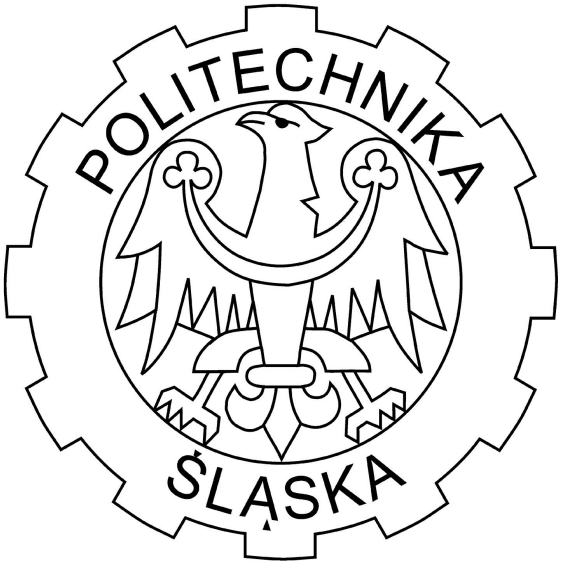
Politechnika Śląska w Gliwicach
Wydział Informatyki, Elektroniki i Informatyki



Podstawy Programowania Komputerów

Autor **Olaf Kocyła**
Prowadzący **mgr inż. Marek Kokot**
Rok akademicki **2017/2018**
Kierunek **Teleinformatyka**
Rodzaj studiów **SSI**
Semestr **1**
Termin laboratorium / ćwiczeń **piątek, 10:00 - 11:30**
Grupa **2**
Sekcja **4**
Termin oddania sprawozdania **2018-01-25**
Data oddania sprawozdania **2018-01-25**

1. Treść zadania

Napisać program, który odczytuje z pliku listę książek zapisaną w sposób: (autor); (tytuł); (etykieta) [,(etykieta)] i zapisuje do pliku książki według etykiet.

- Przykład listy książek:

Lem; Summa Technologiae; futurologia, filozofia
Platon; Uczta; filozofia
Montgomery; Ania z Zielonego Wzgórza; powieść, lektura
Dante; Boska Komedia; tragedia, lektura
Mickiewicz; Pan Tadeusz; klasyka, romantyzm, lektura
Toffler; Budowa nowej cywilizacji; futurologia
Żeromski; Ludzie Bezdomni; lektura

- Plik wyjściowy po uruchomieniu programu:

romantyzm:
Mickiewicz; Pan Tadeusz

klasyka:
Mickiewicz; Pan Tadeusz

tragedia:
Dante; Boska Komedia

lektura:
Dante; Boska Komedia
Mickiewicz; Pan Tadeusz
Montgomery; Ania z Zielonego Wzgórza
Żeromski; Ludzie Bezdomni

powieść:
Montgomery; Ania z Zielonego Wzgórza

filozofia:
Lem; Summa Technologiae
Platon; Uczta

futurologia:
Lem; Summa Technologiae
Toffler; Budowa nowej cywilizacji

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

-**i** plik wejściowy
-**o** plik wyjściowy

2. Analiza zadania

Zagadnienie przedstawia problem sortowania książek zapisanych w określony schemat. Program musi odczytać plik, sprawdzić czy książki zapisane są według schematu, a następnie posortować według etykiet do odpowiedniej struktury danych. Na koniec program powinien zapisać pogrupowane książki według etykiet do pliku wyjściowego.

