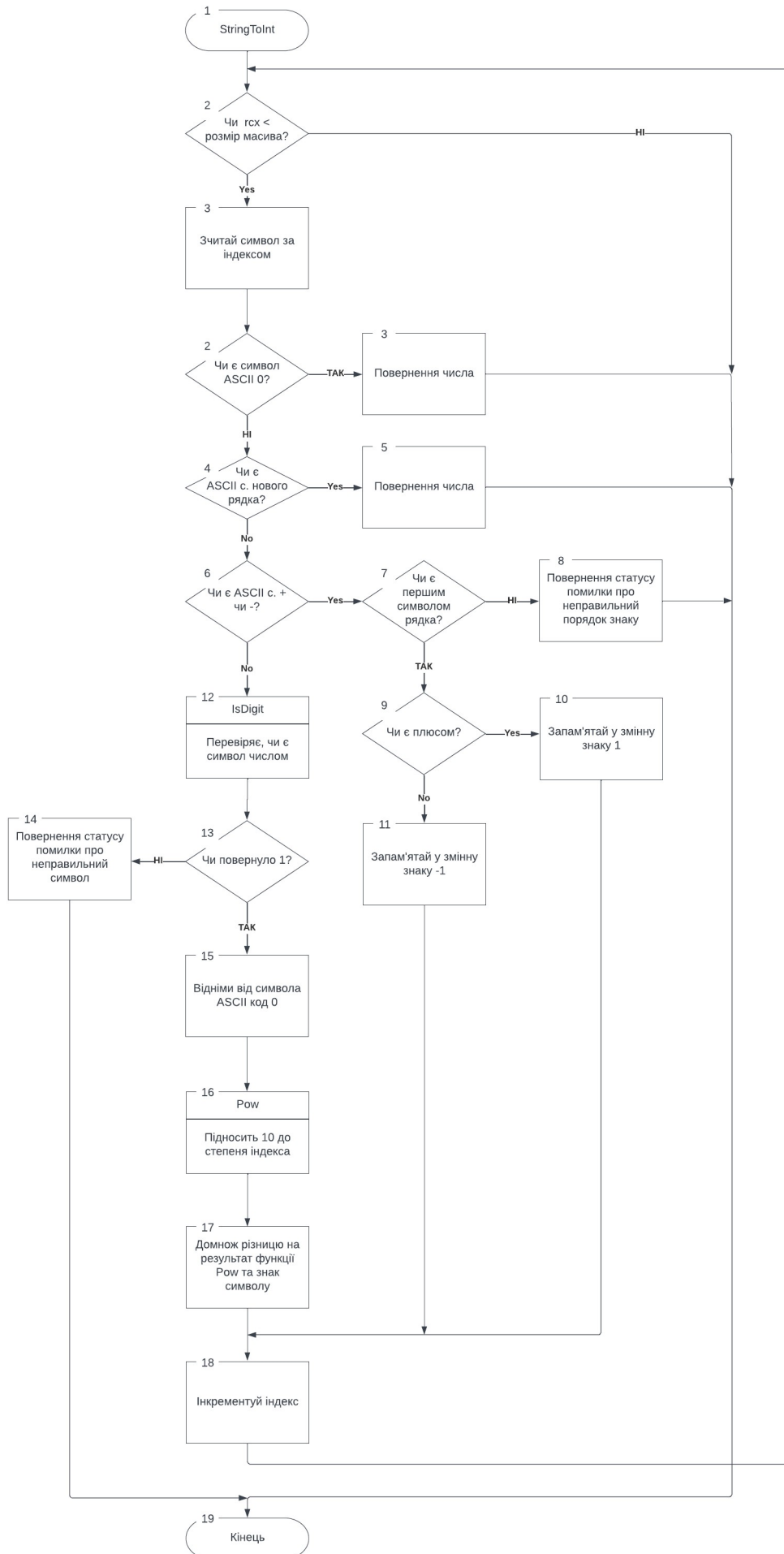


Рисунок 4.3 — схема функції StringToInt

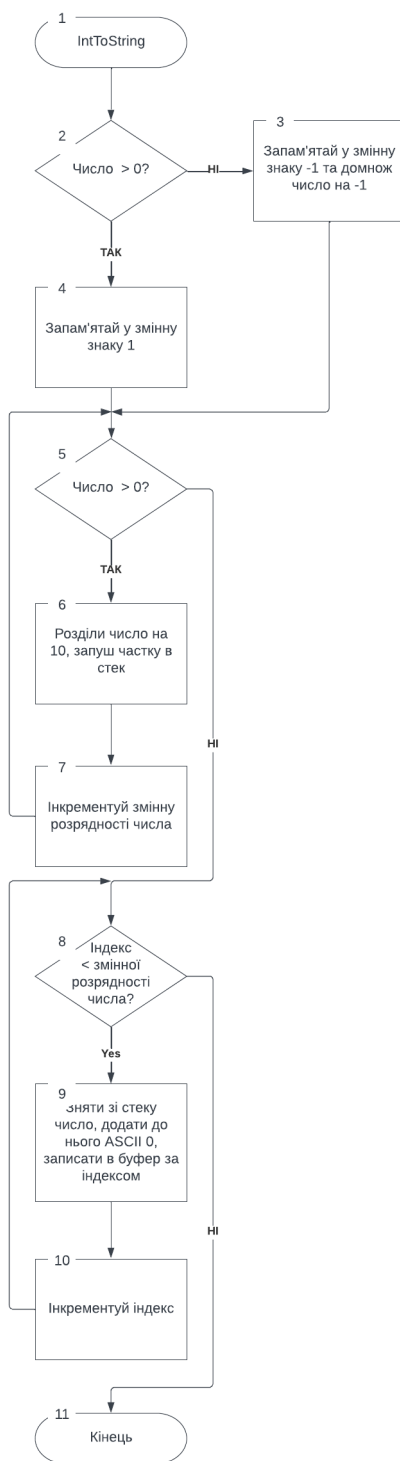


Рисунок 4.4 — схема функції IntToStr





