

Tychy, 16.06.2016

Laboratorium Programowania Komputerów

Temat: Dziekanat

Autor: Magdalena Bajerska

Informatyka , sem.2 , grupa dziekańska 6 , sekcja 3

Prowadzący: dr inż. Karolina Nurzyńska

1. Temat

Dziedzinat

Oceny z egzaminów zbierane są na protokołach egzaminacyjnych. Każdy przedmiot ma odrębny protokół. Protokół ma następujący format: W pierwszej linii umieszczona jest nazwa przedmiotu, w drugiej prowadzący, w kolejnych oceny studentów w postaci krotki:

⟨imię⟩ ⟨nazwisko⟩ ⟨nr albumu⟩ ⟨ocena⟩ ⟨data wpisu⟩

Przykładowy plik protokołu ma postać:

```
Wstep do sztucznej inteligencji
dr inż. Jan Heweliusz
Jan      Jaworek 14332 3.5 2012-02-29
Bartłomiej Bukowy 14344 4.5 2012-01-30
Karol    Krol    12433 3.0 2012-01-28
```

Plików protokołów może być dowolna liczba. Po wykonaniu programu uzyskujemy pliki wynikowe zawierające oceny studentów. Każdy plik zawiera oceny tylko jednego studenta. Nazwa pliku jest tożsama z numerem albumu studenta. W pliku tym podane jest nazwisko studenta i oceny, które uzyskał. Przedmioty opisane są w następujący sposób:

⟨nazwisko prowadzącego⟩ ⟨nazwa przedmiotu⟩ ⟨ocena⟩ ⟨data wpisu⟩

Przedmioty są posortowane wg nazw. Przykładowy plik dla studenta Jana Jaworka:

```
Jan Jaworek
nr albumu: 14332
```

```
dr hab. inż. Mikołaj Kopernik  Mechanika nieba          4.5 2012-01-14
dr inż. Jan Heweliusz         Wstep do sztucznej inteligencji 3.5 2012-02-29
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następującego przełącznika:
-i pliki wejściowe z protokołami

2. Analiza, projektowanie

Do realizacji powyższego projektu posłużyłam się listą list (listą podwieszaną), która świetnie sprawdza się w tego typu zadaniach. Lista studentów jest w tym przypadku główną listą do której podwieszona jest lista z wykazem ocen dla poszczególnych studentów. Protokoły egzaminacyjne mogą zawierać bardzo dużą ilość studentów. Program może odczytywać wiele plików zawierających protokoły egzaminacyjne. Biorąc pod uwagę dwa powyższe warunki wybranie zastosowanie w programie struktur dynamicznych, a konkretniej listy podwieszanej wydaje się być najtrafniejszą opcją ze względu na to, że ta struktura dynamiczna bardzo dobrze operuje na tak dużych ilościach danych. Wadą zastosowania tego typu rozwiązania jest większa możliwość popełnienia błędu podczas pisania kodu i mogą występować wycieki pamięci.

3. Specyfikacja programu

Użytkownik tworzy pliki stanowiące protokoły egzaminacyjne w formacie txt. W pierwszym wierszu pliku tekstowego powinna być umieszczona nazwa przedmiotu. W drugim wierszu pliku tekstowego należy umieścić nazwę prowadzącego.

W kolejnych wierszach umieszczane są informacje o studencie wraz z danymi dotyczącymi jego oceny w następującej kolejności:

- imię,
 - nazwisko,
 - ocena,
 - data wystawienia oceny,
- oddzielone od siebie spacjami.

Przykład prawidłowego uzupełnienia pliku:

