Podstawy programowania

Projekt nr 2

Wojciech Zieliński

Czwartek 10.15

Sala 09

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Prowadzący laboratorium: dr inż. Piotr Firek

Warszawa 2016

1.Treść projektu

Moim zadaniem było napisanie programu, który jest kalkulatorem macierzy kwadratowych obsługującym operacje logiczne takie jak: AND, OR, NAND, NOR, NOT i XOR. Rozmiar macierzy jest podawany przez użytkownika i wynosi maksymalnie 5. Macierze mogą być wypełniane tylko wartościami 0 i 1. Po każdym wykonanym obliczeniu użytkownik ma możliwość kontynuowania obliczeń używając otrzymany wynik do kolejnych obliczeń, podania całkowicie nowych wartości do obliczeń lub zakończenie działania programu. W przypadku podania błędnych wartości użytkownik wczytuje je do momentu aż będą poprawne.

2. Propozycja rozwiązania

Na początku użytkownik otrzymuje krótką instrukcję obsługi programu. Następnie zostaje poproszony o wybór operacji jaką ma w danym momencie wykonywać kalkulator. Jego możliwości to liczby całkowite od 1 do 6. Użytkownik może również zakończyć w tym momencie działanie programu wczytując wartość 7. W przypadku podania błędnej wartości użytkownik zostaje poproszony o podanie poprawnej wartości.

W tym momencie w programie rozpoczynam pętlę *while*, która działa do momentu aż użytkownik nie wczyta wartości 7 przy wyborze operacji, którą ma wykonać kalkulator.

Pojawia się instrukcja warunkowa *if*. Odpowiada ona za to, czy w naszych obliczeniach będziemy korzystali z otrzymanego przed chwilą wyniku, czy chcemy wczytać całkowicie nowe wartości. W przypadku pierwszego obrotu pętli *while* wczytujemy całkowicie nowe wartości do macierzy.

Następnie użytkownik podaje rozmiar macierzy, którego chce używać w swoich obliczeniach. Może podać liczbę całkowitą z przedziału od 1 do 5. W przypadku podania błędnej wartości użytkownik zostaje poproszony o podanie poprawnej wartości. Jeżeli użytkownik wybrał liczbę odpowiadającą operacji NOT, to wtedy musi wczytać tylko jedną macierz (chyba, że używa

wyniku z poprzednich obliczeń, to wtedy nie wczytuje nowej macierzy). W innym przypadku musi wczytać dwie macierze (chyba, że korzysta z wyniku otrzymanego w poprzednim obliczeniu, to wtedy musi wczytać tylko jedną macierz). Wartości danych pól macierzy są zapisywane są w tablicach dwuwymiarowych.

W tym momencie program odczytuje za pomocą instrukcji *switch*, która operację ma wykonać. W przypadku podania odpowiedniej wartości program odwołuje się do odpowiedniej funkcji, która wykonuje obliczenia i wypisuje na ekran otrzymany wynik.

Następnie mamy możliwość wyboru. Jeżeli chcemy kontynuować obliczenia wykorzystując otrzymany wynik musimy wczytać 0. Jeżeli chcemy kontynuować obliczenia wykorzystując całkowicie nowe wartości musimy wczytać 1. Jeżeli chcemy zakończyć działanie programu musimy wczytać 2. Jeżeli wczytamy 0 lub 1 to zostajemy poproszeni o wybór operacji (w tym miejscu również mamy możliwość zakończenia działania programu) i wracamy do początku pętli *while*. Jeżeli wczytaliśmy wartość 2, to, dzięki użyciu instrukcji break, wychodzimy z pętli i kończymy działanie programu.

3. Zmienne , stałe

W moim programie użyłem tylko jednej stałej MAKS_TAB równej 5, która odpowiada za to, aby każda tablica, w której zapisuję wartości macierzy mogła mieć maksymalnie wymiary 5 x 5.