Рисунок 4.5 — схема функції `InputArgument`

Рисунок 4.6 — схема функції `printFloat`

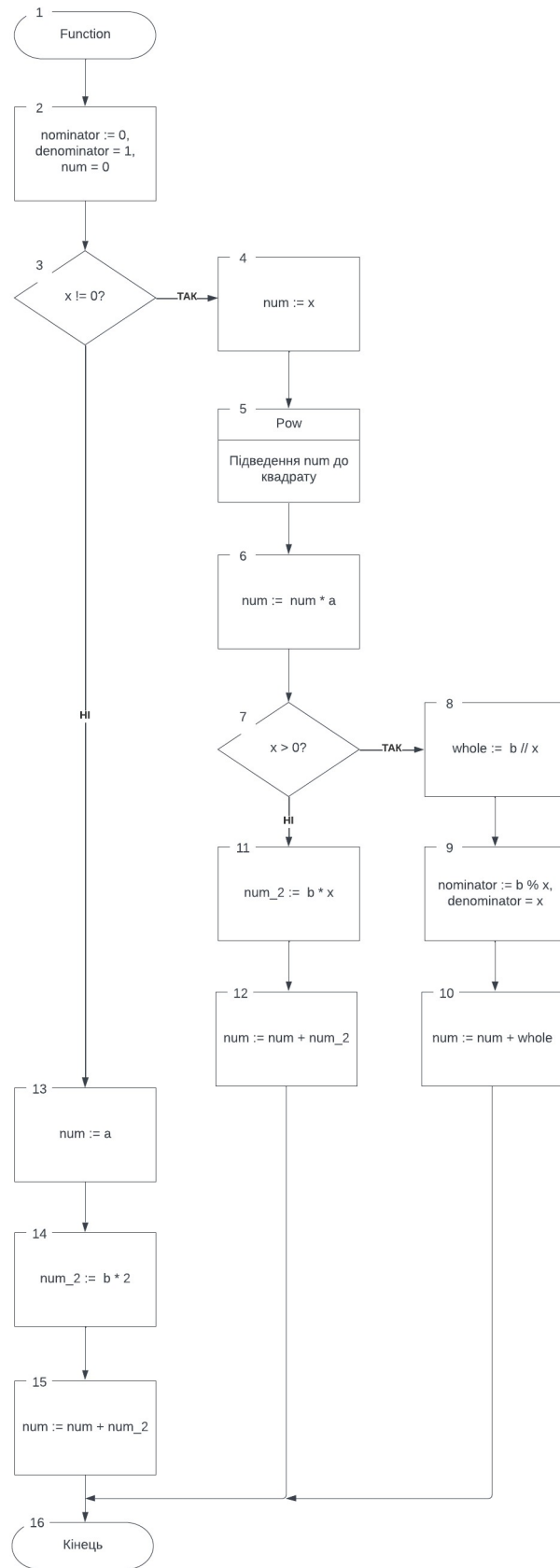


Рисунок 4.7 — схема функції Function

