Politechnika Śląska w Gliwicach
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

TUC

autor	Sławomir Krzykała
prowadzący	dr inż. Tomasz Moroń
rok akademicki	2018/2019
kierunek	teleinformatyka
rodzaj studiów	SSI
semestr	1
termin laboratorium / ćwiczeń	piątek, 08:15 – 9:45
grupa	3
sekcja	7
termin oddania sprawozdania	2019-01-02
data oddania sprawozdania	2019-01-25

1 Treść zadania

Napisać program analizujący układ złożony z bramek logicznych. Dostępne są następujące bramki: and, nand, or, nor, xor, xnor, neg. Każda bramka ma jedno wyjście i dwa wejścia. Jedynym wyjątkiem jest bramka neg , która ma jedno wejście i jedno wyjście. Połączenie wejść i wyjść bramek jest traktowane jako węzeł.

Plik wejściowy przedstawiający układ na następujący format: W pierwszej linii podane są numery wę- złów będących wejściem układu. W drugiej linii numery węzłów będące wyjściem układu. Każda następna linia zawiera opis jednej bramki w postaci:

```
<węzeł wejściowy> <węzeł wejściowy> <wezeł wyjściowy>
```

Drugi plik wejściowy zawiera w każdej linii stany wejść, dla których należy znaleźć stan wyjść.

Trzeci plik - wynikowy zawiera wartości wyjść dla zadanych stanów wejść.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

`-u` plik wejściowy z układem
`-i` plik wejściowy ze stanami wejść
`-o` plik wejściowy ze stanami wyjść

albo

`-h` wyświetlona zostane pomoc

2 Analiza zadania

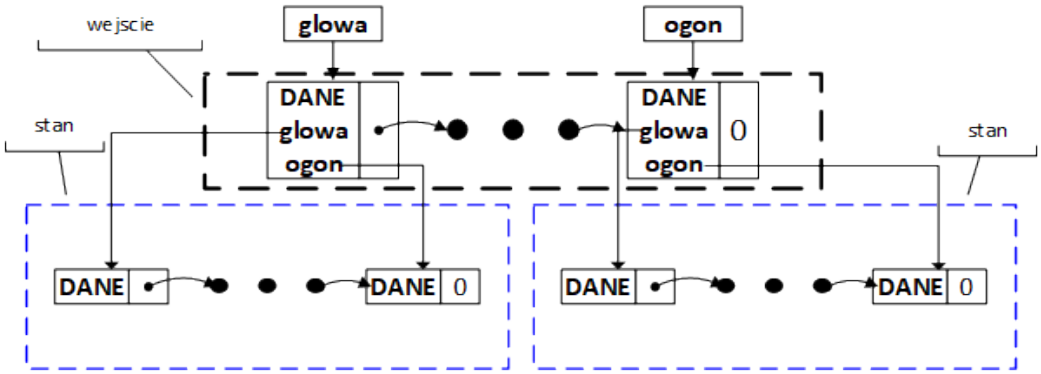
Zagadnienie przedstawia problem rozwiązania układów logicznych odpowiednio zapisanych w pliku wejściowym, na podstawie podanych stanów wejsciowych zapisanych w drugim pliku wejściowym oraz zapisania wyników do pliku wyjściowego - stanów odpowiednich wyjść.

2.1 Struktury danych

W programie wykorzystano cztery struktury dynamiczne typu lista jednokierunkowa, do przechowywania odpowiednio wartości - informacji o wejściach i ich stanach, informacji o bramkach logicznych w układzie i wyjść układu.

Taka struktura danych umożliwia łatwe wczytanie danych z pliku oraz przetworzenie ich.

Rysunek 1: Schemat użytych struktur typu lista jednokierunkowa.



Rysunek 2: Schemat połączenia listy wejść z listami stanów każdego z wejść w kolejnych przypadkach.

2.2 Algorytmy

Program po wczytaniu danych do struktur, rekurencyjnie wyznacza stan podanych wyjść - zaczynając od wezła wyjściowego dążąc do wejść/a.

