

## Projekt 5.:

**„Program do mnożenia macierzy 3x3 argumenty pobierane z konsoli, wynik zapisywany do pliku tekstowego i wyświetlany na konsoli”**

**Jan Niedziółka**

### Spis treści:

1. Specyfikacja problemu. Przyjęte założenia i ograniczenia
2. Opis algorytmów oraz zmiennych
3. Kod programu
4. Instrukcja dla użytkownika
5. Przykładowe dane i wyniki

### **Specyfikacja problemu. Przyjęte założenia i ograniczenia**

Głównym zadaniem jest obliczenie iloczynu macierzy 3x3 z iloczynem skalarnym albo inną macierzą o takim samym rozmiarze. Zadanie zostało rozłożone na następujące podproblemy:

- pobranie z konsoli pierwszej macierzy reprezentowanej przez tablicę o dziewięciu elementach uzupełnianą przez użytkownika
- wybranie czy pierwsza macierz (ARR1) będzie mnożona przez skalar czy przez macierz
- podanie iloczynu skalarnego ALBO wczytanie drugiej macierzy (ARR2)
- wykonanie obliczeń zapisując od razu wyniki do wynikowej macierzy RES
- wyświetlenie macierzy RES na konsoli
- zapisanie jej do pliku tekstowego.

Dwa znaczące ograniczenia to:

- reprezentacja macierzy ww. postaci, generalnie nie wpływa to na wynik obliczeń, natomiast nie jest tak intuicyjne jak np. wykorzystanie dwuwymiarowej tablicy w językach wysokopoziomowych gdzie odnosimy się do elementu macierzy używając dwóch indeksów - w tym przypadku należy uważnie wykonywać obliczenia, ponieważ aby „odnieść” się do elementu trzeba najczęściej ręcznie przesunąć indeks

- ograniczenie się do liczb całkowitych.

Oczywistym ograniczeniem jest również maksymalny zakres czterobajtowych liczb, ale z uwagi na swoją wartość nie jest to istotne w praktyce.

## Opis algorytmów oraz zmiennych

|      |    |     |        |
|------|----|-----|--------|
| ARR1 | DD | 128 | DUP(?) |
| ARR2 | DD | 128 | DUP(?) |
| RES  | DD | 128 | DUP(?) |

Rys.1.

*Deklaracja wykorzystanych tablic.*

Główne zmienne to ARR1, ARR2 i RES. Poza nimi używane są również zmienne przechowujące wyświetlane na konsoli komunikaty (m.in. naglow, fillMat, podajSka) i zmienne. do obsługi plików (do przechowywania adresu czy nazwy utworzonego pliku).

```
wzor      DB      0Dh,0Ah,"%ld, %ld, %ld",0Dh,0Ah,"%ld, %ld, %ld",0Dh,0Ah,"%ld, %ld, %ld",0Dh,0Ah,0  ;%ld oznacza formatowanie w formacie dziesiętnym
```

Rys.2.

*Zmienna wzor posłuży do wyświetlenia wyniku na konsolę.*

Główne algorytmy to:

- alg. wypełniania macierzy,
- iloczyn skalarny,
- iloczyn dwóch macierzy,
- wypisanie wyniku na konsolę,
- utworzenie pliku z wynikiem.

```

166
167 ;--- wypełnienie Arr1
168     cld
169     mov EDI, OFFSET ARR1
170     mov ECX, 9
171     petla:
172     push ecx
173 ;--- czekanie na wprowadzenie znaków, koniec przez Enter ---
174     push 0 ; rezerwa, musi być zero
175     push OFFSET rinp ; wskaźnik na faktyczną liczbę wprowadz
176     push rbuf ; rozmiar bufora
177     push OFFSET bufor ; wskaźnik na bufor
178     push hinp ; deskryptor buforu konsoli
179     call ReadConsoleA ; wywołanie funkcji ReadConsoleA
180     lea EBX, bufor
181     mov EDX, rinp
182     mov BYTE PTR [EBX+EDX-2], 0; zero na końcu tekstu
183
184 ;--- przekształcenie A
185     push OFFSET bufor
186     call ScanInt ; pobraną liczbę mamy teraz w akumulator
187     stosd ; i przesyłamy ją do pamięci w miejsce ws
188     pop ecx
189     loop petla

```

Rys.3.

