Politechnika Śląska w Gliwicach

Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Bibliografia

autor	Daniel Ochman
prowadzący	mgr.inż. Marek Kokot
rok akademicki	2015/2016
kierunek	Teleinformatyka
rodzaj studiów	SSI
semestr	1
grupa	1

1. Treść zadania

Napisać program, który w tekście artykułu zastępować będzie etykiety książek \cite{etykieta}, w nim użytych, odpowiednimi numerami z bibliografii. W wyniku działania programu zostanie utworzony plik wynikowy, który zawierać będzie treść artykułu z elementami zamienionymi na nawiasy kwadratowe zawierające odpowiednie numery z bibliografii, która będzie dodana jako ostatni akapit w pliku. Te książki, które nie są cytowane, ale występują w pliku z opisami bibliograficznymi, nie są umieszczane w bibliografii. Książki w bibliografii są posortowane wg nazwisk autorów i są ponumerowane. Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

-t plik wejściowy z tekstem

-b plik z opisami bibliograficznymi

-o plik wyjściowy

2. Analiza zadania

Zagadnienie przedstawia problem implementacji i korzystania z dynamicznych struktur danych oraz zarządzania pamięcią w programie. Problem polega na sczytywaniu z plików informacji o danych książkach (umieszczaniu ich w dynamicznej strukturze danych), a także części artykułu oraz zastąpieniu etykiet odpowiednimi numerami z bibliografii.

2.1 Struktury danych

W programie wykorzystano dwa drzewa binarne. Pierwsze do przechowywania informacji o wszystkich książkach zawartych w pliku z opisami bibliograficznymi. Drugie do przechowywania informacji tylko o tych książkach, których etykiety występują w treści artykułu. Węzeł może mieć od 0 do 2 potomków, przy czym po lewej stronie węzła znajdują się potomkowie przechowujący etykiety nie większe alfabetycznie niż węzeł rodzicielski, po prawej zaś większe alfabetycznie. Taka struktura danych ułatwia szybkie sortowanie elementów.

2.2 Algorytmy

Program sczytuje z opisów bibliograficznych wszystkie dane dotyczące danych książek, a następnie rekurencyjnie tworzy drzewo binarne, którego kluczem są etykiety. Drzewo jest posortowane wstępnie według właśnie etykiet. Następnie po sczytaniu całego artykułu i po napotkaniu odpowiedniego znacznika \cite, etykieta wyszukiwana jest w drzewie i wtedy tworzone jest drugie drzewo, do którego te dane są przepisywane i sortowane według nazwisk autorów. W ten sposób otrzymujemy drzewo zawierające wszystkie elementy, które zostaną użyte w dalszej części wykonywania programu.

3. Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: wejściowego z treścią artykułu, wejściowego z opisami bibliograficznymi oraz wyjściowego po odpowiednich przełącznikach (odpowiednio -t , -b, -o), np.

```
program -t Publikacja.txt -b Opisy_bibliograficzne.txt -o Plik_wynikowy.txt
```

Pliki są plikami tekstowymi. Przełączniki mogą być podane w dowolnej kolejności. Uruchomienie programu bez żadnego parametru powoduje wyświetlenie komunikatu oraz krótkiej pomocy. Uruchomienie programu z nieprawidłowymi parametrami powoduje wyświetlenie komunikatu:

```
Podane zostały błędne parametry, zastosuj sie do pomocy wyświetlonej poniżej.
```

i wyświetlenie pomocy. Podanie nieprawidłowej nazwy pliku powoduje wyświetlenie odpowiedniego komunikatu:

```
Nie można otworzyć pliku: <nazwa_pliku>
```

4. Specyfikacja wewnętrzna

Program został zrealizowany w taki sposób, że nie ma komunikacji z użytkownikiem (dane są przechowywane w plikach wcześniej przygotowanych, program nie prosi o nic) i wyniki wypisywane są od razu do pliku wynikowego bez wyświetlania w konsoli.