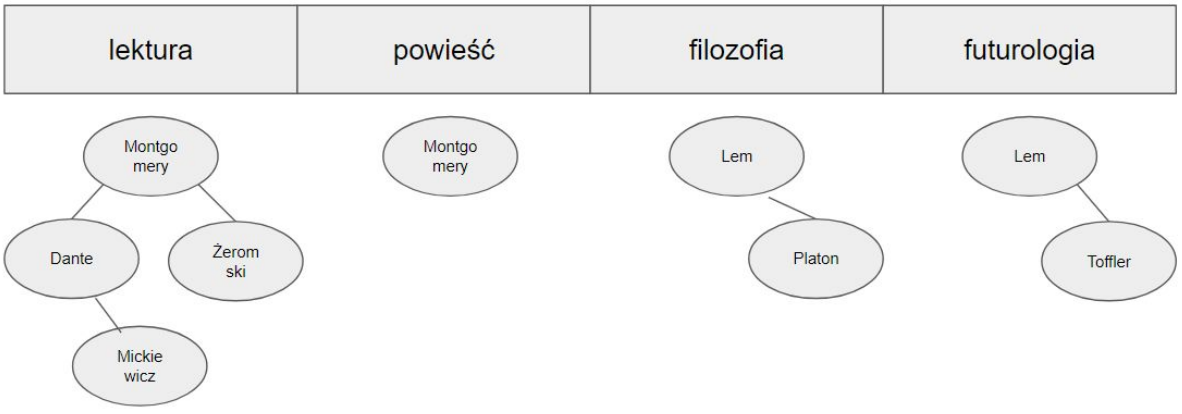
3. Specyfikacja zewnętrzna

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

-i input.txt **-o** output.txt

4. Struktura danych

Przykład reprezentacji danych za pomocą listy etykiet oraz drzew binarnych z książkami (dla zachowania czytelności przedstawiono tylko kilka etykiet z końca listy dla zadanego pliku wejściowego):



Podczas wczytywania danych z pliku wejściowego dane zapisywane są w następującej strukturze (jak na ilustracji powyżej):

- etykiety zapisywane są na jednokierunkowej liście wiązanej - dodanie nowej etykiety odbywa się poprzez wstawienie jej na początek listy oraz przesunięcie wskaźników:
    - wskaźnik na kolejny po wstawionym elemencie przesuwany jest na bieżący początek listy
    - następnie wskaźnik pierwszego elementu przesuwany jest na nowo wstawiony element
- książki należące do danej etykiety zapisywane są na drzewach binarnych, tj. pod każdą etykietą budowane jest drzewo binarne (budowane zg. z porządkiem alfabetycznych wg nazwisk autorów) trzymające referencje do książek - wykorzystanie drzewa binarnego ma następujące zalety:
    - ułatwia utrzymanie porządku alfabetycznego książek
    - umożliwia znacznie szybsze przeszukiwanie (i tym samym wstawianie) niż w przypadku zwykłej listy

Implementacja opisanej wyżej struktury w programie (plik “struktura.hpp”):

```
struct ksiazka {
	string autor;
	string tytul;
	int licznik_referencji;
};
```

Struktura opisująca pojedynczą książkę - zawiera jej autora i tytuł (wczytywane z pliku wejściowego) oraz licznik referencji - umieszczenie książki pod kilkoma etykietami spowoduje zwiększenie licznika, zamiast utworzenia książki kilka razy w pamięci. Utrzymywanie licznika potrzebne jest do prawidłowego zwolnienia pamięci po zakończeniu przetwarzania danych (książka nie może zostać usunięta, dopóki nie zostaną usunięte wszystkie referencje do niej - zapobiega to potencjalnym wyciekom pamięci).

```
struct ksiazka_wezel {
	ksiazka *ksiazka_ptr;
	struct ksiazka_wezel *left;
	struct ksiazka_wezel *right;
};
```

Struktura opisująca węzeł drzewa binarnego z książkami pod pojedynczą etykietą. Węzeł przechowuje wskaźnik na książkę, a także referencje do jej lewego oraz prawego dziecka. Dziecko lewe wskazuje na książkę, której autor ma naziwsko alfabetycznie wcześniej, a dziecko prawe - dalej.

```
struct etykieta_elem {
```

```
	string nazwa;
	struct ksiazka_wezel *ksiazka_korzen;
	struct etykieta_elem *nastepny;
};
```

Struktura opisuje element listy z etykietami. Zawiera nazwę etykiety, wskaźnik na kolejny element listy oraz wskaźnik na węzeł, będący korzeniem drzewa z książkami przypisanymi do etykiety.

5. Opis funkcji w programie:

Najważniejsze funkcje do tworzenia struktury danych (plik “struktura.cpp”):

```
static void wstaw_wezel_ksiazki(ksiazka_wezel **korzen, ksiazka *ksiazka)
{
	ksiazka_wezel *wezel;
	if (!*korzen) {
		*korzen = utworz_wezel(ksiazka);
		return;
	}

	wezel = *korzen;

	if (ksiazka->autor < wezel->ksiazka_ptr->autor) {
		if (wezel->left) {
			wezel = wezel->left;
		} else {
			wezel->left = utworz_wezel(ksiazka);
			return;
		}
	} else {
		if (wezel->right) {
			wezel = wezel->right;
		} else {
			wezel->right = utworz_wezel(ksiazka);
			return;
		}
	}

	wstaw_wezel_ksiazki(&wezel, ksiazka);
}
```