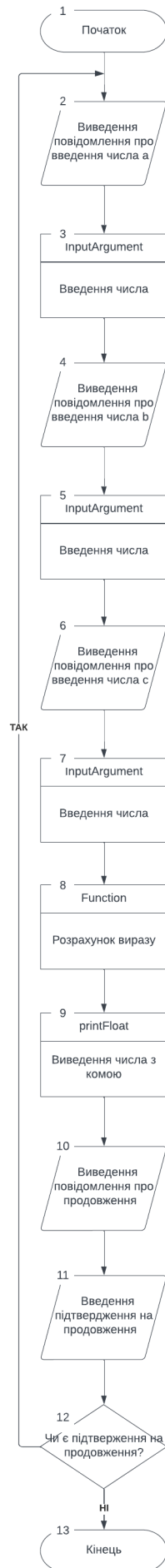


Рисунок 4.7 — схема функції Main

## 5 ПРИКЛАД ВИКОНАННЯ

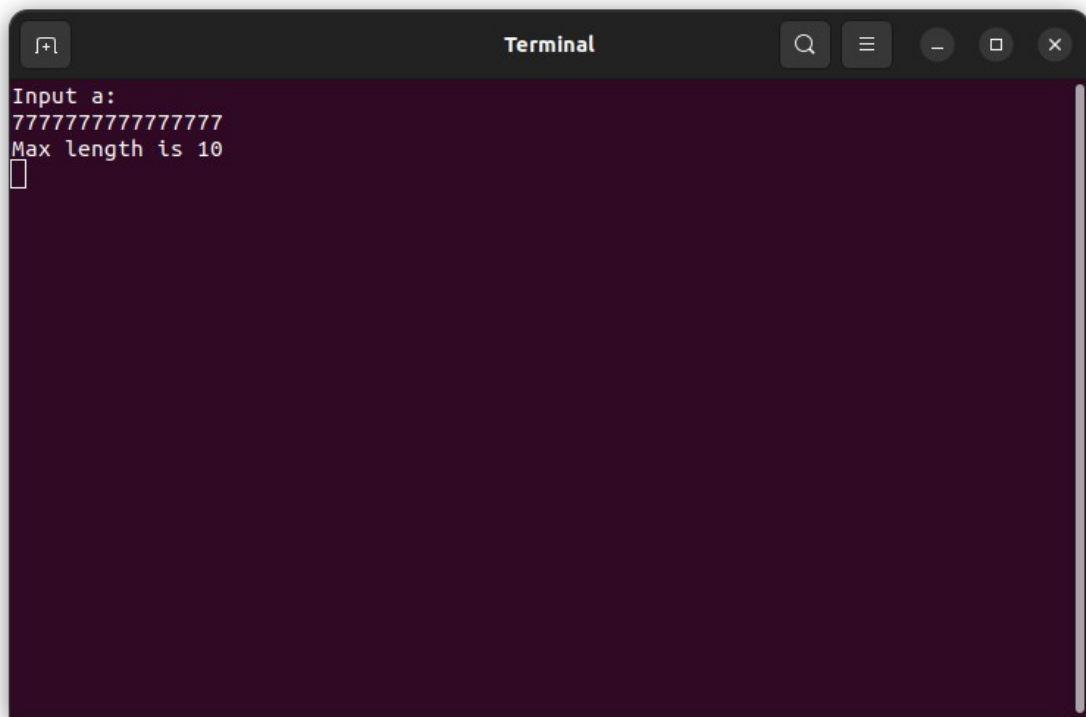