4.1 Typy zdefiniowane w programie

W programie zdefiniowano następujące typy:

```
struct MainNode
{
    string Etykieta;
    string Imie;
    string Nazwisko;
    string Tytul;

    MainNode *left, *right;
};
```

```
struct SubNode
{
    string Etykieta;
    string Imie;
    string Nazwisko;
    string Tytul;
    int Numer;

    SubNode *left, *right;
};
```

Typy te służą do budowania drzew binarnych. Struktura MainNode to drzewo główne przechowujące informacje o wszystkich książkach z opisów bibliograficznych. Pola string są wartościami przechowywanymi w węźle natomiast **MainNode *left, *right** są to potomkowie węzła. Taką samą strukturę ma drugie drzewo jednakże ma ono dodatkowe pole przechowujące wartość **int** będącą kolejnym numerem węzła.

4.2 Ogólna struktura programu z szczegółowymi opisami funkcji

W funkcji main wywołane są funkcje:

```
- bool Parsuj_parametry(int argc, char**argv, string& Publikacja_file, string& Opisy_bibliograficzne_file, string& Wyjsciowy_file)
```

Funkcja ta sprawdza, czy program został wywołany w prawidłowy sposób. Wtedy funkcja zwraca wartość typo bool. Gdy program nie został wywołany prawidłowo zostaje wypisany stosowny komunikat i program się kończy. Po sprawdzeniu parametrów w zmiennych Publikacja_file, Opisy_bibliograficzne_file, Wyjsciowy_file przechowywane są nazwy plików wejściowych i wyjściowego.

```
- void CzytajOpisy(const string& Opisy_bibliograficzne_file, MainNode*& root)
```

Funkcja ta otwiera plik Opisy_bibliograficzne_file, sczytuje wszystkie elementy opisu książki do określonych zmiennych string i po wczytaniu jednej "partii" danych wpisuje je do drzewa binarnego korzystając z pomocniczej funkcji: `void DodajGlowneDrzewo(MainNode*&root, string Etykieta, string Imie, string Nazwisko, string Tytul)`, która w sposób rekurencyjny tworzy kolejne elementy głównego drzewa binarnego.

```
- void CzytajPublikacje(const string& Publikacja_file, SubNode*& subroot, MainNode*& root)
```

Funkcja ta otwiera plik Publikacja_file, sczytuje po kolei każdą linię tekstu, a następnie wyszukuje w pętli każdy element w linii zawierający frazę "\\cite". Gdy ją odnajdzie wydziela spomiędzy klamr nazwę etykiety w danym miejscu, a następnie wywołuje pomocniczą funkcję: `MainNode* SzukajWDrzewie(MainNode*& wsk, string tmp)`, która w drzewie głównym wyszukuje węzeł z odpowiednią etykietą "wyciągniętą z tekstu" i zwraca wskaźnik na ten węzeł. Następnie wywoływana jest funkcja `void DodajPoddrzewo(SubNode*&subroot, string Etykieta, string Imie, string Nazwisko, string Tytul, int Numer)` tworząca rekurencyjnie kolejne elementy drugiego drzewa binarnego, dane są kopiowane z pierwszego drzewa do drugiego przy pomocy wcześniej odnalezionego wskaźnika.

```
- void NadajNumery(SubNode*&subroot, int& licznik)
```

Funkcja ta przechodzi w porządku inorder przez drugie drzewo przechowujące tylko potrzebne informacje a następnie każdemu węzłowi nadaje kolejny numer począwszy od 1.

```
- void ZastapNumerami(const string& Publikacja_file, const string& Wyjsciowy_file,SubNode*& subroot, MainNode*& root)
```

W funkcji tej otwierane są dwa pliki Publikacja_file oraz Wyjsciowy_file. Powtarzana jest ta sama pętla co w przypadku pierwszego otwarcia pliku Publikacja_file, po wyodrębnieniu etykiety zawartej w tekście wywoływana jest funkcja `SubNode* SzukajWDrzewie(SubNode*& wsk, string tmp)` szukająca odpowiedniej etykiety w pierwszym drzewie. Zwraca ona wskaźnik na ten węzeł i zapisuje go w dodatkowym wskaźniku. Następnie do zmiennej

