Politechnika Śląska w Gliwicach
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Temat projektu: Lister

Autor:	Sara Witek
Prowadzący	dr inż. Tomasz Moroń
Rok akademicki:	2018/2019
Rodzaj studiów:	Teleinformatyka
Semestr:	1
Termin laboratorium:	piątek 10.00 - 11.30
Grupa:	T3
Sekcja:	8
Data oddania sprawozdania:	16.01.2019

1 Treść zadania

Korzystając z drzewa binarnego i listy napisać program, który wczytuje plik tekstowy i wypisuje w pliku wynikowym wszystkie słowa w kolejności alfabetycznej wraz z numerami linii, w których dane słowo wystąpiło.
Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

-i plik wejściowy z tekstem.

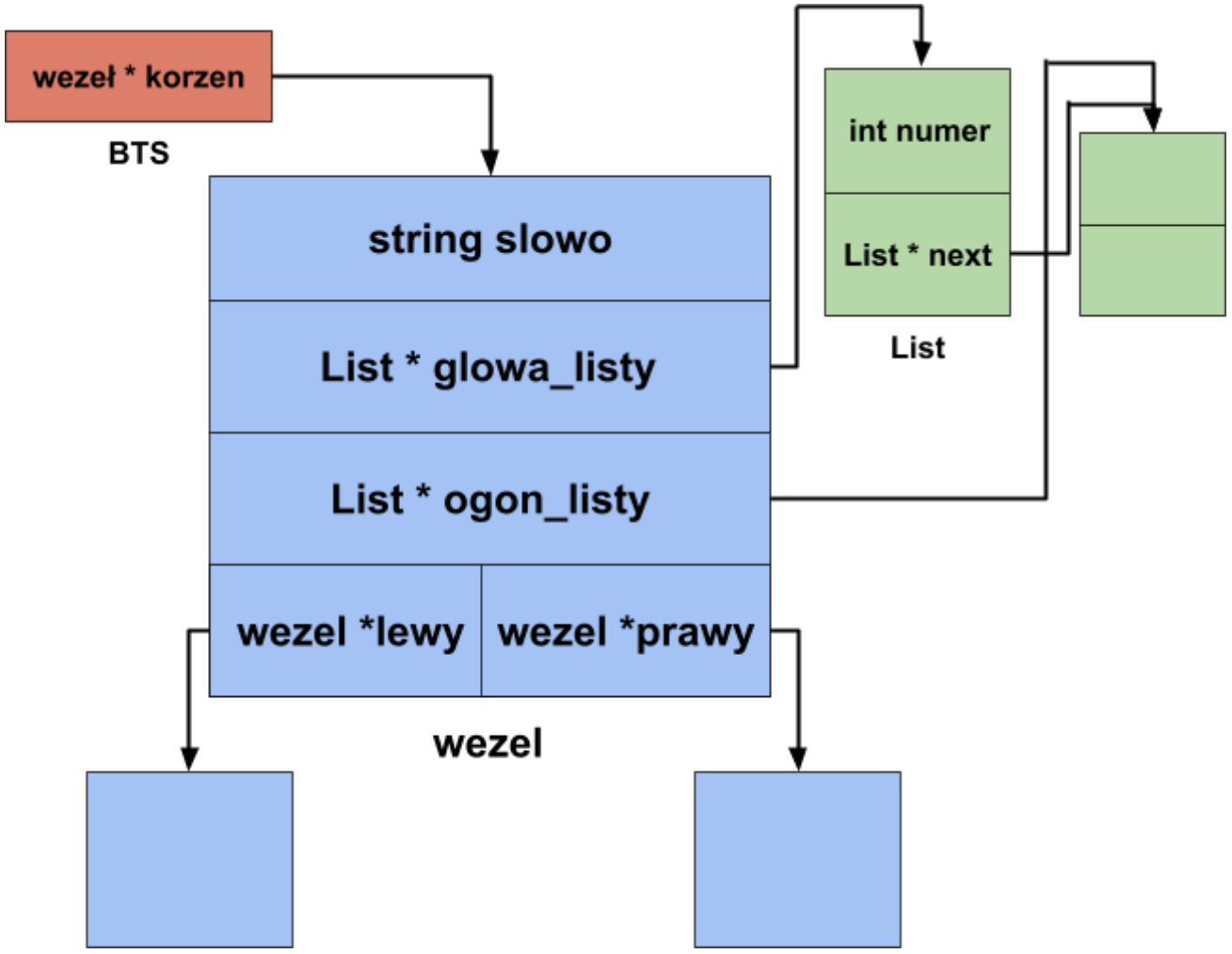
-o plik wyjściowy z listą słów.

