Politechnika Śląska w Gliwicach

Wydział Informatyki, Elektroniki i Informatyki

**Podstawy Programowania**

**Komputerów**

Książki

|  |  |
| --- | --- |
| autor | Mateusz Wilczyński |
| prowadzący | dr inż. Karolina Nurzyńska |
| rok akademicki | 2014/2015 |
| kierunek | informatyka |
| rodzaj studiów | SSI |
| semestr | 2 |
| termin laboratorium / ćwiczeń | Czw. 08:30 – 10:00 |
| grupa | 4 |
| termin oddania sprawozdania | 2015-06-11 |
| data oddania sprawozdania | 2015-06-11 |
|  |  |

# Treść zadania

Książki w bibliotece opisane są przez nazwisko autora, tytuł i etykiety. Etykiety przyporządkowują książki do działów. Każda książka może mieć kilka etykiet. Zadanie polega na zapisaniu do pliku książek wg etykiet. Książki opatrzone tą samą etykietą są posortowane wg nazwiska autora.

# Analiza zadania

Zagadnienie przedstawia problem wczytania z pliku danych, a następnie zapisanie ich do innego pliku w zmienionym formacie.

## Struktury danych

Etykiety są przechowywane w sortowanej liście jednokierunkowej. Nazwiska autorów oraz tytuły są zapisane w liście podwieszanej pod etykietami. Nazwisko i tytuł są przechowywane w jednej strukturze danych. Przedstawiona struktura danych jest strukturą dynamiczną umożliwiającą dodawanie różnej ilości danych.

## Algorytmy

Algorytm służący do wczytywania z pliku wczytuje autora, tytuł oraz listę etykiet. Po wczytaniu następuje sprawdzenie, czy dane są poprawne, oraz czy format linii jest odpowiedni. Jeśli jakieś kryterium nie jest spełnione, to dalsze linie nie są wczytywane. Jeśli wczytanie autora, tytuł oraz listy etykiet przebiegło pomyślnie następuje dodanie tych wartości do dynamicznej struktury danych. Lista etykiet jest dzielona na pojedyncze etykiety, dla każdej etykiety następuje dodanie jej do listy etykiet, oraz stworzenie pod nią listy podwieszanej zawierającej odpowiedniego autora oraz tytuł. Jeśli dana etykieta istnieje już w liście, to autor oraz tytuł są dodane pod istniejącą etykietę. Zarówno etykiety jak i autor oraz tytuł są dodawane do odpowiednich list alfabetycznie. Lista podwieszana jest dodawana alfabetycznie na podstawie nazwiska autora, w przypadku gdy jest ono takie same dla kilku książek, to są one ułożone na podstawie tytułu. Po dodaniu do listy wszystkich etykiet z danej linii algorytm przechodzi do następnej.

Algorytm zapisujący wynik działania do pliku zaczyna od głowy listy (*head),*  która zostaje przypisana do lokalnej zmiennej wskaźnikowej *wsk*. Na początku następuje zapisanie pola *label*  struktury na którą wskazuje wskaźnik do pliku, następnie odbywa się zapis listy podwieszonej do danego wskaźnika. Rozpoczyna się ono od elementu *wsk->under*  jest to pierwszy element listy podwieszanej, następuje zapis pola *line* tego elementu do pliku, następnie przechodzimy do elementu *next* i znowu zapisujemy jego pole *line*  do pliku. Powtarzamy to dopóki element *next* jest różny od *NULL,* wtedy kończy się zapis listy podwieszonej pod daną etykietę i następuje przejście do następnej etykiety poprzez *wsk->next*. Wyżej opisane działania są powtarzane dopóki *wsk->next* jest różny od *NULL.*

# Specyfikacja zeewnętrzna

Program można uruchomić otwierając plik slownik.exe. Po uruchomieniu program wczytuje wstępną konfigurację z pliku *config.txt*. W pliku w jednej linii wpisujemy atrybut, a w następnej jego wartość. Można w nim określić maksymalną długość listy ostatnio używanych słów (*max\_length*) oraz nazwę pliku z danymi (*file\_name*). Przykładowy plik jest dołączony do programu. Jeżeli plik jest niepoprawny lub nie został znaleziony zostanie wyświetlony komunikat: *plik config.txt nie istnieje lub jest niepoprawn.*

Po wczytaniu pliku configu następuje wczytanie pliku z danymi. Jeśli plik nie istnieje lub ma niewłaściwy format zostanie wyświetlony komunikat: *plik z danymi nie istnieje lub ma niewlasciwy format.*

Jeśli oba powyższe pliki mają właściwy format wyświetla się menu aplikacji, w przeciwnym wypadku po naciśnięciu *Enter* program kończy działanie.