### Wypełnienie macierzy ARR1.

W rejestrze EDI przechowano adres zmiennej ARR1, w pętli (wykonywanej 9 razy) odczytano podawane w konsoli znaki używając procedurę ReadConsoleA i zapisywano je w buforze. Znaki przechowywane w buforze wkładano na stos, z którego z użyciem stosd zapisano je na komórki pamięci wskazywane przez rejestr EDI (adres ARR1 i zmiana co 4 z każdym powtórzeniem).

```

push    OFFSET bufor
call    ScanInt
mov     skalar, EAX

cld
mov     esi, offset arr1
mov     edi, offset res
mov     ecx, 9
L1:
    lodsd
    mul  skalar
    stosd
loop  L1

```

Rys.4.

*Mnożenie ARR1 przez skalar i zapisywanie wyników w RES.*

Po ustawieniu flagi kierunku (cld) ładowano wartości z pamięci o adresie wskazywanym przez ESI (a więc wartości z ARR1) przy użyciu lodsd, mnożono poprzez  
mul skalar  
i zapisywano wynik (przechowywany w EAX) do miejsca w pamięci wskazywanego przez EDI (czyli kolejnych „znaków” tablicy RES).

```

mov     ESI, offset ARR1
mov     EDI, offset ARR2
MOV     ecx, 3
MOV     edx, 0
Sum00:
mov     EAX, [ESI]
imul    EAX, [EDI]
add     EDX, EAX
add     ESI, 4
add     EDI, 12
loop    Sum00
mov     [EBX], EDX

```

Rys.5.

*Mnożenie macierzy.*

*Po wczytaniu macierzy ARR2 i przekazaniu adresów trzech tablic zaczęto wymnażać przez siebie kolejne elementy każdego rzędu ARR1 z elementami*

każdej kolumny ARR2:

|      |         |         |        |          |   |          |          |   |   |
|------|---------|---------|--------|----------|---|----------|----------|---|---|
|      | ARR1[0] | ARR1[4] | ARR[8] | ARR1[12] |   | ARR1[24] | ARR1[32] |   |   |
| ARR1 | 1       | 2       | 3      | 4        | 5 | 6        | 7        | 8 | 9 |

|      |         |         |         |         |   |         |          |   |   |
|------|---------|---------|---------|---------|---|---------|----------|---|---|
|      | ARR2[0] | ARR2[4] | ARR2[8] | ARR[12] |   | ARR[24] | ARR2[32] |   |   |
| ARR2 | 1       | 2       | 3       | 4       | 5 | 6       | 7        | 8 | 9 |

|                       |
|-----------------------|
| pierwszy rząd/kolumna |
| drugi rząd/kolumna    |
| trzeci rząd/kolumna   |

|      |   |   |   |                   |   |   |
|------|---|---|---|-------------------|---|---|
|      |   |   |   | ARR2              |   |   |
|      |   |   |   | 1                 | 2 | 3 |
|      |   |   |   | 4                 | 5 | 6 |
|      |   |   |   | 7                 | 8 | 9 |
| ARR1 | 1 | 2 | 3 | $1*1 + 2*4 + 3*7$ |   |   |
|      | 4 | 5 | 6 |                   |   |   |
|      | 7 | 8 | 9 |                   |   |   |

RES



|      |   |   |   |      |     |     |
|------|---|---|---|------|-----|-----|
|      |   |   |   | ARR2 |     |     |
|      |   |   |   | 1    | 2   | 3   |
|      |   |   |   | 4    | 5   | 6   |
|      |   |   |   | 7    | 8   | 9   |
| ARR1 | 1 | 2 | 3 | 30   | 36  | 42  |
|      | 4 | 5 | 6 | 66   | 81  | 96  |
|      | 7 | 8 | 9 | 102  | 126 | 150 |

RES

w szesnastkowym:

|    |    |    |
|----|----|----|
| 1E | 24 | 2A |
| 42 | 51 | 60 |
| 66 | 7E | 96 |

RES

Address: 0x00C34530

```

0x00C34530 1e 00 00 00 24 00 00 00 2a 00 00 00 42 00 00 00 51 00 00 00 60 00 00 00 66 00 00 00 7e 00 00 00 96 00 00 00
0x00C3453F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00C3454E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00C345BD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Memory 1 Registers

main.asm