Taka struktura danych umożliwia łatwe wczytanie danych z plików oraz przetworzenie ich.

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy dwóch plików wejsciowych z: układem, stanami wejć oraz wyjściowego po odpowiednich przełącznikach (odpowiednio: `-i` dla pliku wejściowego z stanami wejć, `-u` dla pliku wejściowego z układem i `-o` dla pliku wyjściowego z wynikiem), np.

```
program  -o uklad.txt -i stany_wejsc.txt -o wynik.txt
program  -o uklad.txt -o wynik -i stany_wejsc
```

Pliki są plikami tekstowymi, ale mogą mieć dowolne rozszerzenie (lub go nie mieć). Przełączniki mogą być podane w dowolnej kolejności. Uruchomienie programu bez żadnego parametru lub z parametrem `-h`

```
program
program -h
```

powoduje wyświetlenie krótkiej pomocy. Uruchomienie programu z nieprawidłowymi parametrami powoduje wyświetlenie komunikatu

```
Nieprawidlowe podane parametry uruchomienia!
```

i wyświetlenie pomocy.

Podanie nieprawidłowej nazwy pliku lub gdy otwieranie pliku się nie powiedzie (np. jest zabezpieczony) powoduje wyświetlenie odpowiedniego komunikatu:

```
Blad otwierania pliku: [ <nazwa_pliku> ]!
```

4 Specyfikacja wewnętrzna

4.1 Typy zdefinowane w programie

W programie zdefiniowano następujące typy:

```
1 struct stan
2 {
3 unsigned int linia;
4 bool wartosc;
5 stan *nast = nullptr;
6 };
```

Służy do zbudowania listy stanów dla kolejnych przypadków.

```
1 struct wejscie
2 {
3 unsigned int nr;
4 stan* glowa_stan = nullptr;
5 stan* ogon_stan = nullptr;
6 wejscie *nast=nullptr;
7 };
```

Służy do zbudowania listy wejść, każde wejście ma własną listę zawierającą stany wejść podawanych w kolejnych przypadkach (liniach).

```
struct wyjscie
{
unsigned int nr;
wyjscie *nast= nullptr;
};
```

Służy do zbudowania listy wyjść.

```
struct bramka
{
short typ;
unsigned int  nr_wejscia1, nr_wejscia2, nr_wyjscia;
bramka *nast=nullptr;
};
```

Służy do zbudowania listy bramek w układzie.
któych głowy i ogony są zapisane w zmiennych globalnych o deklaracji:

```
wejscie * glowa_wejscie = nullptr;
wyjscie * glowa_wyjscie = nullptr;
bramka * glowa_bramka = nullptr;
wejscie * ogon_wejscie = nullptr;
wyjscie * ogon_wyjscie = nullptr;
bramka * ogon_bramka = nullptr;
```

4.2 Ogólna struktura programu

W funkcji głównej wywołana jest funkcja

```
bool przelaczniki(int argc, char* argv[], string &
    src_uklad, string &src_wejscia, string &src_wyjscia)
    ;
```

Funkcja ta sprawdza, czy program został wywołany w prawidłowy sposób (patrz 3). Funkcja wywołana jest w instrukcji warunkowej zwraca stan logiczny odpowiadający rezultatowi jej wykonania. Gdy program nie został wywołany prawidłowo funkcja zwraca FALSE, zostaje wypisany stosowny komunikat (informacja o błędzie oraz pomoc) i program się kończy.

Po sprawdzeniu parametrów w zmiennych src_wejscia , src_uklad i src_wyjscia przechowywane są nazwy pliku wejściowego, układu i wyjściowego. Następnie wywoływana jest funkcja

```
bool wczytaj_uklad(string src_uklad);
```

Funkcja ta otwiera plik src_uklad, sczytuje linijka po linijce i pobiera z tych linii wejścia, wyjścia oraz bramki logiczne za pomocą odpowiednich funkcji **bool** wczytaj_in(**string** ln);, **bool** wczytaj_out(**string** ln); i **bool** wczytaj_bramke (**string** ln);

Po sczytaniu wszystkich danych funkcja zamyka plik. W razie wystąpienia błędu funkcja wyświetla odpowiedni komunikat i zwraca FALSE, w przeciwnym wypadku zwraca TRUE.