W funkcji main użyłem następujących zmiennych (zmienne, które użyłem w funkcjach opiszę później):

- *rozmiar* – (typ int) zapisywany jest w niej aktualny rozmiar macierzy podany przez użytkownika
- *operacja* – (typ int) odpowiada za wybór operacji, którą ma wykonać kalkulator
- *nmdo* – (typ int) odpowiada za wybór tego, co chcemy robić po zakończeniu obliczeń. Na początku ma wartość 1, ponieważ przy pierwszych obliczeniach

nie mamy możliwości jej zmiany i wtedy wczytujemy całkowicie nowe wartości do macierzy i nie mamy możliwości korzystania z poprzedniego wyniku.

- *tab1[MAKS_TAB][MAKS_TAB]* – zmienna tablicowa w której są zapisywane wartości całkowite. Odpowiada za obsługę pierwszej macierzy do obliczeń.
- *tab2[MAKS_TAB][MAKS_TAB]* – zmienna tablicowa w której są zapisywane wartości całkowite. Odpowiada za obsługę drugiej macierzy do obliczeń.
- *tab3[MAKS_TAB][MAKS_TAB]* – zmienna tablicowa w której są zapisywane wartości całkowite. Odpowiada za obsługę wyniku otrzymanego podczas obliczeń.

4. Funkcje

W moim programie użyłem 9 funkcji. Każda z nich nie zwraca wartości, czyli jest typu void. W każdej funkcji (z wyjątkiem funkcji *wypełnij)* deklaruję zmienne całkowite *i* oraz *j*, które odpowiadają odpowiednio za numer wiersza i kolumny w tablicach, które są używane w tych funkcjach. W każdej funkcji, w której wykorzystuję te zmienne, zmienna *i* odpowiada również za indeks zewnętrznej pętli for, a zmienna *j* za indeks wewnętrznej pętli for.

1. not(int a, int wez[][MAKS_TAB],int oblicz[][MAKS_TAB])

Zmienna *a* odpowiada za rozmiar tablic i tyle razy wykonują się pętle *for* użyte w tej funkcji. Ta funkcja odpowiada za obsługę operacji NOT. Użyłem zagnieżdżonych pętli *for* do obliczania wartości w poszczególnych polach tablicy *oblicz* z wykorzystaniem tablicy *wez*. Wartość każdego pola tablicy *wez* jest negowana i przypisywana do odpowiedniego pola tablicy oblicz.

2. nor(int a, int wez1[][MAKS_TAB], int wez2[][MAKS_TAB],int oblicz[][MAKS_TAB])

Zmienna a odpowiada za rozmiar tablic i tyle razy wykonują się pętle *for* użyte w tej funkcji. Ta funkcja odpowiada za obsługę operacji NOR. Użyłem zagnieżdżonych pętli *for* do obliczania wartości
w poszczególnych polach tablicy *oblicz* z wykorzystaniem tablic *wez1* i *wez2*. Wartość operacji logicznej OR pomiędzy każdym polem tablicy *wez1* i *wez2* jest przypisywana do odpowiedniego pola tablicy *oblicz*.

3. nand(int a, int wez1[][MAKS_TAB], int wez2[][MAKS_TAB],int oblicz[][MAKS_TAB])

Zmienna *a* odpowiada za rozmiar tablic i tyle razy wykonują się pętle for użyte w tej funkcji. Ta funkcja odpowiada za obsługę operacji NOR. Użyłem zagnieżdżonych pętli for do obliczania wartości
w poszczególnych polach tablicy *oblicz* z wykorzystaniem tablic *wez1* i *wez2*. Wartość operacji logicznej NAND pomiędzy każdym polem tablicy *wez1* i *wez2* jest przypisywana do odpowiedniego pola tablicy *oblicz*.

4. and(int a, int wez1[][MAKS_TAB], int wez2[][MAKS_TAB],int oblicz[][MAKS_TAB])

Zmienna *a* odpowiada za rozmiar tablic i tyle razy wykonują się pętle for użyte w tej funkcji. Ta funkcja odpowiada za obsługę operacji AND. Użyłem zagnieżdżonych pętli for do obliczenia wartości
w poszczególnych pól tablicy *oblicz* z wykorzystaniem tablic *wez1* i *wez2*. Wartość operacji logicznej AND pomiędzy każdym polem tablicy *wez1* i *wez2* jest przypisywana do odpowiedniego pola tablicy *oblicz*.