2 Analiza zadania

Zagadnienie przedstawia problem sortowania alfabetycznie słów zapisanych w pliku wejściowym.

2.1 Struktury danych

W programie wykorzystano binarne drzewo poszukiwań w celu przechowywania wczytanych słów z pliku wejściowego. Drzewo binarne przechowuje dane w węzłach. Węzeł może mieć od 0 do 2 potomków, przy czym po lewej stronie węzła znajdują się potomki przechowujące wartości nie większe niż węzeł rodzicielski, po prawej zaś większe. Taka struktura danych umożliwia łatwe sortowanie słów w kolejności alfabetycznej (rosnącej).
Ponadto każdy węzeł w drzewie posiada wskaźnik na głowę i ogon listy jednokierunkowej, która zawiera numery linii występowania danego słowa w węźle.
Rysunek 1 przedstawia aktualnie zastosowana strukturę danych.



Rysunek 1

2.2 Algorytmy

Program sortuje słowa poprzez umieszczenie je w binarnym drzewie poszukiwań. Wypisanie słów posortowanych alfabetycznie jest realizowane poprzez rekurencyjne przejście przez drzewo. Koszt wykonania podstawowych operacji w drzewie BST (wstawienie, wyszukanie) jest proporcjonalny do wysokości drzewa *h*. Dla drzewa o *n* węzłach optymistyczny koszt każdej z podstawowych operacji wynosi O (log *n*). W przypadku pesymistycznym (gdy drzewo zregeneruje do listy) mamy O (n). [1]

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Do programu należy przekazać nazwy plików po odpowiednich przełącznikach odpowiednio: -i dla pliku wejściowego i -o dla pliku wyjściowego, np.

```
Projekt.exe     -i   Tekst.txt     -o   Słowa.txt
Projekt.exe     -o   Słowa.txt     -i   Tekst.txt
```

Wybrane pliki są plikami tekstowymi. Przełączniki mogą być podane w dowolnej kolejności. Uruchomienie programu bez żadnego parametru lub z parametrem -h powoduje wyświetlenie krótkiej pomocy.

```
**********************************************************
[ HELP ] WPROWADZONE ARGUMENTY SĄ NIEPOPRAWNE
[ HELP ] SPRÓBUJ PONOWNIE
[ HELP ] PRZYKŁADOWY WYGlĄD POPRAWNIE WPROWADZONYCH DANYCH
[ HELP ]     -i     ŚCIEŻKA DO PLIKU WEJŚCIOWEGO
[ HELP ]    -o     ŚCIEŻKA DO PLIKU WYJŚCIOWEGO
**********************************************************
```

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

- ❖ dla -i
  - *Nie podano argumentu wyjściowego* lub *Nie określono ścieżki wejściowej* lub *Nie udało się otworzyć pliku wejściowego z podanej ścieżki.*
- ❖ dla -o

  Nie podano argumentu wyjściowego lub *Nie określono ścieżki wyjściowej* lub *Nie udało się otworzyć pliku wyjściowego z podanej ścieżki.*
- ❖ W przypadku poprawnych argumentów , ale pustego pliku wejściowego

  Plik jest pusty.

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs (komunikacje z użytkownikiem) od logiki aplikacji (sortowania słów).