```
Podstawy Informatyki
dr inż. Radosław Myśliwski
Agnieszka Melisa 34781 4.5 2016-05-03
Dorota Nadewa 35295 4.0 2016-04-28
Artur Gulik 19372 5.0 2016-05-06
Marek Wolski 10392 4.5 2016-03-06
Anatol Wiza 45282 3.0 2016-08-04
```

Program uruchamiany jest z wiersza poleceń przy pomocy przełącznika „-i” .

W pierwszej kolejności do wiersza poleceń należy wprowadzić nazwę programu, a później po spacji użyć przełącznika „-i” i po kolejnej spacji podawać nazwy plików z protokołami egzaminacyjnymi oddzielając je spacją.

Przykład prawidłowego wprowadzenia argumentów do wiersza poleceń:

Dziekanat.exe -i Informatyka.txt Matematyka.txt

Wiersz poleceń należy wywołać w folderze w którym znajduje się aplikacja. Pliki protokołów najlepiej umieścić w tym samym folderze co aplikacja, aby nie trzeba było do wiersza poleceń wprowadzać całych ścieżek dostępu. Następnie za pomocą klawiszy SHIFT+CTRL i prawego klawisza myszy klikamy na folderze w którym znajdują się protokoły i aplikacja i wywołujemy polecenie „Otwórz okno polecenia tutaj.” i postępujemy zgodnie z instrukcją napisaną powyżej.

W przypadku gdy do wiersza poleceń wprowadzone zostaną błędne argumenty program wyświetli komunikat:

```
Blednie podane argumenty.
Prosze podac inne argumenty.
Przyklad poprawnego podania plikow : Dziekanat.exe -i Informatyka.txt Matematyka.txt .
Nie ma elementow do zapisania.
```

W przypadku gdy użytkownik użyje błędnego przełącznika program wyświetli komunikat:

```
Bledny format danych wejsciowych.  
Nie ma elementow do zapisania.
```

Jeśli pliki podane przez użytkownik nie istnieją program wyświetli informację:

```
Nie ma takiego pliku.  
Nie ma elementow do zapisania.
```

W razie gdy plik z indeksem danego studenta nie zostanie utworzony wyświetli się komunikat:

```
Blad otwarcia pliku.
```

Jeśli lista studentów jest pusta program wyświetli komunikat:

```
Nie ma elementow do zapisania.
```

4. Specyfikacja wewnętrzna

4.1. Deklaracje zmiennych, stałych i wskaźników użytych w programie oraz struktury

W programie zostały zdefiniowane dwie struktury :

W programie zostały użyte dwie struktury. Jedna odpowiada za przechowywanie danych o studencie, druga z kolei stworzona została w celu utworzenia listy podwieszanej dla każdego ze studentów. Każda ze struktur zawiera składowe elementu typu Grade lub Student, a także wskaźnik na kolejny element listy.

```
struct Student  
{  
    char* Name;  
    char* Surname;  
    int ID;  
    struct Grade *List;  
    struct Student *next;  
};  
  
