Uniwersytet Przyrodniczo-Humanistyczny w Siedlcach

Informatyka, II rok, 3-ci semestr

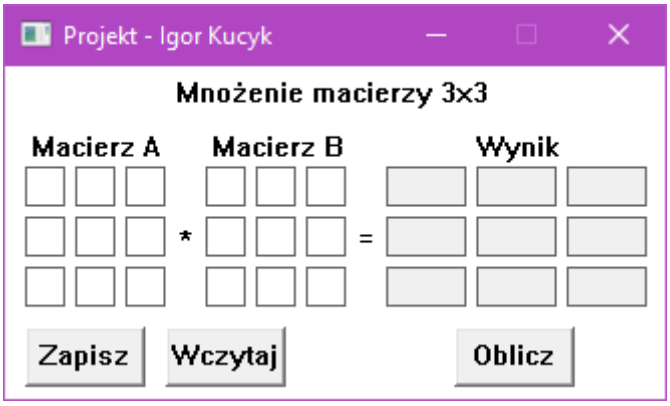
2020/2021

Projekt z programowania niskopoziomowego

Temat: Program do mnożenia macierzy 3x3 napisany w języku asembler. Argumenty pobrane od użytkownika jak i wynik obliczenia umieszczone w interfejsie graficznym. Możliwość zapisu oraz odczytu danych z programu.

Igor Kucyk

Opis programu



Program mnoży dwie macierze 3x3 i wyświetla wynik obliczenia – wszystko dzieje się w interfejsie graficznym. Aplikacja posiada:

- po 9 pól edycyjnych dla obu macierzy (razem 18 pól)
- 9 pól tekstowych dla macierzy wynikowej
- przycisk „Zapisz”, który zapisuje wszystkie trzy macierze do pliku binarnego lub informuje użytkownika o braku wprowadzonych danych (nie wykonano obliczeń ani razu)
- przycisk „Wczytaj”, który wczytuje dane dla wszystkich macierzy z pliku binarnego lub informuje użytkownika o braku pliku
- przycisk „Oblicz”, który po weryfikacji poprawności wypełnienia przez użytkownika macierzy (tzn. czy nie podano innych znaków niż cyfry i minus) mnoży obie macierze i wynik wypisuje w macierzy wynikowej

Opis rozwiązania

Uruchomienie programu:

```
;--- Pola edycyjne macierzy A i B oraz pola tekstowe macierza C (wynikowego) ---
createElements hedtAbuf, tedt, 10, 25, 20
createElements hedtBbuf, tedt, 100, 25, 20
createElements htxtCbuf, ttxt, 190, 45, 40
;--- Przyciski ZAPISZ, WCZYTAJ i OBLICZ ---
createButton hzap, zapisz,  10
createButton hwcz, wczytaj, 80
createButton hobl, oblicz,  224
```

Gdy program zostaje włączony, następuje inicjalizacja okna programu, elementów interfejsu graficznego oraz uchwytów. Jako, iż tworzenie elementów interfejsu wygląda podobnie, napisałem dwa makra *createElements* (jako argumenty przyjmuje tablice uchwytów do pól edycyjnych, typ elementu, początkowe X, długość odstępu między elementami i długość elementów) oraz *createButton* (jako argumenty przyjmuje uchwyt do przycisku, typ elementu i początkowe X).

Sprawdzenie poprawności wypełnienia pól edycyjnych:

```
;--- Sprawdzenie, czy wszystkie pola są poprawnie wypełnione ---
push OFFSET hedtAbuf
push OFFSET macierzA
call checkElement
cmp EAX, 0
je wndend
push OFFSET hedtBbuf
push OFFSET macierzB
call checkElement
cmp EAX, 0
je wndend
```

Program potrafi sprawdzić, czy podane przez użytkownika dane są poprawnie wypełnione i informuje go o ewentualnych błędach (np. gdy wprowadzono litery lub występują puste pola). Do sprawdzania napisałem procedurę *checkElement*, która przyjmuje dwa argumenty przez stos (adres do tablicy uchwytów pól edycyjnych oraz adres do tablicy wartości macierzy) i przy poprawnie wypełnionych polach uzupełnia tablicę wartości macierzy. Jeżeli wystąpi błąd, użytkownik otrzymuje komunikat, a następnie rejestr EAX jest ustawiany na 0, co wykorzystane jest do pominięcia części kodu dla poprawnie wypełnionych pól.

