Politechnika Śląska w Gliwicach

Wydział Automatyki , Elektroniki i Informatyki

Podstawy Programowania Komputerów

„Indeksy”

autor	Magdalena Bajerska
prowadzący	mgr. Paweł Sadowski
rok akademicki	2015/2016
kierunek	Informatyka
rodzaj studiów	SSI
semestr	1
termin laboratorium	12-13.30 wtorek
grupa	6
sekcja	2

Sprawozdanie z realizacji projektu

1.Temat

„Indeksy”

Tematem projektu było opracowanie programu, którego zadaniem będzie tworzenie indeksów poszczególnych studentów.

Wymagania początkowe dotyczące opracowywanego programu:
- pobieranie danych z plików tekstowych
- tworzenie list na podstawie danych pobranych z plików
- generowanie indeksów studentów w osobnych plikach wyjściowych

Program pobiera z nazw plików wejściowych nazwy przedmiotów dla których zostały wprowadzone dane w plikach.

Program będzie pobierał z pliku :
- imiona studentów
-nazwiska studentów
-ocenę jaką uzyskał student z przedmiotu określonego w nazwie pliku tekstowego

Program na podstawie powyższych danych będzie tworzył indeksy poszczególnych studentów.

2. Instrukcja

Program można uruchomić za pomocą linii poleceń ,wpisując odpowiednio określone parametry.

W wierszu należy wpisać :

(nazwa programu).exe /o (nazwa pliku wejściowego 1).txt (nazwa pliku wejściowego 2).txt \

np.:

indeksy.exe /o elektrotechnika.txt informatyka.txt \

W przypadku gdy podane pliki zostaną wpisane błędnie program wyświetli komunikat o błędnym podaniu argumentów i przykład prawidłowego ich wpisania.

Jeśli podane dane będą błędne program także zasygnalizuje to komunikatem o błędnym formacie danych wejściowych.

W pliku wejściowym w każdej z linijek powinny pojawić się kolejno :

- imię studenta

- nazwisko studenta

- ocena studenta z określonego w nazwie pliku przedmiotu

Nazwa pliku tekstowego powinna być równocześnie nazwą przedmiotu, którego dotyczą dane umieszczone w pliku.

W plikach wyjściowych zostaną wygenerowane indeksy poszczególnych studentów.

Nazwa wyjściowego pliku tekstowego tworzona będzie dla każdego studenta indywidualnie i będzie miała postać :

ImięNazwisko.txt

W pliku wyjściowym pojawi się wykaz ocen z danych przedmiotów dla danego studenta.

3.Specyfikacja wewnętrzna

W programie zostały użyte dwie struktury:

```
struct Grade
{
	string Subject;
	double grade;
	Grade * next;
};

struct Student
{
	string Name;
	string Surname;
	Grade * List;
	Student * next;
};
```

Pierwsza ze struktur (struct Grade – struktura Ocena) zawiera 3 elementy. Pierwszym z nich jest zmienna Subject (Przedmiot) typu string przechowująca nazwę przedmiotu. Drugi to zmienna grade (ocena) typu double przechowująca ocenę z danego pzedmiotu. Ostatnim elementem jest wskaźnik na kolejny element listy przedmiotów i ocen.

Druga struktura (struct Student – struktura Student) zawiera 4 elementy. Pierwsze dwa to zmienne Name (imię studenta) i Surname (nazwisko studenta) typu string. Zmienne te przechowują imię i nazwisko danego studenta. Następnym elementem jest List (lista) wskaźnik na element typu Grade (wykaz ocen studenta). Ostatni element struktury to wskaźnik na następny element listy (wskaźnik na następnego studenta.