Opcje menu:

0 – wpisanie *0*  i zatwierdzenie *Enter*  powoduje wyjście z programu.

1 – wyszukiwanie hasła. Pojawia się komunikat *wpisz slowo*  po wpisaniu słowa i zatwierdzeniu Enterem następuje wyszukiwanie słowa. Jeśli słowo zostanie znalezione w pliku lub w liście zostanie wyświetlone to słowo oraz jego definicja. Jeśli słowo nie zostało znalezione zostanie wyświetlony komunikat: *Slowo nie zostalo znalezione*.

2 – dodanie nowego hasła. Pojawia się komunikat *wpisz slowo*. Po wpisaniu słow zatwierdzamy. Następnie pojawia się komunikat *wpisz opis*. Po wpisaniu i zatwierdzeniu opisu, jeśli wszystko przebiegło prawidłowo pojawia się komunikat *dodano definicję* a definicja została zapisana do pliku z danymi. Jeśli słowo lub definicja będzie pustym ciągiem znaków pojawi się komunikat *niepoprawne slowo lub definicja*

3 – zmiana długości listy. W tym punkcie można zmienić ilość słów zapamiętanych w liście. Po wybraniu tej opcji pojawia się komunikat *podaj nowa maksymalna długość listy*. Następnie podajemy nową długość listy i zatwierdzamy.

4 – zmiana pliku z danymi. Można w tym podpunkcie wybrać inny plik z którego mają być pobierane dane. Pojawi się komunikat *podaj nowa nazwe pliku z danymi* po podaniu nazwy pliku program sprawdzi czy plik istnieje i jest poprawny, jeśli nie, zostanie wyświetlony komunikat*: plik z danymi nie istnieje lub ma niewlasciwy format.* Jeśli plik jest poprawny zostanie wyświetlony komunikat: *plik został załadowany poprawnie.*

Po zakończeniu wykonywanie opcji 1 – 4 program ponownie wyświetli menu.

# Specyfikacja wewnętrzna

## Zmienne

|  |  |
| --- | --- |
| F | Zmienna typu **Text**, zmienna ta umożliwia nam operowanie na pliku z danymi na których działa program/ |
| head | Zmienna typu **wsk**, przechowujemy w niej wskaźnik na początek listy. |
| tail | Zmienna typu **wsk**, przechowujemy w niej wskaźnik na koniec listy. |
| max\_length | Zmienna typu **integer**, służy do przechowywania maksymalnej dozwolonej długości listy. |
| current\_list\_length | Zmienna typu **integer**, służy do przechowywania obecnej długości listy. |
| decision | Zmienna typu **string**, umożliwia wybieranie odpowiedniej opcji z menu oraz wybranie odpowiedniej procedury do obsługi wyboru. |
| correct | Zmienna typu **boolean,** jest używana do monitorowania przebiegu wczytywania wstępnej konfiguracji. |

## Funkcje i Procedury

|  |
| --- |
| **function load\_initial\_config(var max\_length:integer; var file\_name:string):boolean;** |
| Funkcja służy do wczytywania wstępnej konfiguracji.  Parametry:   * max\_length – referencja do zmiennej, której ma zostać przypisana maksymalna długość listy, * file\_name - referencja do zmiennej, której ma zostać przypisana nazwa pliku z danymi.   Funkcja zwraca *true* jeśli przypisała poprawne wartości parametrom, lub *false* jeśli wystąpiły jakieś problemy. |

|  |
| --- |
| **function load\_file(var F:Text; file\_name:string):boolean;** |
| Funkcja przypisuje plik o nazwie podanej w file\_name do zmiennej F.  Parametry:   * F – zmienna do której ma zostać przypisany plik, * file\_name – nazwa pliku, który ma zostać przypisany.   Funkcja zwraca *true*  jeśli wykonanie przebiegło pomyślnie, lub *false* jeśli plik nie został znaleziony lub ma niepoprawny format. |