Funkcja służy do umieszczania nowej książki w drzewie binarnym o zadanym korzeniu (“**korzen”). Jeśli książka jest pierwszą w drzewie to wykonanie funkcji kończy się na utworzeniu nowego węzła i umieszczeniu go w korzeniu. W przeciwnym wypadku nazwisko autora książki porównywane jest z nazwiskiem autora książki w korzeniu. Jeśli nazwisko jest wcześniej w porządku alfabetycznym - książka powinna trafić do lewego poddrzewa. Jeśli dalej - do poddrzewa prawego. Funkcja wywoływana jest rekursywnie aż do momentu znalezienia “wolnego” miejsca dla nowej książki (tj. wskaźnik na lewe lub prawe poddrzewo wskazuje na null).

```
static etykieta_elem* wstaw_etykiete(etykieta_elem **head, const string&
name)
{
	etykieta_elem *sprawdzany;
	sprawdzany = *head;

	while (sprawdzany) {
		if (sprawdzany->nazwa == name) {
			return sprawdzany;
		}

		sprawdzany = sprawdzany->nastepny;
	}

	sprawdzany = new etykieta_elem;
	sprawdzany->nazwa = name;
	sprawdzany->ksiazka_korzen = NULL;
	sprawdzany->nastepny = *head;

	*head = sprawdzany;

	return sprawdzany;
}
```

Funkcja służy do dodawania etykiety do listy z etykietami (lub odnajdywania, jeśli już się wcześniej pojawiła). Na wejściu podawana jest nazwa etykiety oraz wskaźnik na początek listy. Funkcja przechodzi listę element po elemencie korzystając z wskaźników “nastepny” w każdym. Dla każdego elementu sprawdza, czy nazwa jest taka sama jak wstawianej etykiety - jeśli tak, przerywa przeszukiwanie i zwraca wskaźnik na odnaleziony element. W przeciwnym wypadku dodaje etykietę do listy. Nowy element dodawany jest na początku listy w następujący sposób:

- tworzony jest element listy (“etykieta_elem”) z zadaną nazwą
- wskaźnik na następny element ustawiany jest na aktualny pierwszy element listy
- wskaźnik pierwszego elementu listy przesuwany jest na dodany element (jest to nowy początek listy)

```
void wstaw_ksiazke(etykieta_elem **korzen, const string& etykieta, ksiazka
*ksiazka)
{
	etykieta_elem *item;

	item = wstaw_etykiete(korzen, etykieta);

	wstaw_wezel_ksiazki(&item->ksiazka_korzen, ksiazka);
}
```

Funkcja umieszcza nową książkę pod zadaną etykietą korzystając z opisanych wyżej funkcji. Najpierw wywołuje funkcję “wstaw_etykiete”, której wynikiem etykieta z listy etykiet, do której powinna zostać dopisana książka. Następnie funkcja “wstaw_wezel_książki” umieszcza książkę w drzewie przypisanym do etykiety.

```
void struktura_do_pliku(etykieta_elem *root, ostream& os)
{
	while (root) {
		os << root->nazwa << ":" << endl;

		przejdź_drzewo(root->ksiazka_korzen, os);
		os << endl;

		root = root->nastepny;
	}
}

static void przejdź_drzewo(ksiazka_wezel *wezel, ostream& os)
{
	if (!wezel) {
		return;
	}

	przejdź_drzewo(wezel->left, os);

	os << wezel->ksiazka_ptr->autor << "; " << wezel->ksiazka_ptr->tytul
<< endl;

	przejdź_drzewo(wezel->right, os);
}
```

Powyższe funkcje przekształcają strukturę (listę etykiet z drzewami książek) do docelowego formatu i zapisują je w pliku wyjściowym. Dla każdej etykiety drukowana jest jej nazwa, a następnie książki w porządku alfabetycznym wg autorów. Odtworzenie porządku alfabetycznego książek możliwe jest dzięki algorytmowi przechodzenia

drzewa w głąb. Przejście to realizowane jest z pomocą rekursywnych wywołań funkcji “przejdz_drzewo” na lewym i prawym poddrzewie.

```
void zwolnij_pamiec(etykieta_elem **root)
{
	etykieta_elem *item;
	etykieta_elem *tymczasowy;

	item = *root;

	while (item) {
		kasuj_wezel(item->ksiazka_korzen);

		tymczasowy = item;
		item = item->nastepny;
		delete tymczasowy;
	}

	*root = NULL;
}

static void kasuj_wezel(ksiazka_wezel *wezel)
{
	if (!wezel) {
		return;
	}

	kasuj_wezel(wezel->left);
	kasuj_wezel(wezel->right);

	wezel->ksiazka_ptr->licznik_referencji--;
	if (wezel->ksiazka_ptr->licznik_referencji == 0) {
		delete wezel->ksiazka_ptr;
	}

	delete wezel;
}
```