Następnie wywoływana jest funkcja

```
1 bool wczytaj_stan_wejsc(string src_wejscia);
```

Funkcja ta otwiera plik src_wejscia, sczytuje linijka po linijce i pobiera z tych linii stany wejść, za pomocą funkcji **bool** wczytaj_stany(**string** ln, **int** nr_ln);.

Po sczytaniu wszystkich danych funkcja zamyka plik. W razie wystąpienia błędu funkcja wyświetla odpowiedni komunikat i zwraca FALSE, w przeciwnym wypadku zwraca TRUE.

Kolejną funkcją programu jest funkcja analizująca układ

```
1 void analizuj(string &wynikowy);
```

która zapisuje w zmiennej typu string o nazwie wynikowy, stany wyjść dla danych stanów wejść wejścia są szukane po przez funkcję wejscie* szukaj_wejscia (**int** nr_wejscia);; ,stany wejściowe dla danego przypadku są szukane po przez funkcję stan* szukaj_stan(stan* glowa_stan, **int** nr_ln);, stany wynikowe są obliczane za pomocą funkcji **bool** wynik_wezla(**int** nr_wezla, **int** nr_ln);. W przypadku jakiegoś niepowodzenia wyświetlany jest stosowny komunikat i program kończy się.

Przedostatnią funkcją jest

```
1 bool zapisz_wynik(string src_wyjscia, string wynik);
```

która zapisuje zmienną z wynikiem do pliku src_wyjscia. W razie wystąpienia błędu funkcja wyświetla odpowiedni komunikat i zwraca FALSE, w przeciwnym wypadku zwraca TRUE.

ostatnią funkcją jest

```
1 void usun_struktury();
```

która usuwa wszystkie utworzone struktury.

Po niej wyświetlony zostaje komunikat o pomyślnym przebiegu programu i program kończy się.

4.3 Szczegółowy opis implementacji funkcji

Opis pozostałych użytych funkcji:

```
1 void walidacja(string &ln);
```

Poprawia błędy zapisu w linicje, eliminuje nadmiarowe białe znaki, zamienia cały tekst na duże litery, zamienia ';' na ':' oraz eliminuje nadmiarowe ':'.

```
1 int pobierz_liczbe(string &ln);
```

Pobiera z linijki kolejne liczby (nieujemne) i je zwraca, jako separator dwóch liczb uważa się każdy znak inny niż cyfra. Pobrane znaki usuwa ze zmiennej. Przy braku cyfr zwaraca '-1'.

```
1 bool wczytaj_in(string ln);
```

Po przetworzeniu linijki przez fukcje **void** walidacja(**string** &ln);, pobiera numery węzłów z wejściami po przez funkcję **int** pobierz_liczbe (**string** &ln); i zapisuje do listy z numerami wejść do układu.

```
1 bool wczytaj_out(string ln)
```

Po przetworzeniu linijki przez fukcje **void** walidacja(**string** &ln);, pobiera numery węzłów z wyjściami po przez funkcję **int** pobierz_liczbe (**string** &ln); i zapisuje do listy z numerami wyjść z układu.

```
1 int pobierz_bramke(string &ln);
```

Pobiera typ bramki z linii i zamienia na odpowiadającą jej liczbe według globalnej statycznej tablicy **const string** rodzaj_bramki [] = { "NAND", "AND ", "NOR", "XOR", "XNOR", "OR", "NEG"};

Bramce NAND odpowiada 0, AND to 1 etc. Pobrany typ bramki usuwa ze zmiennej. Przy braku rozpoznania typu zwaraca '-1'.

```
1 bool wczytaj_bramke(string ln);
```