|  |
| --- |
| **function find\_in\_list(var head:wsk; key:string; var definition:definition\_type):boolean;** |
| Funkcja szuka w liście odpowiednie definicji na podstawie klucza.  Parametry to :   * head – głowa listy, * key - klucz, na podstawie którego odbywa się wyszukiwanie, * definition – zmienna do której zostanie wstawiona znaleziona definicja.   Funkcja zwraca *true* jeśli definicja została znaleziona w liście, lub false w przeciwnym wypadku. |

|  |
| --- |
| **function deserialize\_line(line:string; var word:string; var description:string):boolean;** |
| Funkcja dokonuje przetwarzania linii tekstu: wyodrębnia słowo i jego opis.  Parametry:   * line – linia, która ma zostać przetworzona, * word – zmienna do której zostanie wstawione słowo, * description – zmienna do której zostanie wstawiony opis.   Funkcja zwraca *true* jeśli udało się odnaleźć słowo i jego opis lub *false* w przeciwnym wypadku |

|  |
| --- |
| **function find\_in\_file(var F:Text; key:string; var definition:definition\_type):boolean;** |
| Funkcja szuka w pliku odpowiednie definicji na podstawie klucza.  Parametry:   * F – plik, w którym ma odbyć się wyszukiwanie, * key - klucz, na podstawie którego odbywa się wyszukiwanie, * definition – zmienna do której zostanie wstawiona znaleziona definicja.   Funkcja zwraca *true* jeśli definicja została znaleziona w pliku, lub *false* w przeciwnym wypadku. |

|  |
| --- |
| **procedure append\_to\_list(var head:wsk; var tail:wsk; definition:definition\_type; var current\_list\_length:integer; max\_length:integer);** |
| Procedura dodaje definicję na początek listy i w razie potrzeby usuwa elementy z końca listy.  Parametry:   * head – głowa listy, * tail – ogon listy, * definition –definicja, która ma zostać wstawiona do listy, * current\_list\_length – obecna długość listy, * max\_length – maksymalna długość listy. |

|  |
| --- |
| **function find\_definition(var definition:definition\_type; var head:wsk; var tail:wsk; key:string; var F:Text; var current\_list\_length:integer; max\_length:integer):boolean;** |
| Funkcja obsługuje wyszukiwanie definicji poprzez uruchamianie odpowiednich funkcji i procedur.  Parametry:   * definition – zmienna do której zostanie wstawiona znaleziona definicja, * head – głowa listy, * tail – ogon listy, * key - klucz, na podstawie którego odbywa się wyszukiwanie, * F – plik, w którym ewentualnie ma odbyć się wyszukiwanie, * current\_list\_length – obecna długość listy, * max\_length – maksymalna długość listy.   Funkcja zwraca *true* jeśli definicja została w pliku, lub *false* w przeciwnym wypadku. |

|  |
| --- |
| **function read\_key():string;** |
| Funkcja pobiera od użytkownika słowo klucz, a następnie je zwraca. |

|  |
| --- |
| **procedure print\_definiton(definition:definition\_type);** |
| Procedura wyświetla słowo i opis definicji.  Parametry:   * definition – definicja, której atrybuty mają zostać wyświetlone. |

|  |
| --- |
| **procedure search\_definition(var head:wsk; var tail:wsk; var F:Text; var current\_list\_length: integer; max\_length:integer);** |
| Funkcja obsługuje pobieranie szukanego słowa, wyszukiwanie definicji, oraz wyświetlenie uzyskanej informacji poprzez uruchamianie odpowiednich funkcji i procedur.  Parametry:   * head – głowa listy, * tail – ogon listy, * F – plik, w którym ewentualnie ma odbyć się wyszukiwanie, * current\_list\_length – obecna długość listy, * max\_length – maksymalna długość listy. |

|  |
| --- |
| **function get\_definition(var word:string; var description:string):boolean;** |
| Funkcja pobiera od użytkownika słowo i opis definicji.  Parametry:   * word – zmienna do której zostanie wstawione słowo, * description – zmienna do której zostanie wstawiony opis.   Funkcja zwraca *true* jeśli word i description nie są puste, lub *false* w przeciwnym wypadku. |