Mnożenie macierzy:

```
;--- Mnożenie macierzy ---
mov ESI, 0 ;licznik macierza A (wiersze)
mov EDI, 0 ;licznik macierza B (kolumny)
push EBP
mov EBP, 0 ;licznik do wynikowego macierza
mov wiersz, ESI
mov kolumna, EDI
mov ECX, 3
A:
    push ECX
    mov EDX, 0 ;suma
    mov ECX, 9
    B:
        push ECX
        call Multiply
        .IF ESI == 12 || ESI == 24 || ESI == 36
            call AddResult
        .ENDIF
        pop ECX
        loop B
    .IF kolumna == 12
        mov kolumna, 0
        mov EDI, kolumna
        add wiersz, 12
        mov ESI, wiersz
    .ENDIF
    pop ECX
    loop A
pop EBP
showResults
```

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 7 & 8 & -9 \end{vmatrix} * \begin{vmatrix} -9 & 8 & 7 \\ 6 & 5 & -4 \\ 3 & 2 & -1 \end{vmatrix} = \begin{vmatrix} a & b & c \\ & & \\ & & \end{vmatrix}$$

$$a = 1*(-9) + 2*6 + 3*3$$
$$b = 1*8 + 2*5 + 3*2$$
$$c = 1*7 + 2*(-4) + 3*(-1)$$

Mnożenie macierzy wykonuje się poprzez mnożenie elementów wiersza macierzy A (zielony kolor) z elementami kolumn macierzy B (czerwony, niebieski i fioletowy kolor). Po przejściu przez wszystkie kolumny następuje zwiększenie licznika wierszy i wyzerowanie licznika kolumn. Wszystkie wyniki obliczeń zapisywane są do tablicy wartości macierzy wynikowej.

Zapisywanie:

```
mov bufor, 0
mov rbuf, 128
invoke GetCurrentDirectoryA, rbuf, OFFSET bufor
invoke lstrcatA, OFFSET bufor, OFFSET fileName
invoke CreateFileA, OFFSET bufor, GENERIC_WRITE, 0, 0, CREATE_ALWAYS, 0, 0
mov hfile, EAX

saveMatrix macierzA
saveMatrix macierzB
saveMatrix macierzC
invoke CloseHandle, hfile
mov amount, 0
```

Program zapisuje wartości macierzy do pliku binarnego *wynik.dat* pod warunkiem, że macierze zostały poprawnie wypełnione i wykonano obliczenie.

Wczytywanie:

```
mov bufor, 0
mov rbuf, 128
invoke GetCurrentDirectoryA, rbuf, OFFSET bufor
invoke lstrcatA, OFFSET bufor, OFFSET fileName
invoke CreateFileA, OFFSET bufor, GENERIC_READ, 0, 0, OPEN_EXISTING, 0, 0
mov hfile, EAX
invoke GetLastError
.IF EAX == 2
    invoke MessageBoxA, 0, OFFSET bladMsg2, OFFSET blad, 0
    invoke CloseHandle, hfile
    jmp wndend
.ENDIF

readMatrix hedtAbuf, macierzA
readMatrix hedtBbuf, macierzB
readMatrix htxtCbuf, macierzC
invoke CloseHandle, hfile
mov amount, 0
```

Program wczytuje wartości do macierzy z pliku binarnego *wynik.dat* i aktualizuje pola edycyjne i tekstowe tak, aby wyświetlały wczytane dane. Jeżeli plik nie istnieje, to użytkownik zostanie o tym poinformowany.

Ograniczenia

Jedynym ograniczeniem programu jest ilość znaków możliwych do wprowadzenia w polach edycyjnych macierzy. Program wykonałem w dość minimalistyczny sposób, dlatego pola te są małe i obsługują maksymalnie 2 znaki. Oznacza to, że zakres liczb, które aplikacja obsługuje, wynosi $\langle -9; 99 \rangle$.