Рисунок 5.1 — виведення повідомлення про переповнення буфера

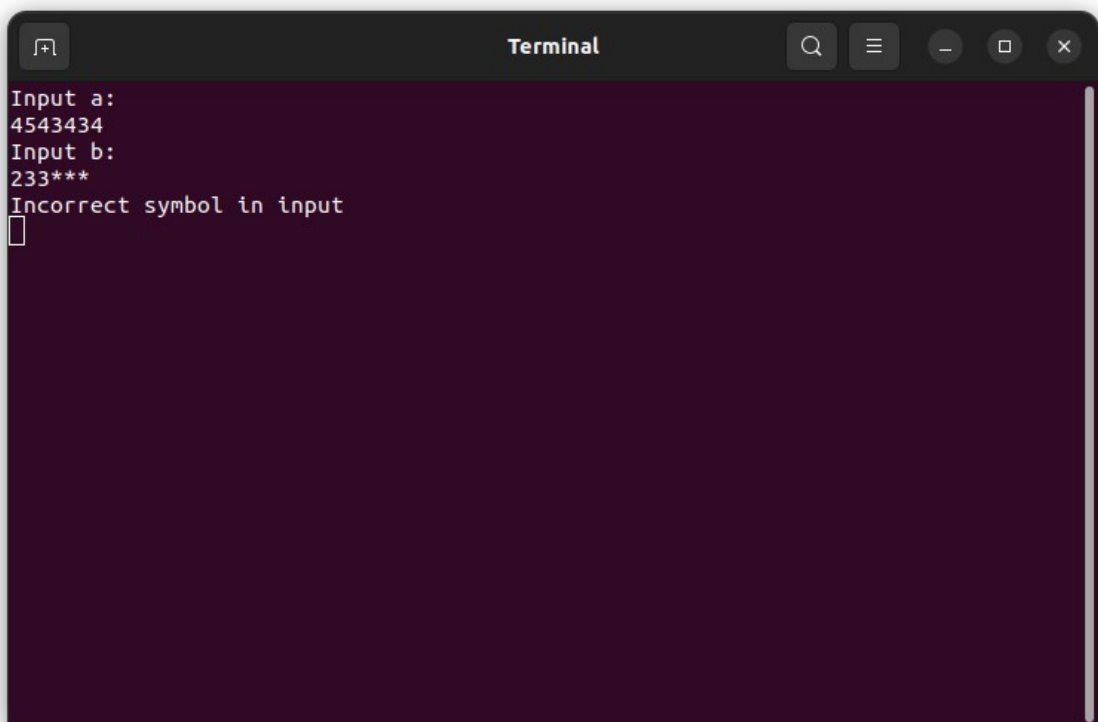


Рисунок 5.2 — виведення повідомлення про некоректний символ

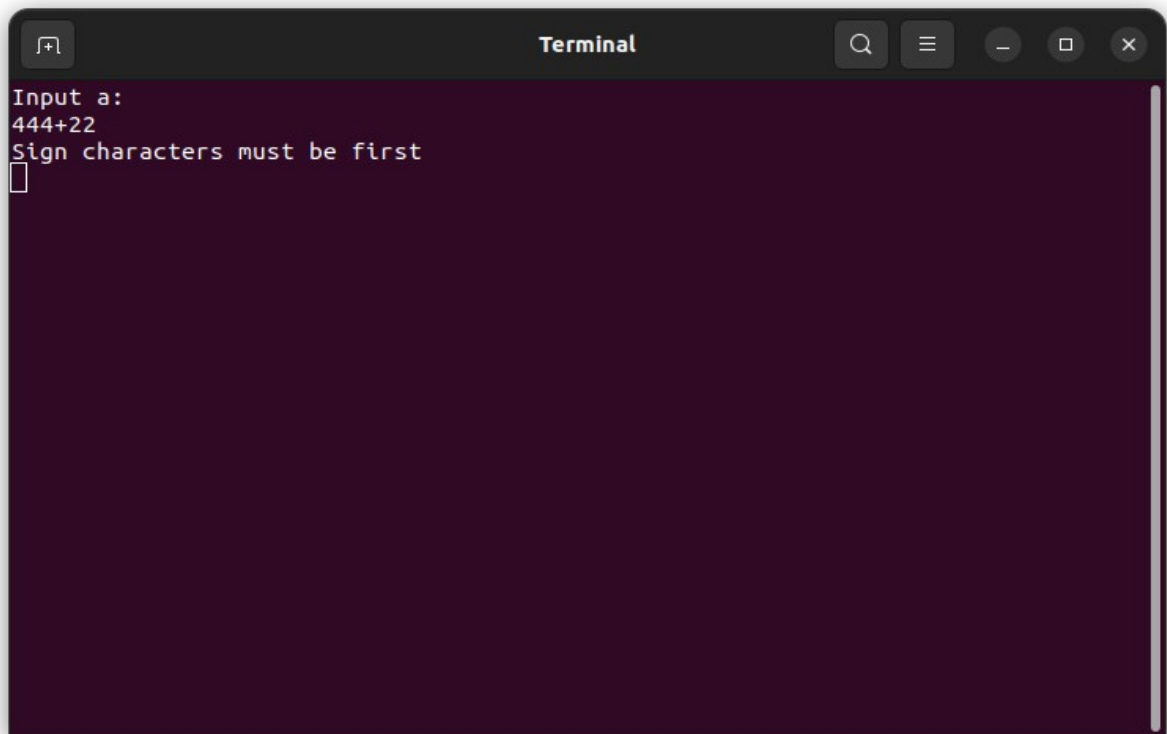