|  |
| --- |
| **function save\_definition(var F:Text; word:string; description:string):boolean;** |
| Funkcja zapisuje definicję do pliku.  Parametry:   * F – plik do którego ma zostać zapisana definicja, * word – słowo definicji, * description – opis definicji.   Funkcja zwraca *true* jeśli zapisywanie się powiodło, lub *false* w przeciwnym wypadku. |

|  |
| --- |
| **procedure print\_add\_definition\_result(result:boolean);** |
| Procedura informująca użytkownika o tym, czy powiódł się zapis definicji do pliku.  Parametry:   * result – informacja o tym, czy powiódł się zapis definicji do pliku. |

|  |
| --- |
| **procedure add\_definition(var F:Text);** |
| Procedura do obsługuje dodanie definicji do pliku przy użyciu innych funkcji i porcedur.  Parametry:   * F – plik do którego ma zostać zapisana definicja, |

|  |
| --- |
| **procedure change\_list\_length(var max\_length:integer);** |
| Procedura do obsługi zmiany maksymalnej długości listy.  Parametry:   * max\_length – zmienna do której ma zostać wstawiona nowa maksymalna długość. |

|  |
| --- |
| **procedure change\_file\_name(var F:Text);** |
| Procedura dokonuje zmiany nazwy pliku z danymi oraz przy użyciu innej funkcji sprawdza, czy nowy plik jest poprawny.  Parametry:   * F – zmienna do której ma zostać wstawione odwołanie do nowego pliku. |

|  |
| --- |
| **procedure print\_config\_error(error:integer);** |
| Procedura wypisuje komunikaty odpowiednie dla podanego kodu błędu.  Parametry:   * error – kod błędu. |

|  |
| --- |
| **function load\_and\_check\_initial\_config(var F:Text; var max\_length:integer):boolean;** |
| Funkcja ładuje wstępną konfigurację programu.  Parametry:   * F – zmienna do której ma zostać wstawione odwołanie do pliku z danymi. * max\_length – zmienna do której ma zostać wstawiona maksymalna długość listy.   Funkcja zwraca *true*, jeśli załadowano poprawną konfigurację, lub *false* w przeciwnym wypadku. |

|  |
| --- |
| **procedure dispose\_list(var head:wsk; var tail:wsk);** |
| Procedura czyści całą pamięć zajętą przez listę oraz ustawia wskaźniki *head* i *tail* na *Nil.*  Parametry:   * head – głowa listy, * tail – ogon listy. |

# Testowanie

Program został przetestowany na poprawnych plikach .txt oraz na plikach z innymi rozszerzeniami. Dla niepoprawnych oraz nieistniejących plików program wyświetla informację o błędzie.

Testowanie:

* Zawartość pliku config.txt:

*file\_name*

*dane.txt*

*max\_length*

*2*

Zawartość pliku dane.txt:

*slo1 - opis1*

*slo2 - opis2*

*slo3 - opis3*

* wybór w menu: „*1”,* szukane słowo: *„slo1”,* zwrócona definicja: *„opis1”.* Rezultat poprawny.
* wybór w menu: „*1”,* szukane słowo: *„invali”,* zwrócony rezultat: *„Slowo nie zostalo znalezione”.* Rezultat poprawny.
* wybór w menu: „*2”,* nowe słowo: *„slo4”,* opis: *„opis4”,* zwrócony rezultat: *„dodano definicje”.* Rezultat poprawny oraz nastąpiło dopisanie definicji do pliku dane.txt.
* Zawartość pliku config.txt:

*file\_name*

*dane.txt*

*max\_length*

*2*

Brak pliku dane.txt. Program wypisuje: „*plik z danymi nie istnieje lub ma niewlasciwy format”.* Działanie poprawne.

* Brak pliku config.txt. Program wypisuje: „plik *config.txt nie istnieje lub jest niepoprawny”.* Działanie poprawne.

Zostały również przetestowane pozostałe możliwości i opcje. Testy dały oczekiwane rezultaty.

# Wnioski

Program książki nie jest programem skomplikowanym. Program składa się z kilku ważnych części: listy dwukierunkowej, wyszukiwania w pliku i dodawania do pliku. Program został podzielony na wiele funkcji i procedur, zapewnia to dużą izolację kodu, co jest korzystne przy większych programach, zwiększa to też łatwość późniejszej zmiany kodu.