4.1 Typy zdefiniowane w programie

W programie zdefiniowano następujący typy:

```
struct wezel {
     string slowo;  // Dane słowo
     List * glowa_listy = nullptr; // Wskaźnik na głowę listy dla danego słowa
     List * ogon_listy = nullptr; // Wskaźnik na ogon listy dla danego słowa
     wezel * lewy = nullptr; // Wskaźnik na lewego syna
     wezel * prawy = nullptr; // Wskaźnik na prawego syna
};
```

Typ ten służy do zbudowania drzewa binarnego.
Zdefiniowano także typ przechowujący wskaźnik na korzeń nowo powstałego drzewa.

```
struct BTS {
     wezel  * korzen; // Wskaźnik na pierwszy element drzewa (korzeń)
};
```

Typ poniżej służy do zbudowania listy dla każdego elementu drzewa.

```
struct List {
     int numer; // Numer linii, w której występuje dane słowo
     List * nast; // Wskaźnik na następny element listy
};
```

4.2 Ogólna struktura programu

W funkcji głównej wywoływane są funkcje:

```
int zaladowanie_pliku (int argc,  char ** argv , ifstream & plik_wejsciowy , ofstream & plik_wyjsciowy );
```

Funkcja ta sprawdza, czy program został wywołany w prawidłowy sposób z odpowiednio sformułowanymi argumentami.

Jeśli funkcja otrzyma prawidłowe parametry zwraca wartość typu int, która jest zapisywana w zmiennej *int wynik,* następnie w zależności od wyniku program jest zamykany, bądź wywoływane są kolejne operacje. Gdy program nie został wywołany prawidłowo zostaje wypisany stosowny komunikat (patrz str. 4), jak również następuje wywołanie funkcji *void pomoc()* i zamknięcie programu. Po sprawdzeniu parametrów przechowywane są nazwy pliku wejsciowego i wyjsciowego.

```
void pomoc ();
```

Funkcja ta wyświetla komunikat zawierający pomocne informacje w przypadku niepoprawnego wprowadzenia argumentów podczas uruchamiania programu (patrz str. 4).

Następnie wywoływaną funkcja jest :

```
BTS * utworz_drzewo ();
```

Funkcja ta tworzy wskaźnik na nowo utworzoną strukturę *BTS* , uzupełnia zawartość struktury ustawiając znajdujący się w niej wskaźnik *wezeł * korzeń* na *nullptr*, a następnie zwraca swoją wartość.

```
bool wczytaj_tekst ( BTS * info, ifstream & plik_wejsciowy );
```

Funkcja ta otwiera plik wejściowy, wczytuje linie i eliminuje ewentualne błędy użytkownika, a następnie dla każdego pojedynczego słowa wywołuje funkcję *buduj_drzewo()*, która umieszcza je w drzewie binarnym.
Funkcja korzysta z zawartości nowo utworzonej struktury BTS, a dokładnie z wskaźnika wezel * *korzen* przekazując go do funkcji *buduj_drzewo()*.
Funkcja wywołuje się aż do końca pliku i zwraca wartość true, co oznacza że plik wejściowy został załadowany poprawnie do drzewa.
Po sczytaniu wszystkich linii funkcja zamyka plik wejściowy.

```
void buduj_drzewo ( BTS * info , string nazwa , int nr_lini );
```

Jeżeli jeszcze nie występuje pierwszy element drzewa czyli korzeń to funkcja tworzy nowy węzeł i uzupełnia go danymi (*słowo, wskaźnik na syna lewego i prawego = nullptr*) , a następnie ustawia wskaźnik *korzen* z struktury *BTS* na ten element. W celu dodania numeru linii wystąpienia danego słowa zostaje wywołana funkcja dodajList() dla danego węzła. W przypadku, gdy korzeń już istnieje odsyła do funkcji *dodaj_el_do_drzewa()*.

```
void dodaj_el_do_drzewa ( wezel * korzen , string nazwa , int
nr_lini ) ;
```

```
void dodaj_el_do_drzewa(wezel * korzen, string nazwa, int nr_lini)
```

Funkcja ta dodaje nowy element do istniejącego już drzewa i uzupełnia go odpowiednimi danymi *(jak wyżej)* wywołując przy tym funkcję `dodajList()`. Funkcja odwołuje się do wyniku funkcji *porownaj_napisy()*, której wynik decyduje po której stronie drzewa powinno zostać przydzielone dane słowo. Odpowiednio wartość równa jeden odpowiada lewej stronie, wartość dwa prawej, za to w przypadku wartości równej zero wywoływana jest jedynie funkcja `dodajList()`.Umieszczanie elementów opiera się na algorytmie drzewa poszukiwań, w którym lewe poddrzewo każdego węzła zawiera wyłącznie elementy o kluczach mniejszych niż klucz węzła, a prawe poddrzewo zawiera wyłącznie elementy o kluczach nie mniejszych niż klucz węzła. Węzły, oprócz klucza (*dane słowo*), przechowują wskaźniki na swojego lewego i prawego syna.

```
int porownaj_napisy (wezel * korzen, string nazwa ) ;
```