Рисунок 5.3 — виведення повідомлення про неправильну позицію знака

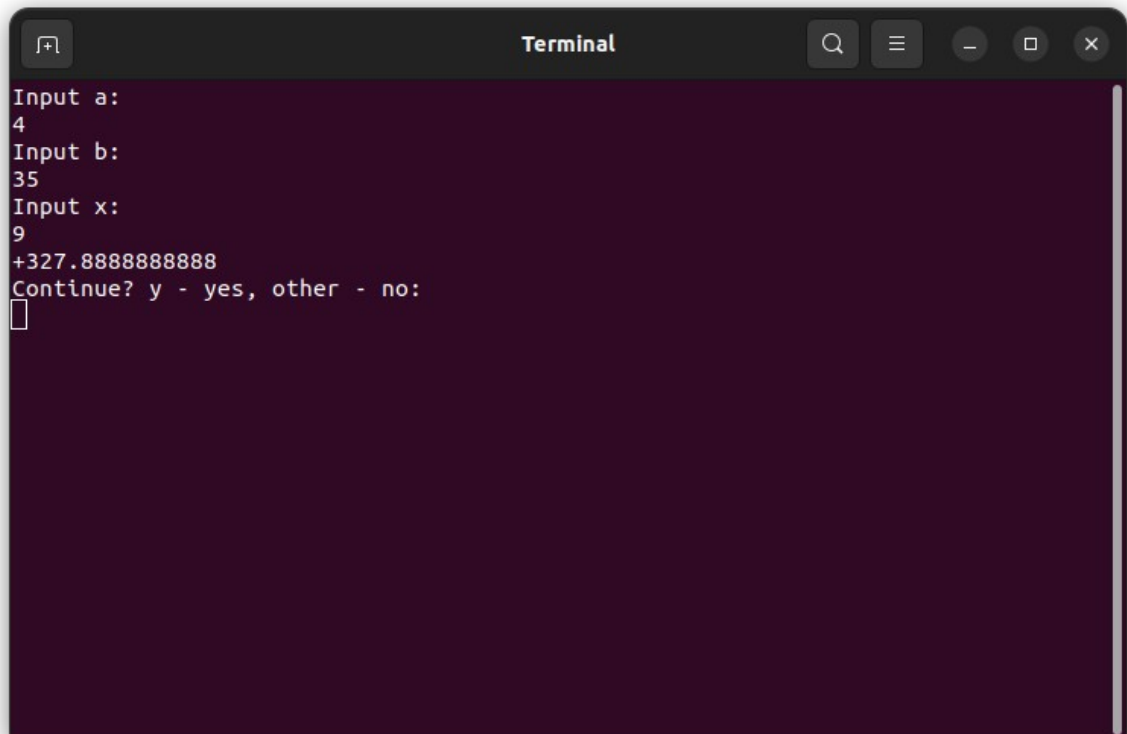
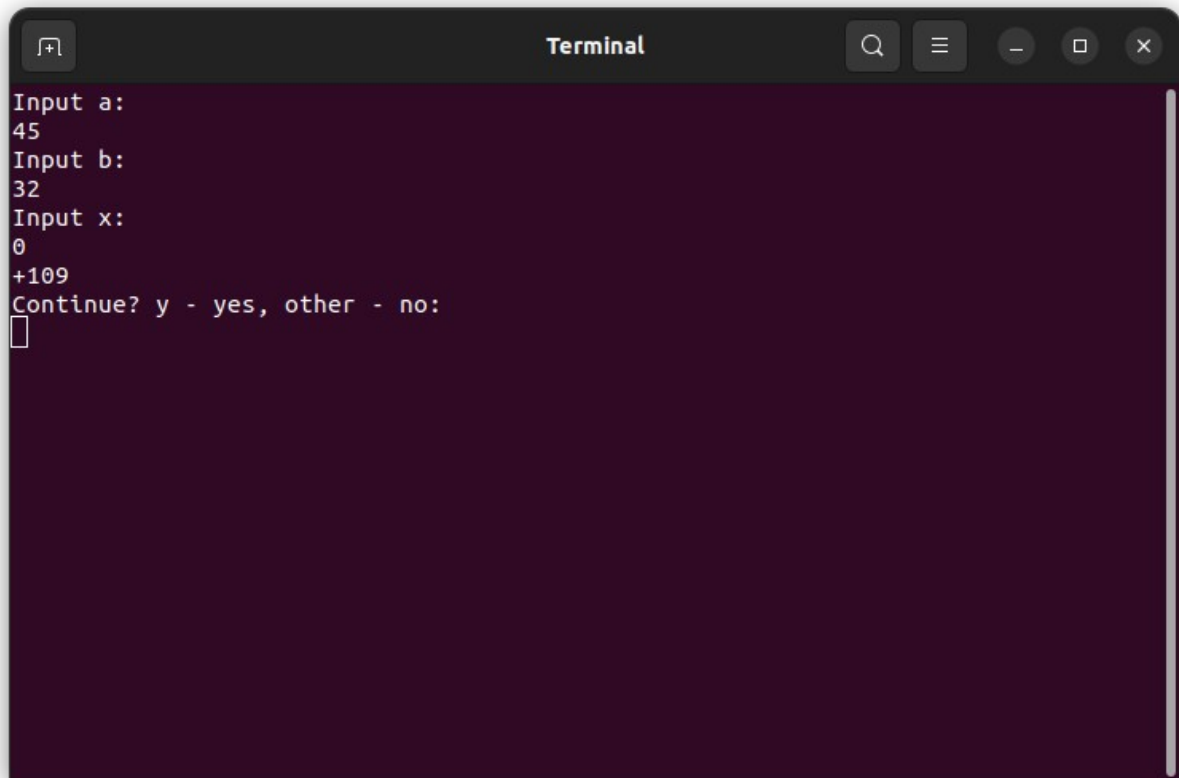