Powyższe funkcje służą do zwolnienia pamięci po zapisaniu danych do pliku wyjściowego. Funkcja “zwolnij_pamiec” przechodzi kolejne elementy listy z etykietami i usuwa powiązane z nimi drzewa binarne z książkami. Kasowanie odbywa się w sposób podobny, co w przypadku przechodzenia drzewa w celu jego wydrukowania, tj. rekursywnie przechodzone jest kolejno lewe i prawe poddrzewo. Dla każdego węzła zmniejszany jest licznik referencji umieszczonej w nim książki. Węzeł jest usuwany natychmiastowo, natomiast książka dopiero, kiedy jej licznik referencji osiągnie wartość 0, tzn. nie będzie już odwołań do niej w żadnym innym drzewie. Usuwanie węzłów

odbywa się zaczynając od węzłów na samym dole drzewa aż po jego korzeń. Taka kolejność gwarantuje, że usunięcie węzła nie spowoduje “zgubienia” referencji do wciąż istniejących węzłów-dzieci, co mogłoby prowadzić do wycieków pamięci. Po usunięciu drzewa można usunąć etykietę - dzieje się to ponownie w funkcji “zwolnij_pamiec”. W tym celu zastosowano zmienną pomocniczą “tymczasowy” - do zmiennej tej zapisywana jest etykieta przeznaczona do usunięcia. Następnie do zmiennej “item” zapamiętywany jest kolejny element listy - zabezpiecza to przed utratą referencji z chwilą usunięcia bieżącego elementu. No koniec można usunąć etykietę zapamiętaną w zmiennej “tymczasowy”.

Funkcje pomocnicze (plik “pomocnicze.cpp”) do przetwarzania danych z pliku wejściowego:

```
string trim(string linia)
{
	int pozycja;

	pozycja = linia.find_first_not_of(' ');
	if (pozycja != string::npos) {
		linia.erase(0, pozycja);
	}

	pozycja = linia.find_last_not_of(' ');
	if (pozycja != string::npos) {
		linia.erase(pozycja + 1, string::npos);
	}

	return linia;
}
```

Funkcja służy do przycinania tekstu, tj. pozbywania się spacji z początku i końca. Funkcja odnajduje pozycję pierwszego znaku innego niż spacja, a następnie usuwa wszystkie znaki od początku do wyznaczonej pozycji. Następnie znajduje ostatnią pozycję znaku innego niż spacja i usuwa znaki od wyznaczonej pozycji do końca napisu.

Funkcja korzysta z funkcji bibliotecznych std::string:

- erease(pozycja, liczba_znakow) - usuwanie określonej liczby znaków poczynając od wskazane pozycji
- find_first_not_of(znak) - zwraca pozycję pierwszego znaku innego niż zadany (jeśli nie znajdzie to zwraca string::npos)

```
bool separuj(string& linia, char separator, string& wynik)
{
	int pozycja;

	pozycja = linia.find_first_of(separator);
```

```
	if (pozycja == string::npos) {
		return false;
	}

	wynik = trim(linia.substr(0, pozycja));
	linia.erase(0, pozycja + 1);

	return true;
}
```

Funkcja służy do odczytywania danych z linijki z pliku CSV poprzez jej podział za pomocą zdefiniowanego separatora. Najpierw funkcja odnajduje pozycję separatora (np. średnika), a następnie odczytuje tekst, który znajduje się przed nim – jest to wynik odczytu. Odczytana wartość oraz separator są natomiast "odcinane" z początku linii (zmienna "linia"). Obcięta linijka może potem być ponownie przekazana do kolejnego wywołania tej funkcji w celu odczytania kolejnych danych. Jeśli odczyt się powiedzie funkcja zwraca "true", w przeciwnym wypadku "false" – wtedy program może np. zgłosić błąd.

6. Testowanie

Program został przetestowany na różne sposoby. W przypadku błędu zostają wyświetlone odpowiednie komunikaty. Program kasuje puste znaki dzięki funkcji *trim* oraz radzi sobie z pustymi liniami. Nie występują wycieki pamięci.

7. Literatura:

Bruce Eckel, *Thinking in C++*

Robert Sedgewick, *Algorytmy w C++*