Główna funkcja programu odpowiedzialna jest za wykonywanie operacji na plikach. Zmienna counter została stworzona w celu zliczania ilości plików z których program ma czytać dane. Funkcja wyświetla komunikaty o błędnym podaniu argumentów w wierszu poleceń. Funkcja wywołuje inne funkcje np.: read_data, create_index, empty_memory.

```
int main(int argc, char *argv[])
{
	string files[MAX_FILES];
if (argc == 1)
	cout << "Blednie podane argumenty." << endl;
	cout << "Przykład poprawnego podania argumentów: ” << endl;
	cout << " \" program.exe /o a.txt b.txt \" ” << endl;
	}
	else
	{
		if (strcmp(argv[1], "/o")==0)
		{
			int counter = 0;
	while (--argc > 1)
			{
				files[counter] = argv[argc];
		counter++;
			}
		}
		else
			cout << "Bledny format danych wejsciowych." << endl;
	}
	Student * head = nullptr;
	int counter = 0;
	while (files[counter] != "")
	{
		if (!(files[counter].find(".txt") == string::npos))
	read_data(files[counter], &head;
		counter++;
	}
	create_index(head);
	empty_memory(&head);
	return 0;
}
```

Funkcja add_student typu void ma za zadanie tworzenie listy studentów. Każdy element posiada dwie dane – dotyczącą imienia studenta (name) i nazwiska studenta (surname). W funkcji użyto dynamicznej alokacji pamięci, wskaźników pomocniczych tmp (temporary-tymczasowy) i it. Wskaźnik na next (następny) przekierowuje do następnego elementu listy (następnego studenta).

```
void add_student(Student ** list, string name, string surname)
{
	Student * tmp = new Student;
	tmp->Name = name;
	tmp->Surname = surname;
	tmp->List = nullptr;
	tmp->next = nullptr;
	if ((*list) == nullptr)
	{
		(*list) = tmp;
	}
	else
	{
		Student * it = (*list);
while (it->next != nullptr)
		{
			it = it->next;
		}
		it->next = tmp;
	}
}
```

Funkcja check_student typu bool ma za zadanie sprawdzanie czy lista studentów (list) nie jest pusta oraz czy odczytane z pliku imię (name) i nazwisko (surname) pojawiło się już wcześniej na liście studentów (list) .

```
bool check_student(Student * list, string name, string surname)
{
	while (list != nullptr)
	{
		if (list->Name == name && list->Surname == surname)
			return true;
		list = list->next;
	}
	return false;
}
```

Funkcja add_grade typu void ma za zadanie tworzenie wykazu ocen. Każdy element posiada dwie dane – dotyczącą oceny (grade) i przedmiotu (subject). W funkcji użyto dynamicznej alokacji pamięci oraz wskaźnika pomocniczego tmp (temporary-tymczasowy). Wskaźnik na next (następny) przekierowuje do następnego elementu listy (następnego studenta).

```
void add_grade(Student * student, double grade, string subject)
{
	Grade * tmp = new Grade;
	tmp->grade = grade;

	tmp->Subject = subject;
	tmp->next = nullptr;
	if (student->List == nullptr)

	{
		student->List = tmp;
	}
	else
	{
		tmp->next = student->List;
		student->List = tmp;
	}
}
```

Funkcja create_index typu void ma za zadanie tworzenie indeksów. Tworzy nazwy indeksów studentów na podstawie ich imienia pobieranego z listy (list->Name) + nazwiska pobieranego z listy (list->Surname) i dodanie końcówki „.txt”. Zajmuje się także obsługą pliku wyjściowego (otwieranie i sprawdzanie poprawności operacji). Zapisuje do plików wyjściowych wykaz ocen studenta z każdego przedmiotu (subject, grade) pobierane z listy. Następnie zamyka plik wyjściowy dla danego studenta i przechodzi do następnego elementu .

```
void create_index(Student * list)
{
	while (list != nullptr)
	{
		string fileName = list->Name + list->Surname + ".txt";
		ofstream stream;
		stream.open(fileName);
		if (stream.is_open())
		{
			Grade * tmp = list->List;
			while (tmp != nullptr)
			{
			stream << tmp->Subject << ": " << tmp->grade << endl;
				tmp = tmp->next;
}
			stream.close();
		}
		list = list->next;

	}
}
```