Funkcja porównuje słowo zawierające się w aktualnym węźle z dopiero co wprowadzonym słowem. Podprogram korzysta z funkcji zewnętrznych *strcoll() i _stricmp(),* które porównują dwie tablice typu char , znak po znaku , traktując duże litery jako małe i uwzględniając przy tym kolejność polskich liter. Porównanie funkcja *strcoll()* jest zinterpretowane zgodnie z kategorią *LC_COLLATE* aktualnie wybranego języka C++ , w naszym przypadku *Polish.*

Funkcje zewnętrzne zwracają wartość większą, mniejsza lub równa 0, co następnie jest odpowiednio interpretowane przez funkcje i zwracane w postaci wartości :

- 0 - w przypadku gdy oba słowa są sobie równe.
- 1 - w przypadku gdy wprowadzone słowo jest pierwsze w kolejności alfabetycznej, a za nim aktualne słowo w węźle.
- 2 - w przypadku odwrotnym.

```
void dodajList ( wezel *listInfo , int nr_lini ) ;
```

Funkcja tworząca nowa listę dla danego słowa, bądź dodająca nowy element do już istniejącej listy. Funkcja wypełnia każdy element listy o numer linii wystąpienia danego słowa i odpowiednio ustawia wskaźnik na kolejny element. Każdy węzeł drzewa zawiera wskaźniki na głowę i ogon swojej listy jednokierunkowej.

```
bool zapisz_tekst( wezel *korzen ,ofstream & plik_wyjsciowy };
```

Funkcja zapisująca zawartość drzewa wraz z odpowiednią zawartością listy w kolejności alfabetycznej do pliku wyjściowego. Funkcja wywoływana jest rekurencyjnie, aby dostać się do najmniejszego elementu w drzewie. Przechodzenie drzewa odbywa się w głąb (*depth-first trawersal*) w porządku poprzeczny (*inorder*), więc rozpoczyna się od lewego poddrzewa, następnie korzenia i na końcu prawego poddrzewa.

```
void zwolnij_pamiec (wezel * korzen ) ;
```

Funkcja ta pozwala zwolnić zaalokowana pamięć dla dynamicznych struktur danych, aby uniknąć tak zwanego wycieku pamięci. Funkcja wywoływana jest rekurencyjnie dopóki nie natrafi na węzeł do usunięcia czyli taki, który nie posiada żadnych potomków. Wraz z usunięciem węzła jest również usuwana lista, którą ten węzeł posiadał.

5 Testowanie

Program został wielokrotnie przetestowany pod kątem różnorodnych błędów użytkownika. Program jest odporny na błędy wprowadzanych danych (brak plików, błędne dane, błędny format danych, niepoprawne parametry uruchomienia programu), wykrycie którekolwiek z nich powoduje wyświetlenie pomocy wraz z komunikatem o zaistniałym błędzie (patrz str.4).
W przypadku pustego pliku wywołany zostaje odpowiedni komunikat o błędzie i zamknięcie programu. Pliki zawierające błędy użytkownika (puste linie, dodatkowe spacje, białe znaki) są dopuszczane przez program, nie powodują wywołania komunikatu o błędzie, gdyż na etapie wczytywania tekstu są eliminowane przez program samoistnie. Program został również sprawdzony pod kątem wycieków pamięci.

6 Wnioski

Program do sortowania słów z użyciem drzewa poszukiwań wydawał się prostym programem, ale na kartce. Podczas tworzenia programu, natrafiłam na wiele przeszkód, które z czasem udało mi się przezwyciężyć dzięki poświęconej

pracy i samozaparciu. Największy kłopot sprawiło mi poprawne skompilowanie drzewa, gdyż wymagało ono zrozumienia działania rekurencji i ostrożności przy alokowaniu pamięci. Stworzenie tego projektu znacznie ułatwiło mi zrozumienie działania wskaźników i dynamicznych struktur danych, co znacznie wpłynęło na mój komfort pracy w środowisku programistycznym.

7 Literatura

[1] https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84