```

441      mov [EBX+32], EDX
442
443
444      koniec:
445      ;--- wyprowadzenie wyniku obliczeń ---
446      push RES[32] ≤ 1ms elapsed
447      push RES[28]

```

Rys.6.

Ogólny schemat sposobu obliczania kolejnych elementów RES wraz z dołączoną zawartością pamięci potwierdzającą poprawność wyników.

Obliczenia takie jak na Rys.5. wykonano 9 razy, za każdym razem przesuwając indeks tablicy o odpowiednią wartość:

```

mov ESI, offset ARR1+24
mov EDI, offset ARR2+4
MOV ecx, 3
MOV edx, 0
Sum21:
mov EAX, [ESI]
imul EAX, [EDI]
add EDX, EAX
add ESI, 4
add EDI, 12
loop Sum21
mov [EBX+28], EDX

```

Rys.7.

*„Sum21” wskazuje na to, że obliczany jest element w trzecim rzędzie, drugiej kolumnie macierzy RES. Aby się do niego dostać należało przenieść indeks tablicy ARR1 o 24 pola (6 elementów po 4 bajty każdy) i ARR2 o 4 pola (2 kolumna, a więc 1 czterobajtowy element w prawo).*

Te obliczenia wykonywano aż 9 razy, ponieważ z uwagi na konieczność pilnowania kilku niezależnych zmiennych/indeksów próba stworzenia jednej uniwersalnej pętli doprowadziłaby do znacznego skomplikowania kodu, ograniczenia przejrzystości (wiele etykiet i skoków, spaghetti code) oraz trudności w implementacji. W ten sposób udało się zachować schludny, czytelny kod o łatwej weryfikowalności.

```

koniec:
;--- wyprowadzenie wyniku obliczeń ---
push    RES[32]
push    RES[28]
push    RES[24]
push    RES[20]
push    RES[16]
push    RES[12]
push    RES[8]
push    RES[4]
push    RES[0]
push    OFFSET wzor
push    OFFSET bufor
call    wsprintfA           ; zwraca liczbę
add     ESP, 12             ; czyszczenie s
mov     rinp, EAX           ; zapamiętywani
;--- wyświetlenie wyniku -----
push    0                  ; rezerwa, musi
push    OFFSET rout        ; wskaźnik na f
push    rinp               ; liczba znaków
push    OFFSET bufor       ; wskaźnik na t
push    hout               ; deskryptor bu
call    WriteConsoleA      ; wywołanie fun

```

Rys.8.

Wyprowadzenie wyniku obliczeń (elementów macierzy RES) na konsolę – wykorzystano zmienną z Rys.2. i nieco rozbudowany kod z laboratorium 4. Zawartość zmiennej bufor zostanie wykorzystana w następnym kroku.