Funkcja read_data typu void ma za zadanie odczytywanie danych z plików wejściowych. Otwiera pliki wejściowe i sprawdza czy operacje zostały przeprowadzone prawidłowo. Sczytuje z pliku kolejno imię (name), nazwisko (surname), ocenę (grade). Wywołuje funkcje sprawdzające czy imię i nazwisko danego studenta nie pojawiło się już wcześniej (check_student). Następnie wywołuje funkcję dodającą studenta do list (add_student) lub najpierw odszukuje wybranego studenta na liście i dopiero wtedy wywołuje funkcję dodającą studenta do listy (add_student). Na końcu zamyka pliki wejściowe (stream).

```
void read_data(string fileName, Student ** head)
{
	string subject = fileName.substr(0, fileName.find(".txt"));
	ifstream stream;
	stream.open(fileName);
	if (stream.is_open())
	{
		while (stream.good())
		{
			string name;
			stream >> name;
			string surname;
			stream >> surname;
			double grade;
			stream >> grade;
			if (!stream.fail())
		  {
				if (!check_student((*head), name, surname))
					add_student(head, name, surname);
			Student * present_student = find_student(name, surname,
(*head));
				add_grade(present_student, grade, subject);
			}
		}
		stream.close();
	}
}
```

Funkcja find_student wskazuje na obiekt typu Student odpowiedzialna jest za odszukiwanie studenta. W pierwszej kolejności sprawdza czy głowa listy (head) jest różna od 0. Następnie sprawdza czy imię (name), nazwisko (surname) pokrywa się z tym na co wskazuje głowa listy (head). Jeśli powyższy warunek jest spełniony zwraca wartość głowy i zaczyna wskazywać na następny element (head->next). W przypadku gdy głowa listy nie wskazuje na nic, zwraca wartość 0.

```
Student * find_student(string name, string surname, Student * head)
{
	while (head != nullptr)
{
		if (head->Name == name && head->Surname == surname)
			return head;
		head = head->next;
	}
	return nullptr;
}
```

Funkcja empty_memory jest funkcją typu void ,która ma za zadanie zwalnianie pamięci. Tworzy wskaźnik pomocniczy wskazujący na wskaźnik na głowę listy (*head). Sprawdza czy wskaźnik ten nie wskazuje na 0. Jeśli tak to przypisuje wskaźnikowi tmp (temporary – tymczasowy) wartość na którą wskazuje głowa listy (head). Następnie sprawdza czy wskaźnik tmp wskazujący na List (lista) wskazuje na jakiś element. Nastepnie tworzy wskaźnik pomocniczy tmp2 (temporary2 = tymczasowy2) i ustawia wskaźnik na ten sam element co tmp. Potem przechodzi do następnego elementu listy i wskaźnik pomocniczy tmp2 zostaje usunięty. W kolejnym kroku funkcja przechodzi na kolejny element listy i usuwany jest wskaźnik tmp.

```
void empty_memory(Student ** head)
{
	Student * tmp = (*head);
	while ((*head) != nullptr)
	{
		tmp = (*head);
		while (tmp->List != nullptr)
		{
			Grade * tmp2 = tmp->List;
			tmp->List = tmp->List->next;
			delete tmp2;
		}
		(*head) = (*head)->next;
		delete tmp;
	}
}
```

4.Testowanie

Komunikaty wyświetlane :