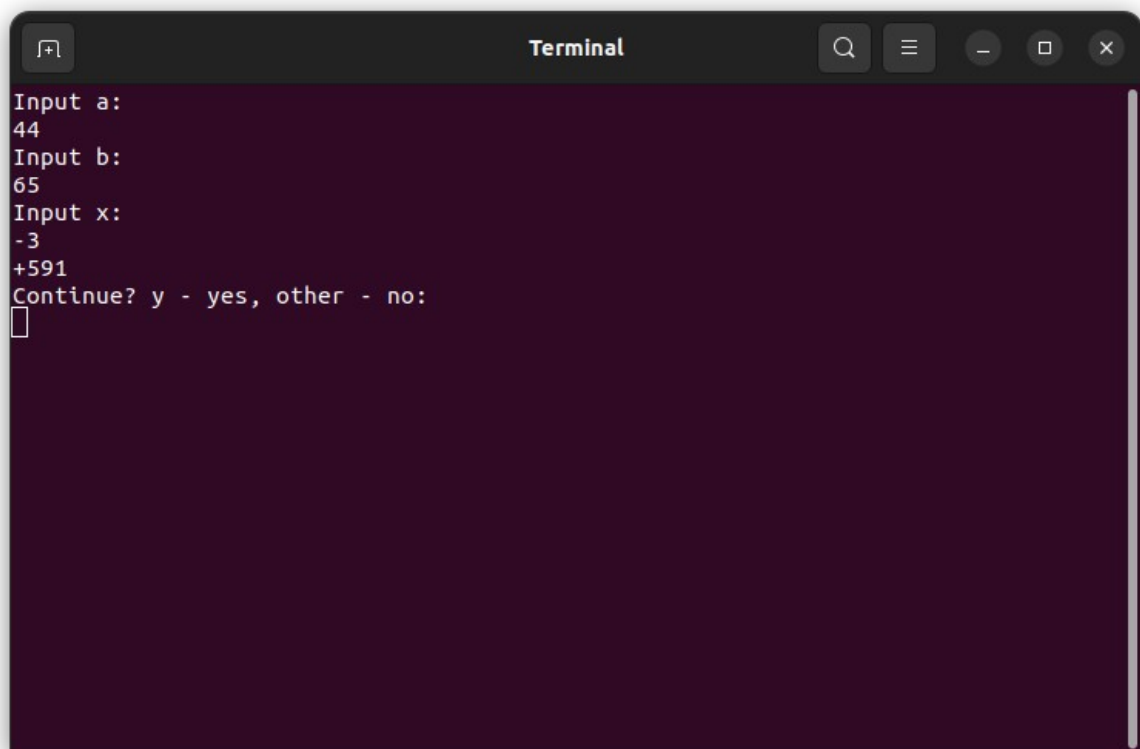
Рисунок 5.4 — виведення результату при  $x > 0$



```
Terminal
Input a:
45
Input b:
32
Input x:
0
+109
Continue? y - yes, other - no:

```

A terminal window titled "Terminal" with a dark background. It shows the execution of a program. The user enters '45' for 'Input a:', '32' for 'Input b:', and '0' for 'Input x:'. The program then outputs '+109'. Finally, it asks 'Continue? y - yes, other - no:' and the cursor is positioned at the start of the next line.

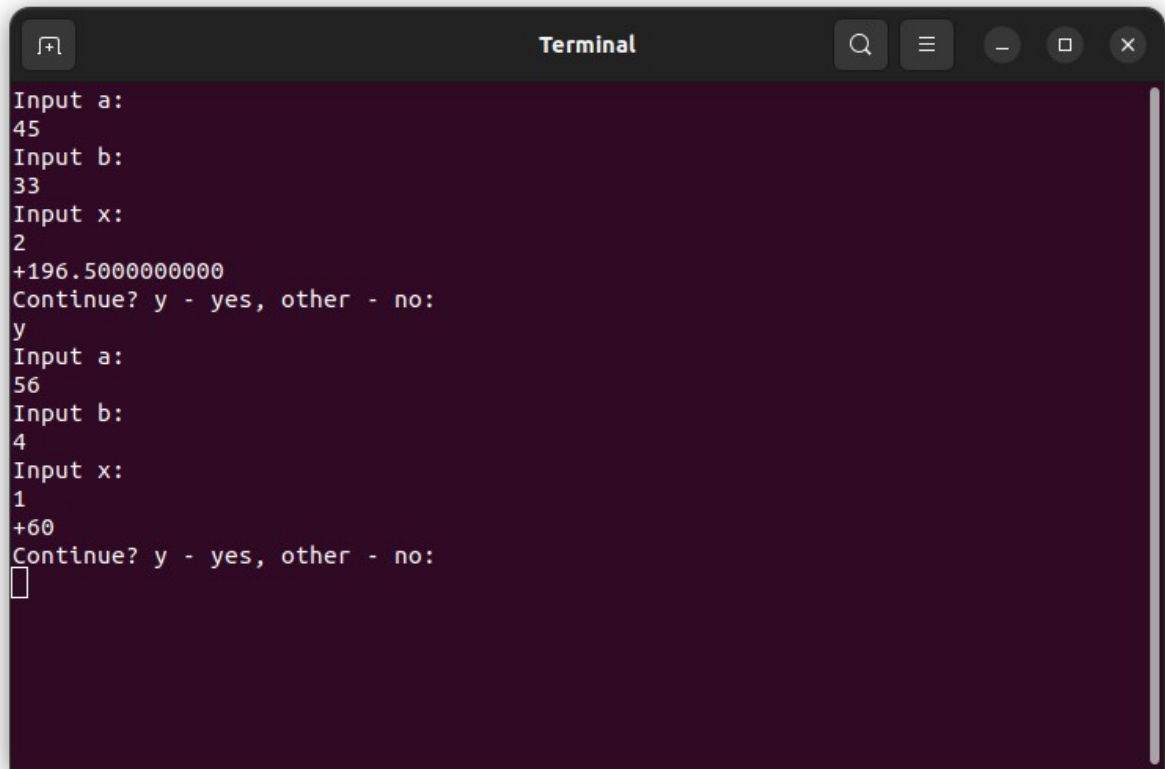
Рисунок 5.5 — виведення результату при  $x=0$ 

```
Terminal
Input a:
44
Input b:
65
Input x:
-3
+591
Continue? y - yes, other - no:

```

A terminal window titled "Terminal" with a dark background. It shows the execution of a program. The user enters '44' for 'Input a:', '65' for 'Input b:', and '-3' for 'Input x:'. The program then outputs '+591'. Finally, it asks 'Continue? y - yes, other - no:' and the cursor is positioned at the start of the next line.

Рисунок 5.6 — виведення результату при  $x<0$

A terminal window titled "Terminal" with a dark background and light text. It shows a sequence of inputs and outputs. The first loop takes inputs 45, 33, and 2, outputs +196.5000000000, and asks "Continue? y - yes, other - no:", with 'y' entered. The second loop takes inputs 56, 4, and 1, outputs +60, and asks the same question, with an empty box entered.

```
Input a:
45
Input b:
33
Input x:
2
+196.5000000000
Continue? y - yes, other - no:
y
Input a:
56
Input b:
4
Input x:
1
+60
Continue? y - yes, other - no:

```

Рисунок 5.7 — виведення повідомлення про запит на продовження



## 6 ВИСНОВОК

### 6.1 Команда безумовного переходу та її особливості

У NASM (Netwide Assembler) безумовні переходи — це інструкції, які передають керування іншій частині програми без будь-яких умов чи обмежень. Вони дозволяють програмі переходити до вказаної адреси пам'яті або мітки, незалежно від будь-яких попередніх інструкцій чи умов. Найпоширенішою інструкцією безумовного переходу в NASM є інструкція JMP. Ця інструкція приймає один операнд, який може бути адресою пам'яті або міткою, визначеною в програмі. Коли інструкція JMP виконується, процесор передасть управління в область пам'яті, визначену операндом. Безумовні переходи можуть бути корисними для реалізації циклів, реалізації викликів функцій або переходу до різних частин програми на основі введення користувача або інших умов. Однак вони також можуть ускладнити читання та налагодження коду, особливо якщо їх використовувати надмірно.

### 6.2 Команди умовного переходу

У NASM (Netwide Assembler) умовні переходи — це інструкції, які передають керування іншій частині програми на основі певної умови. Вони дозволяють програмі перевірити певну умову, а потім перейти до вказаної адреси пам'яті або мітки, якщо умова виконується. Умовні переходи корисні для реалізації потоку керування в програмах. Вони дозволяють програмі виконувати різні шляхи коду на основі значення регістра, розташування пам'яті чи іншої умови.

### 6.3 Команда порівняння CMP

У NASM (Netwide Assembler) інструкція CMP (порівняння) використовується для порівняння двох операндів і встановлення прапорів процесора на основі результату порівняння. Інструкція CMP не змінює жодного операнда, вона лише оновлює регістр прапорів на основі порівняння. Прапори, які встановлюються інструкцією CMP, залежать від результату віднімання. Якщо результат дорівнює нулю, встановлюється позначка нуля (ZF). Якщо результат негативний, встановлюється

позначка (SF). Якщо результат переповнюється, встановлюється прапор переповнення (OF). Якщо результат вимагає запозичення або перенесення, встановлюється прапор перенесення (CF).

#### 6.4 Як здійснити умовний перехід на відстань, більшу за 128 байт?

Якщо вам потрібно виконати умовний стрибок на відстань більше 128 байт, можна використовувати комбінацію інструкцій JMP і міток для створення багаторівневого переходу. Використовуючи кілька міток і інструкцій JMP, можливо ефективно переходити до інструкції, яка знаходиться далі, ніж 128 байт від поточної інструкції.

```
cmp eax, 1  
je section1
```

```
cmp eax, 2  
je section2
```

```
cmp eax, 3  
je section3
```

```
jmp end
```

```
section1:  
    jmp end
```

```
section2:  
    jmp end
```

```
section3:  
    jmp end
```

```
end:
```