struct Grade  
{  
    char* leader;  
    char *subject;  
    double grade;  
    char* data;  
    struct Grade *next;  
};
```

W programie użyte zostały makra – teksty zastępujące stałe użyte w kodzie programu:

#define MAX_FILES 100	określa ilość plików, które zostaną zapisane do tablicy przechowującej nazwy plików wprowadzonych do wiersza poleceń
#define MAX_LENGTH 100	określa długość ciągu znaków wczytywanych z pliku
#define MAX_ID 10	określa maksymalną długość numeru ID studenta

4.2. Funkcje

➤ `struct Student* find_student(int ID_number, struct Student * list);`

Funkcja ta przyjmuje jako argumenty :

- numer ID studenta typu int - `int ID_number`
- zmienna typu wskaźnikowego struct Student , przechowująca wskaźnik na głowę listy studentów - `struct Student * list`

Funkcja sprawdza czy student o danym numerze ID pojawił się już na liście. Jeśli odnajdzie na liście studenta o podanym numerze ID zwraca wskaźnik na niego, a jeśli nie zwraca NULL.

➤ `struct Grade* add_grade(char* leadery, char* subjecty, double gradey, char* datay);`

Funkcja ta przyjmuje jako parametry:

- zmienna leadery typu char* , przechowująca dane prowadzącego - `char* leadery`
- zmienna subjecty typu char* , przechowująca nazwę przedmiotu - `char* subjecty`
- zmienną gradey typu double, przechowującą ocenę studenta - `double gradey`
- zmienna datay typu char*, przechowująca datę wystawienia oceny - `char* datay`

Funkcja alokuje pamięć na obiekt typu struct Grade i na kolejne elementy tego obiektu i kopiuje argumenty przekazane do funkcji odpowiednio do kolejnych składowych danego elementu za pomocą funkcji strcpy. Funkcja korzysta z funkcji strlen odczytującej długość łańcucha znaków. Jest to korzystane z punktu widzenia ilości alokowanej pamięci.

➤ `void add_grade_to_list(struct Student * list, struct Grade *new_grade);`

Funkcja ta przyjmuje jako parametry:

- zmienna typu wskaźnikowego struct Student , przechowująca wskaźnik na głowę listy studentów - `struct Student * list`
- zmienna new_grade typu struct Grade*, przechowująca ocenę studenta - `struct Grade *new_grade;`

Funkcja sprawdza czy lista jest pusta. Jeśli tak to dodaje element i to on staje się początkiem listy, jeśli nie to dodaje element na początku istniejącej listy.

➤ `struct Student* add_student(char *name, char *surname, int id_number);`

Funkcja ta przyjmuje jako parametry:

- zmienną name typu char*, przechowującą imię studenta - `char *name`;

- zmienną surname typu char*, przechowującą nazwisko studenta - `char *surname`;

- zmienną id_number typu int, przechowującą numer ID studenta - `int id_number`;

Funkcja alokuje pamięć na element typu struct Student, kopiuje pobrane z pliku informacje do odpowiednich składowych danego elementu za pomocą strcpy. Długość ciągu znaków odczytywana jest za pomocą strlen.

➤ `void add_student_to_list(struct Student **list, struct Student* new_student);`

Funkcja ta przyjmuje jako argumenty :

- zmienną head typu struct Student**, przechowującą wskaźnik na głowę listy - `struct Student* list`;

- zmienną new_student typu struct Student*, przechowującą nowego studenta - `struct Student* new_student`;

Funkcja sprawdza czy lista jest pusta. Jeśli tak to dodaje element i to on staje się początkiem listy, jeśli nie to dodaje element na początku istniejącej listy.

➤ `void read_data(char* fileName, struct Student **head);`

Funkcja przyjmuje jako argumenty:

– zmienną fileName typu char*, przechowującą nazwę pliku, z którego będziemy odczytywać zawartość - `char* fileName`;

- zmienną head typu struct Student**, przechowującą wskaźnik na głowę listy - `struct Student* new_student`;