- W przypadku gdy błędnie podano argumenty konsolowe

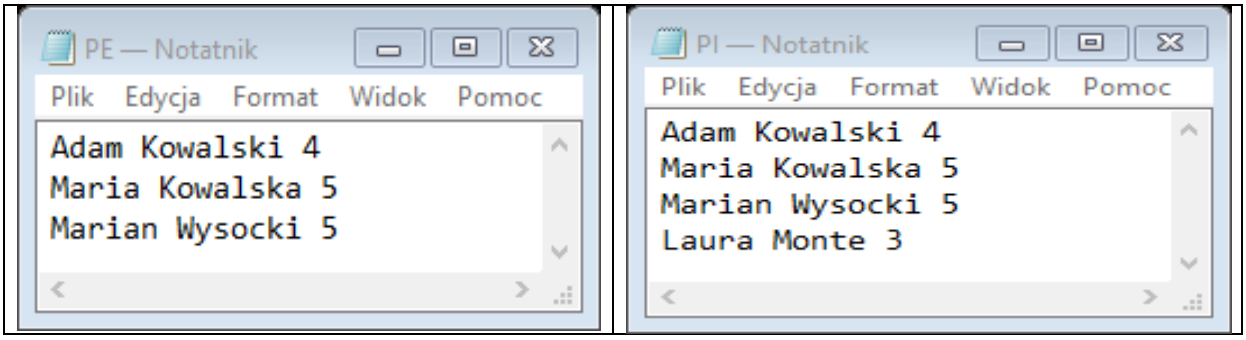
```
Blednie podane argumenty.
Przyklad poprawnego podania argumentow :
"indeksy.exe /o PE.txt PI.txt"
```

- W przypadku gdy format danych wejściowych jest błędny

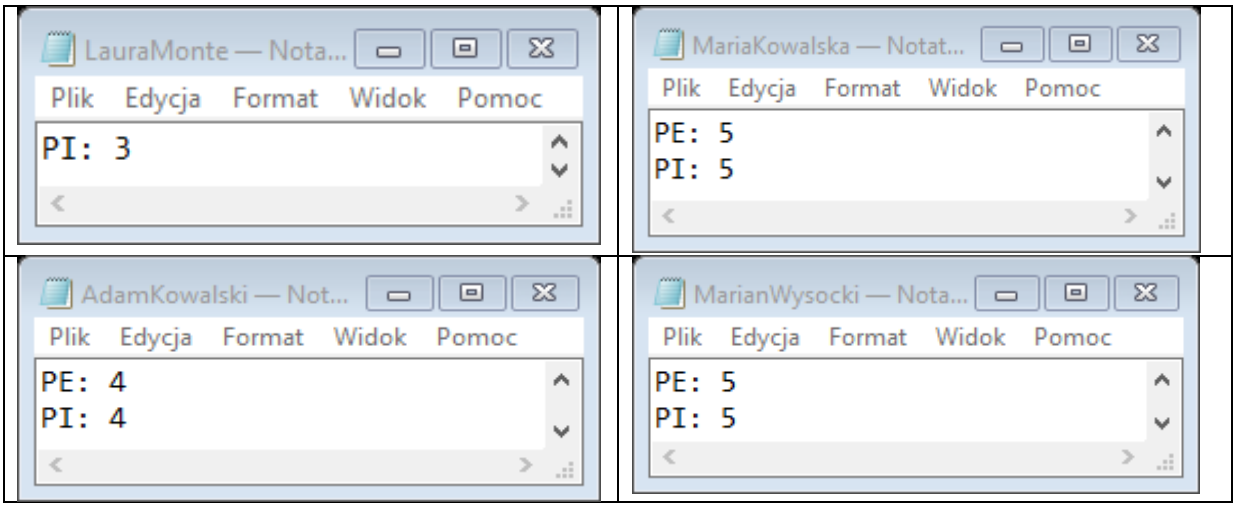
```
Bledny format danych wejsciowych.
```

- W przypadku gdy pliki zostały odczytane prawidłowo na podstawie plików i zawartych w nich danych tworzy indeksy studentów

  Przykładowe testowe pliki wejściowe :



  Pliki wyjściowe do podanych wejściowych plików testowych:



Wnioski :

Powyżej opisany program realizuje założenia obrane na początku , wyświetla prawidłowe komunikaty. Wyniki uzyskane w wyniku realizacji programu są poprawne. W programie można wyróżnić pare elementów które wymagały poświecenia szczególnej uwagi :
- odczytywanie paru argumentów (plików wejściowych) z wiersza poleceń,
- odczytywanie z nazw plików nazw przedmiotów, które były potrzebne do wygenerowania indeksów,
- operacje na listach (lista podwieszana),
- tworzenie plików wyjściowych z nazwą uzyskaną poprzez połączenie imienia i nazwiska studenta.