tymczasowej tmp wpisywane jest nazwisko z węzła i wywoływana jest funkcja `SubNode* SzukajWPoddrzewie(SubNode*& wsk, string tmp)` szukająca w drugim drzewie odpowiednich nazwisk (wartość zwracana przez funkcję zapisana jest do zmiennej wskaznik). Następnie przy pomocy polecenia **replace** elementy \cite{etykieta} zawarte w sczytywanych liniach tekstu zamieniane są na odpowiednie numery węzłów, które można odczytać przy pomocy ustawionego wcześniej na odpowiednim węźle wskaźnika. Po zastąpieniu wszystkich etykiet odpowiednimi numerami linia wpisywana jest przy pomocy strumienia do pliku Wyjsciowy_file.

- `void WypiszBibliografie(const string& Wyjsciowy_file, SubNode*& subroot)`

W funkcji tej otwierany jest ponownie plik Wyjsciowy_file, następnie wpisywane są do niego dodatkowe elementy jak spacje, linia oddzielająca część artykułu od bibliografii oraz sam nagłówek "Bibliografia". Wywoływana wtedy jest pomocnicza funkcja `void WypiszPoddrzewo(SubNode* subroot, const string & Wyjsciowy_file)` przechodząca przez drugie drzewo w porządku inorder i wpisująca informacje zawarte w każdym węźle na podstawie schematu: Numer-Imię-Nazwisko-Tytuł do pliku.

- `void UsunGlowneDrzewo(MainNode*&root)`
- `void UsunPoddrzewo(SubNode*&subroot)`

Funkcje te w sposób rekurencyjny przechodzą przez obydwa drzewa w porządku postorder i zwalniają pamięć po strukturach. Najpierw usuwane jest lewe poddrzewo, potem prawe, a na samym końcu korzeń. Wskaźniki natomiast ustawiane są na wartości nullptr.

5.Testowanie

Program, jeżeli chodzi o zawartość plików, został przetestowany na trzy sposoby. Gdy plik wejściowy, mający zawierać opisy bibliograficzne, jest pusty program zgłasza błąd i dalsze jego wykonywanie zostaje zawieszone. W przypadku, gdy drugi plik wejściowy jest pusty, tzn. nie ma tekstu publikacji to program kompiluje się o kończy działanie jednakże plik wynikowy jest wtedy pusty. W przypadku poprawnych danych w obydwu przypadkach program kompiluje się bez żadnych problemów i plik wynikowy zostaje utworzony w poprawnej formie. Program został także przetestowany pod kątem wycieków pamięci.

6. Wnioski

Program do tworzenia pliku ze zmienionym tekstem oraz bibliografią nie jest programem trudnym, aczkolwiek wymaga od piszącego dobrej znajomości wykorzystywanych struktur dynamicznych oraz samodzielnego zarządzania pamięcią. Najbardziej wymagające okazało się tworzenie drugiego drzewa w oparciu o znalezienie elementu w pierwszym drzewie oraz zastępowanie poszczególnych etykiet w tekście odpowiednimi numerami. Samo alokowanie pamięci czy też jej zwalnianie nie okazało się problemem, jednakże jej efektywne wykorzystanie, wczytywanie do plików w sposób, który

umożliwia wykonanie odpowiednich operacji było już zadaniem trudniejszym. Problemem w trakcie tworzenia programu okazało się także odpowiednie sczytywanie danych z plików w taki sposób by móc swobodnie zmieniać części całych linii na odpowiednie numery z bibliografii. Sczytywanie zwykłym strumieniem powodowało problemy, gdyż tekst publikacji nie mógł wtedy zawierać żadnych znaków interpunkcyjnych jak kropka czy przecinek. Przydatne okazały się wtedy polecenia getline oraz replace, przebudowanie pętli także było konieczne. Oprócz tego, odpowiednie poruszanie się po drzewach także okazało się wyzwaniem. Zwłaszcza w przypadku połączonych operacji na obydwóch strukturach. Rozwiązaniem problemu wykraczania poza obszar struktury przy szukaniu odpowiedniego węzła okazało się przypisywanie wskaźnika zawsze na korzeń (wcześniej zapamiętywało poprzednio wyszukaną pozycję i tych wcześniejszych nie było szans odnaleźć).