Funkcja otwiera plik do odczytu i alokuje pamięć na zmienne, które będą przechowywać informacje, które w dalszym etapie programu znajdą się w indeksie. Funkcja za pomocą funkcji fscanf odczytuje dane zawarte w pliku i zapisuje do wcześniej przygotowanych zmiennych. Wywołuje funkcję sprawdzającą czy dany student pojawił się już na liście. Kolejnym etapem jest wywoływanie funkcji dodających studenta w razie gdy tego nie ma jeszcze na liście, a następnie przypisanie oceny do odnalezionego bądź dodanego studenta. Na samym końcu funkcja zwalnia pamięć i zamyka pliku do odczytu. W razie błędów program wyświetla odpowiedni komunikat.

➤ `void create_index(struct Student *list);`

Funkcja ta przyjmuje parametr:

- zmienną list typu Student*, przechowującą głowę listy - `struct Student * list`

Sprawdza najpierw czy lista nie jest pusta. Jeśli nie, alokuje pamięć na nazwę pliku. Rzuca numer ID studenta na ciąg znaków za pomocą funkcji `sprintf` i otwiera plik do zapisu jeśli taki już istnieje lub go tworzy. Zapisuje za pomocą funkcji `fprintf` do pliku odpowiednie informacje w określonej w temacie zadania kolejności, zamyka plik i przechodzi do obsługi kolejnego studenta. Wyświetla odpowiedni komunikat jeśli operacje się nie powiedą.

➤ `void empty_memory(struct Student ** head);`

Funkcja przyjmuje argument:

- zmienną head typu Student**, przechowującą wskaźnik na głowę listy studentów - `struct Student ** head`

Funkcja ta przyjmuje jeden argument, który jest wskaźnikiem na wskaźnik na obiekt typu student- głowę listy. Funkcja tworzy wskaźnik pomocniczy aby nie utracić głowy listy studentów. Kolejno przechodzi przez listę studentów, kolejno usuwa studenta wraz z podwieszoną do niego listą ocen i przechodzi do usuwania kolejnego elementu listy.

➤ `int main(int argc, char** argv);`

Funkcja ta jako parametry przyjmuje parametry wiersza linii poleceń. Pierwszy z nich determinuje nam ilość argumentów wprowadzanych do wiersza poleceń, a drugi odnosi się już bezpośrednio do podanych przez użytkownika parametrów. W razie błędnego odczytu danych z pliku funkcja będzie wyświetlać odpowiednie komunikaty. Funkcja czyta kolejno parametry z jakimi został wywołany program. Program wychwytuje w wierszu poleceń przełącznik „-i” po którym użytkownik wpisuje nazwy plików w wierszu poleceń. Funkcja `strcmp` służy do porównywania znaków (sprawdzania w jakim momencie pojawia się przełącznik). Po sczytaniu pliku rozpoczyna operacje na nim - otwiera je i wywołuje kolejno najważniejsze funkcje. Na koniec wywołuje funkcję do tworzenia indeksów i czyszczenia pamięci.

5. Wydruk/ postać elektroniczna

W programie zostało zachowane formatowanie narzucone z góry przez program w jakim pisany był program. Komentarze do poszczególnych funkcji w programie zostały umieszczone nad każdą z funkcji zadeklarowanych w plikach nagłówkowych. Sprawozdanie w szerszym stopniu opisuje działanie programu. Aby nie zmniejszyć czytelności kodu komentarze do wątpliwych fragmentów kodu, zostały umieszczone razem z komentarzami nad deklaracjami funkcji. Deklaracja zmiennych i potrzebnych wskaźników zostały opisane w specyfikacji wewnętrznej. Nazwy zmiennych w kodzie zostały dobrane w taki sposób aby użytkownik łatwo mógł zorientować się do czego służy dana zmienna.

6. Testowanie

Program był wielokrotnie testowany na etapie tworzenia kodu by sprawdzić poprawność napisanej części fragmentu kodu. Testy docelowe odbywały się pod koniec tworzenia programu i obejmowały różne możliwe kombinacje, które mogą pojawić się w trakcie uruchamiania programu. W przypadku

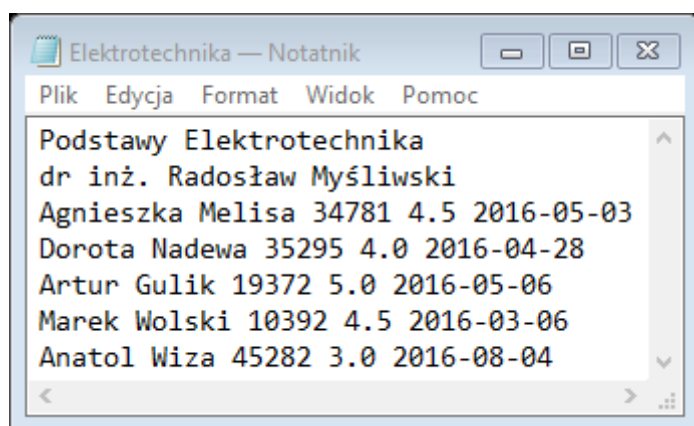
wystąpienia jakichkolwiek błędów podczas działania programu zostały wyświetlane komunikaty, które zostały już wcześniej przytoczone i opisane w poprzednich rozdziałach.

Pierwszy test został przeprowadzony dla wprowadzonych do wiersza poleceń następujących danych:






```
Dziekanat.exe -i Elektrotechnika.txt
```

Wiersz poleceń został wywołany w folderze, w którym znajduje się plik exe.

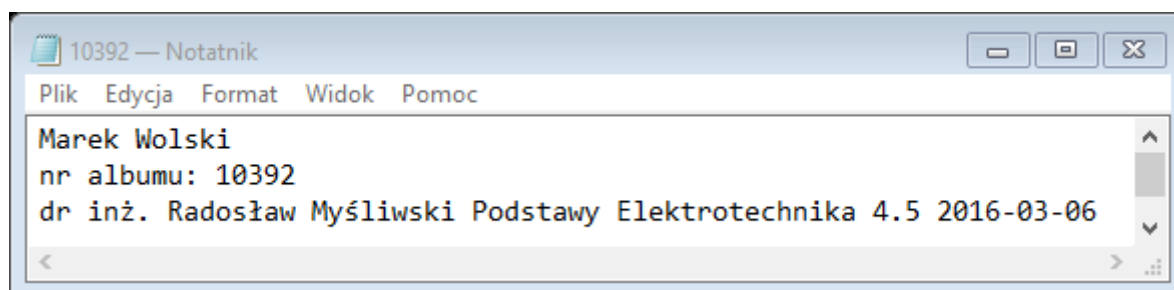
Pierwszy test został przeprowadzony dla jednego pliku. W pierwszej kolejności stworzony został plik tekstowy w formie protokołu stworzonego na wzór protokołu z zaczerpniętego z tematu programu.



Po uruchomieniu programu z wiersza poleceń w folderze, w którym znalazła się aplikacja pojawiło się pięć plików:

-  10392
-  19372
-  34781
-  35295
-  45282

Stworzone pliki są następującej postaci:

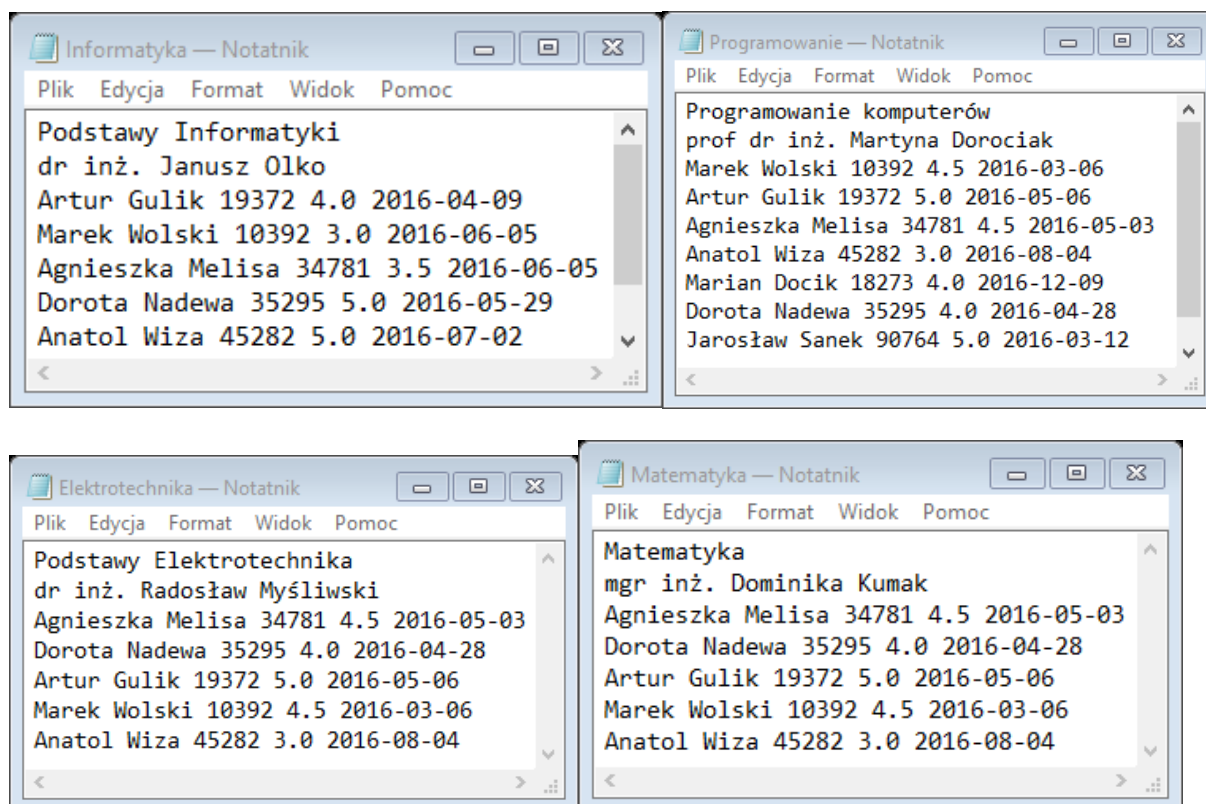


Jak widać na powyższym zdjęciu nazwa pliku jest adekwatna do numeru ID studenta.








Drugi test został przeprowadzony dla następujących danych w wierszu poleceń:

```
Dziekanat.exe -i Elektrotechnika.txt Informatyka.txt Matematyka.txt Programowanie.txt
```

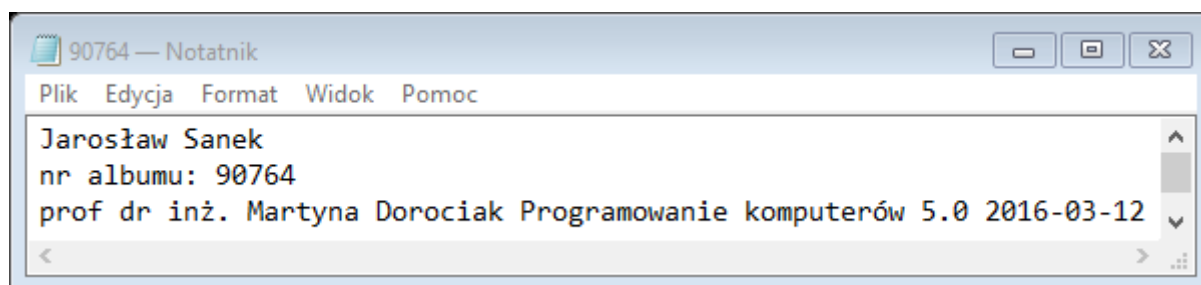

Kolejny test został przeprowadzony dla większej ilości plików. Pliki zawierające protokoły egzaminacyjne zostały tak dobrane aby sprawdzić czy program odczyta dwa pliki o takiej samej zawartości, pliki w których kolejność studentów została zachowana oraz plik ze zmienioną kolejnością studentów i różną liczbą tych studentów.

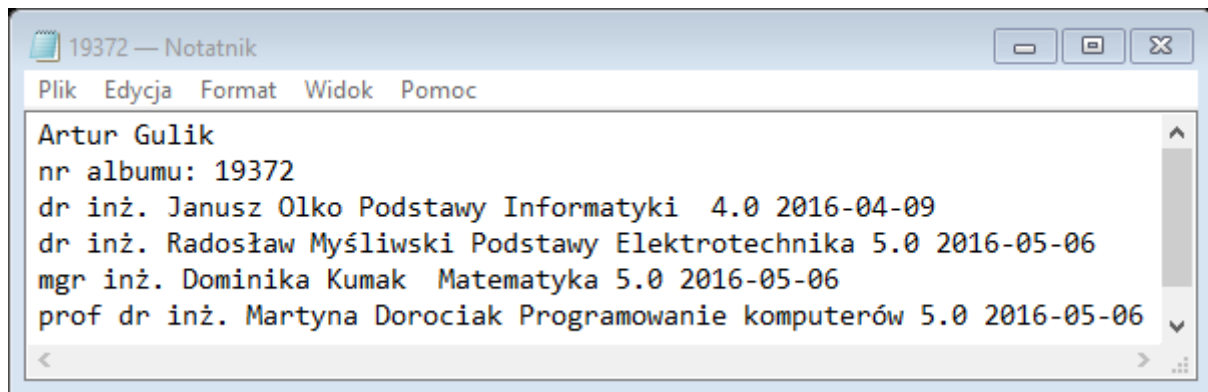


Po wykonaniu tego programu uzyskujemy:

-  10392
-  18273
-  19372
-  34781
-  35295
-  45282
-  90764

Uzyskujemy dwa typy plików wyjściowych w zależności od tego w ilu protokołach egzaminacyjnych został odnaleziony dany student. Obie postaci powstałych plików zamieszczam poniżej:





Jak widać w wyniku działania programu uzyskujemy wyniki zgodne z założeniami obranymi przy rozpoczęciu pisania programu.

7. Wnioski

Wybrana przeze struktura dynamiczna okazała się bardzo trafiona do realizacji tego typu programu. Lista podwieszana idealnie wpasowała się w temat mojego projektu. Była ona chyba najprostszą formą, za pomocą której można było zrealizować algorytm. Program wymagał znajomości obsługi parametrów wywoływanych z wiersza poleceń, operacji na ciągach znaków i dynamicznego zarządzania pamięcią. Stosowanie struktur dynamicznych bardzo dobrze sprawdza się przy obsłudze dużej ilości danych.