5. or(int a, int wez1[][MAKS_TAB], int wez2[][MAKS_TAB],int oblicz[][MAKS_TAB])

Zmienna *a* odpowiada za rozmiar tablic i tyle razy wykonują się pętle for użyte w tej funkcji. Ta funkcja odpowiada za obsługę operacji OR. Użyłem zagnieżdżonych pętli for do obliczania wartości

w poszczególnych pól tablicy *oblicz* z wykorzystaniem tablic *wez1* i *wez2*. Wartość operacji logicznej OR pomiędzy każdym polem tablicy *wez1* i *wez2* jest przypisywana do odpowiedniego pola tablicy *oblicz*.

6. xor(int a, int wez1[][MAKS_TAB], int wez2[][MAKS_TAB],int oblicz[][MAKS_TAB])

Zmienna *a* odpowiada za rozmiar tablic i tyle razy wykonują się pętle for użyte w tej funkcji. Ta funkcja odpowiada za obsługę operacji XOR. Użyłem zagnieżdżonych pętli for do obliczenia wartości w poszczególnych polach tablicy *oblicz* z wykorzystaniem tablic *wez1* i *wez2*. Wartość operacji logicznej XOR pomiędzy każdym polem tablicy *wez1* i *wez2* jest przypisywana do odpowiedniego pola tablicy *oblicz*.

7. wczytaj(int *a, int p, int k)

Ta funkcja odpowiada za wczytywanie liczb połączone z obsługą błednie wczytywanych wartości. Do zmiennej wskaźnikowej a wczytujemy naszą wartość. Jeżeli *a nie jest scanfem, to wtedy pętla while użyta w tej funkcji zaczyna działać i użytkownik zostaje poproszony o wczytanie poprawnej wartości. Przed każdym wczytaniem nowej wartości czyścimy bufor. Wczytujemy kolejną wartość aż do skutku. W przypadku jeżeli wczytamy np. „1.” to też będzie to uznane za błąd ponieważ w pętli while jest instrukcja if, która odpowiada za to, że jeżeli wczytamy liczbę całkowitą a po niej jakiś inny zank drukowany, to też będzie to wyświetlonej jako błąd gdyż w buforze będzie zapisany ten znak, a pętla if działa jeżeli znak odczytany z bufora nie będzie znakiem niedrukowanym. Zmienne p oraz k są użyte po to aby określić możliwy przedział wczytywanych wartości całkowitych.

8. wypelnij_macierz(int a, int wyp[MAKS_TAB][MAKS_TAB]);

Ta funkcja odpowiada za wypelnianie pol odpowiednich macierzy. Zmienna a odpowiada za rozmiar tablic i tyle razy będą działały pętle zagnieżdżone dzięki

którym wypełniamy kolejne pola tablicy *wyp*. W tej funkcji używam funkcji *wczytaj*, dzięki której mam zapewnioną obsługę błednych wartości przy wypełnianiu kolejnych pól macierzy.

9. zamien_macierz(int a, int wez[MAKS_TAB][MAKS_TAB], int zamien[MAKS_TAB][MAKS_TAB])

Ta funkcja odpowiada za wypełnienie tablicy *zamien* wartościami z tablicy *weź.* Zmienna a odpowiada za rozmiar tablic i tyle razy będą działały pętle zagnieżdżone dzięki którym możemy do danego pola tablicy *zamien* przypisać odpowiednie pole z tablicy *weź.*

5. Testowanie

Sprawdzałem czy program poprawnie wykonuje obsługę błędów poprzez wczytywanie liczb, które wykraczają poza proponowane wartości przy instrukcjach wybierających oraz wypełnianiu macierzy. Wczytywanie liter bądź innych znaków drukowanych, które nie są cyframi. Sprawdzałem czy podany rozmiar macierzy prawidłowo określa ile mamy wypełnić pól macierzy oraz czy wybrany przez użytkownika rodzaj operacji sprawia, że program poprawnie oblicza

6. Odstępstwa

Jeżeli wczytamy poprawną wartość następnie spację lub tabulator i potem jakiś dowolny ciąg znaków, to program nie odczyta tego jako błąd.

7. Algorytm w postaci schematu blokowego