```

481
482     push OFFSET pliktxt
483     push OFFSET adresdat
484     call lstrcata                                ;łączenie dwóch łańcuchów()
485
486     push 0
487     push 0
488     push CREATE_ALWAYS
489     push 0
490     push 0
491     push GENERIC_WRITE OR GENERIC_READ
492     push OFFSET adresdat
493     call CreateFileA
494     mov hfile, EAX
495
496     push offset bufor
497     call lstrlenA
498
499     push 0
500     push OFFSET rout
501     push EAX
502     push OFFSET bufor
503     push hfile
504     call WriteFile
505
506     push hfile
507     call CloseHandle

```

Rys.9.

*Utworzenie pliku tekstowego z zawartością macierzy RES.*

Najpierw wykorzystano procedury do obsługi plików aby uzyskać ścieżkę pliku (zmienna adresdat), następnie użyto procedur CreateFileA (stworzenie pliku tekstowego) i WriteFile (zapisywanie do pliku zawartości zmiennej bufor, EAX – liczba znaków do wypisania, rout – rzeczywista liczba wpisanych znaków).

### Kod programu

Kod programu zostanie umieszczony na Strefie lub wysłany mailowo prowadzącemu laboratoria.

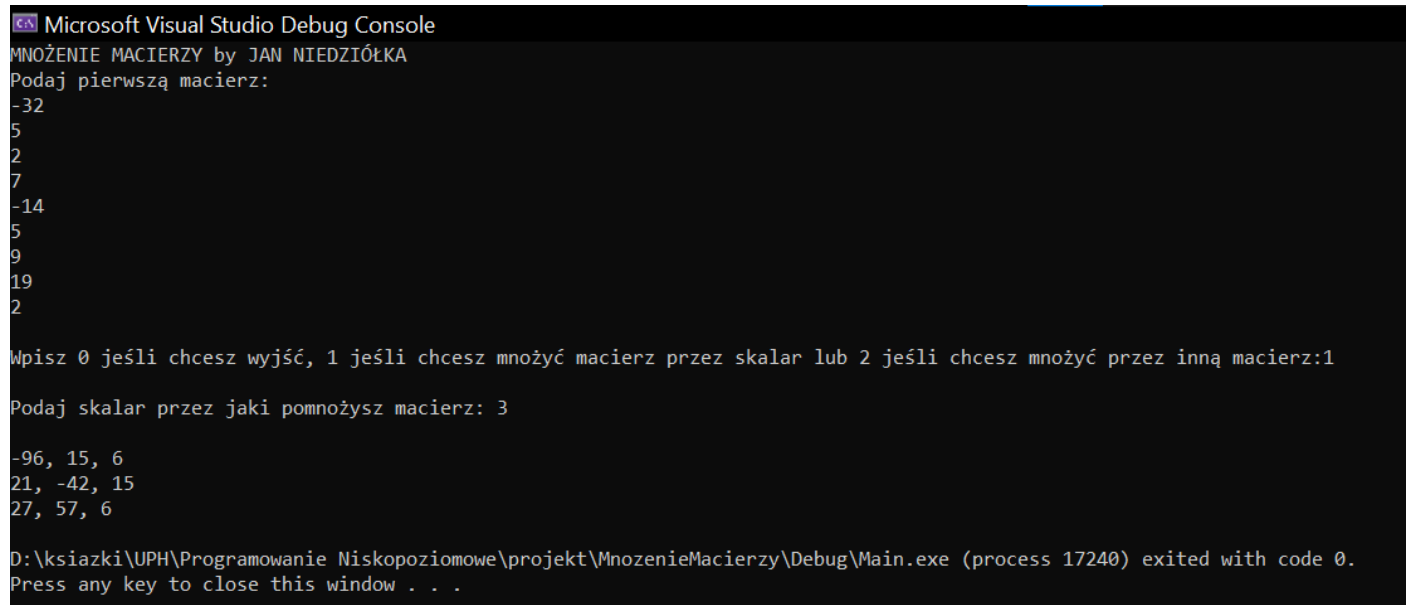
### Instrukcje dla użytkownika

Należy podążać za instrukcjami wyświetlającymi się na konsoli. Jedynie liczby całkowite zostaną uwzględnione przez program – zmiennoprzecinkowe (z



przecinkiem lub kropką) zostaną potraktowane jak całkowite, np. 0.62 – 62, 1.72 – 172; litery i inne znaki zostaną wpisane do macierzy jako 0. Plik tekstowy z wynikiem znajduje się w tym samym folderze co plik .asm.

### Przykładowe dane i wyniki



```
Microsoft Visual Studio Debug Console
MNOŻENIE MACIERZY by JAN NIEDZIÓŁKA
Podaj pierwszą macierz:
-32
5
2
7
-14
5
9
19
2

Wpisz 0 jeśli chcesz wyjść, 1 jeśli chcesz mnożyć macierz przez skalar lub 2 jeśli chcesz mnożyć przez inną macierz:1
Podaj skalar przez jaki pomnożysz macierz: 3

-96, 15, 6
21, -42, 15
27, 57, 6

D:\ksiazki\UPH\Programowanie Niskopoziomowe\projekt\MnozenieMacierzy\Debug\Main.exe (process 17240) exited with code 0.
Press any key to close this window . . .
```

Rys.10.

*Przykładowe mnożenie macierzy przez skalar.*



```
Microsoft Visual Studio Debug Console
MNOŻENIE MACIERZY by JAN NIEDZIÓŁKA
Podaj pierwszą macierz:
-1
2
3
4
5
6
7
8
9

Wpisz 0 jeśli chcesz wyjść, 1 jeśli chcesz mnożyć macierz przez skalar lub 2 jeśli chcesz mnożyć przez inną macierz:2
Podaj macierz przez jaką pomnożysz macierz: 8
-9
7
6
5
4
3
2
1

13, 25, 4
80, 1, 54
131, -5, 90

D:\ksiazki\UPH\Programowanie Niskopoziomowe\projekt\MnozenieMacierzy\Debug\Main.exe (process 1716) exited with code 0.
Press any key to close this window . . .
```

Rys.11.

*Mnożenie przez siebie dwóch macierzy.*