Po przetworzeniu linijki przez fukcje **void** walidacja(**string** &ln);,pobiera typ bramki przez funkcję **int** pobierz_bramke(**string** &ln);, następnie pobiera numery wejść i wyjść po przez funkcję **int** pobierz_liczbe (**string** &ln); i zapisuje do listy z bramek układu. W przypadku bramki NEG numer wejścia pierwszego i drugiego jest ustawiany jako taki sam.

```
1 wejscie* szukaj_wejscia(int nr_wejscia);
```

Szuka wejścia o podanym numerze i zwraca jego adres. W przypadku gdy go nie znajdzie zwraca nullptr.

```
1 void dodaj_stan(wejscie* w_wejscia, bool stan_ln, int
    nr_ln);
```

Zapisuje stan wraz z jego numerem lini(numer przypadku do analizy) danego wejścia do listy.

```
1 bool wczytaj_stany(string ln, int nr_ln)
```

Po przetworzeniu linijki przez fukcje **void** walidacja(**string** &ln);,pobiera numer wejśćа i jego stan po przez funkcję **int** pobierz_liczbe(**string** &ln);, szuka wejścia o podanym numerze za pomocą funkcji wejscie* szukaj_wejscia(**int** nr_wejscia); i dodaje stan do wyszukanego wejścia za pomocą funkcji **void** dodaj_stan(wejscie* w_wejscia, **bool** stan_ln, **int** nr_ln);. Powtarza te kroki (bez walidacji), aż do wczytania wszystkich stanów wejść z danej linii.

```
1 stan* szukaj_stan(stan* glowa_stan, int nr_ln);
```

Szuka stanu wejścia w podanym numerze lini(przypdku) i zwraca jego adres. W przypadku gdy go nie znajdzie zwraca nullptr.

```
1 bool wyjscie_bramka(int rodzaj, bool w1, bool w2);
```

Zwraca stan wyjścia danej bramki na podstawie otrzymanech parametrów: rodzaju bramki, stanu wejścia pierwszego i stanu wejścia drugiego. Jeżeli rodzaj bramki jest niepoprawny wyświetla odpowiedni komunikat i kończy program.

```
1 bool wynik_wezla(int nr_wezla, int nr_ln;
```

Wyznacza stan podanego węzła, na podstawie stanów wejściowych z podanego numeru linii. Robi to rekurencyjnie, dążąc od podanego węzła do węzłów wejściowych.

5 Testowanie

Program został przetestowany na różnego rodzaju plikach. Pliki niepoprawne (niezawierające bramek/wejść/wyjść/stanów, puste, niezgodne ze specyfikacją) powodują zgłoszenie błędu. Również gdy użytkownik poda niepoprawny logicznie układ program wyświetli stosowny komunikat i zkończy się. Pliki w których dane są zniekształcone po przez zmianę wielkości znaków, białe znaki, losowe znaki nie powodujące zmiany sensu pliku są obsługiwane poprawnie. Maksymalna numer wejścia akceptowana zależy od kompilatora (typ integer może być realizowany jako zmienna dwu- lub czterobajtowa). Maksymalna liczba przypadków akceptowana zależy od kompilatora (typ **unsigned** integer może być realizowany jako zmienna dwu- lub czterobajtowa).

Program został sprawdzony pod kątem wycieków pamięci.

6 Wnioski

Program do analizowania bramek logicznych po przeanalizowaniu problemu nie jest programem skomplikowanym, chociaż wymaga samodzielnego zarządzania pamięcią. Najbardziej wymagające okazało się sposób obliczania stanów wyjściowych. Szczególnie wymagajace było zapewnienie wyświetlania jak najbardziej szczegółowych komunikatów o błędach, aby użytkownik wiedział co jest niepoprawne. Myślę, że w przyszłości dobrym pomysłem będzieo użycie obsługi wyjatków do takich celów.