Sprawozdanie

Spis treści

1. [Treść zadania](#Treść)
2. [Analiza problemu](#Analiza)

1. [Specyfikacja zewnętrzna](#Zewnętrzna)
2. [Specyfikacja wewnętrzna](#Wewnętrzna)
3. [Opis testowania](#Opis_testowania)
4. Treść zadania

Implementacja własnej struktury danych: lista jednokierunkowa

Zaimplementować własną strukturę danych, w moim projekcie to lista jednokierunkowa, bez użycia kontenerów STL, z użyciem inteligentnych wskaźników, stworzenie konstruktorów kopiujących/przenoszących, konstruktora bezargumentowego, operatorów przypisania/przeniesienia, pełnienia funkcjonału dodawania elementów do kontenera, wyszukiwanie elementu, sortowanie zawartości. Zrealizować, przedstawić i przetestować to na przykładowym pliku z dużą liczbą danych. Projekt ma być podzielony na pliki .h i .cpp.

1. Analiza problemu

Dla realizacji danego projektu, listy jednokierunkowej, po przemyślaniu, najlepsze będzie wykorzystać implementację struktury dynamicznej, bo z założenia obowiązkowym jest korzystanie z inteligentnych wskaźników, co już oznacza dynamicznie alokowanie obiektów za pomocą np. „std::shared\_ptr<Struktura>”, który w dalszym ciągu umożliwia przechowywanie wskaźników na kolejne elementy, co spełnia główną ideę struktury listy jednokierunkowej. Razem z tym brak ograniczeń z góry o rozmiarze, czyli można dodawać, usuwać elementy w danej liście.

Do wyborów algorytmów, a dokładnie która metoda, najważniejszym faktem gra wielkość danych wskazana w tym projekcie. Ma być to plik z dużą liczbą danych. Ten kryterium już odrzuca takie metody sortowanie jak: sortowanie bąbelkowe o złożoności czasowej O(n2), sortowanie przez wstawiania o złożoności czasowej O(n2), sortowanie przez wybieranie o złożoności czasowej O(n2). Najbardziej zoptymalizowane metody pod kątem wydajności, spośród których można już wybrać to sortowanie szybkie o złożoności czasowej O(n log n), sortowanie przez scalanie o złożoności czasowej O(n log n), sortownie kubełkowe o złożoności czasowej O(n), sortowanie przez zliczanie o złożoności czasowej O(n+k). Ostatecznie do zaimplementowanie została wybrana metoda szybkiego sortowania na podstawie jego działania: wybór pivot’a, według którego będzie dzielony zbiór danych, podział zbiory danych, rekurencyjne sortowanie lewej i prawej części i zakończenie rekurencyjne, czyli gdy zbiór danych składa się z jednego elementu (lub pustego zbioru), ponieważ jednoelementowy zbiór jest zawsze posortowany.

Zadanie, można podzielić na takie części: odczyt danych z pliku i umieszczenie ich w pamięci operacyjnej, dodanie, wyszukiwanie, usunięcie elementu, wyświetlenie listy, sortowanie listy, zamiana miejscami elementów, zapis wyników do pliku.

* 1. Przedstawienie tematyki, na której się opiera realizacja projektu

Dane, które umieszczone w pliku listaStutentow.txt, mieszczą w sobie imię, nazwisko, pięć przedmiotów i ich oceny. Te dane, będą odczytywane, sortowane, usuwane, wyświetlane, zamieniane, zapisywane oraz przez użytkownika po wprowadzanie każdego po kolei dodawane. Plik będzie już stworzony z powodu tego, że ma zawierać dużą ilość danych, więc użytkownik nie musi go stwarzać ręcznie, choć może, po wielokrotnym wprowadzaniu każdego studenta.

1. Specyfikacja zewnętrzna

3.1. Sposób uruchomienia programu

Program pracuje w środowisku Windows 10. Aby go uruchomić wykorzystuję Microsoft Visual Studio 2022, standard języka ISO C++ 14, typ konfiguracji aplikacja (.exe), plik tekstowy z wejściowymi danymi jest już stworzony pod nazwą „listaStudentow.txt”, do której będą również wpisywane wyniki programu.

3.2. Format danych wejściowych

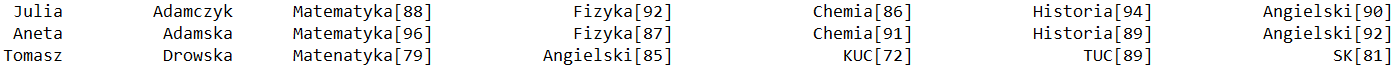
Plik z danymi wejściowymi jest plikiem tekstowym zapisanym w formacie ASCII, czyli jeden znak zajmuje jeden bajt. Danymi do sortowania są wyrazy wykorzystujące 10 znaków cyfr oraz 24 znaki liter, co oznacza że wyrazy nie mogą zawierać "polskich" znaków. Dane one jako informacja studentów w postaci:

Julia Adamczyk Matematyka[88] Fizyka[92] Chemia[86] Historia[94] Angielski[90]

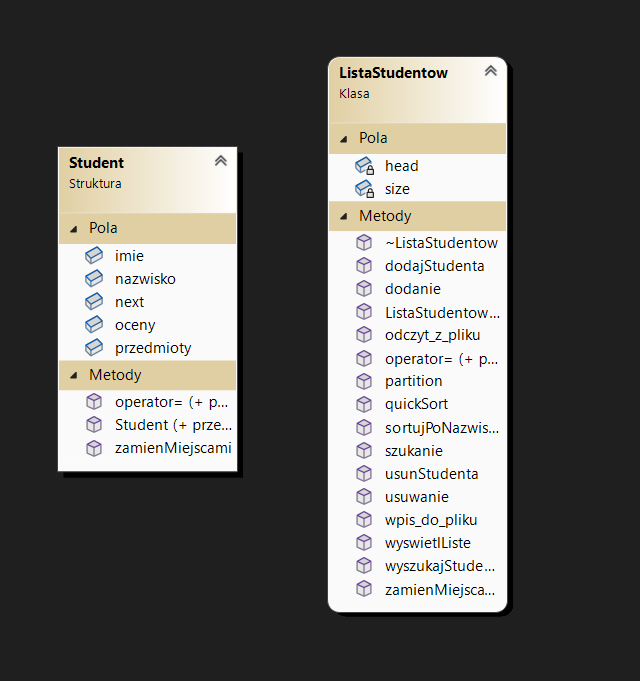
Gdzie odpowiednio (imie, nazwisko, przedmiot[ocena], przedmiot[ocena], przedmiot[ocena], przedmiot[ocena], przedmiot[ocena]). Sortowanie się odbywa według nazwisk (alfabetycznie). Powyższy zapis się traktuje jako jedna linia i informacja o kolejnym studencie ma się podawać dokładnie w taki sam sposób.

3.3. Format danych wyjściowych

Dane wyjściowe zostają zapisane w pliku tekstowym w formacie ASCII. Posortowane, dodane, zamienione miejscami, wyrazy zostają umieszczone w pliku tak, aby każdy znajdował się w oddzielnej linii oraz jak posortowane alfabetycznie, jeżeli tej operacji się użyło np.:



1. Specyfikacja wewnętrzna



W kodzie użyto kilku technik programowania obiektowego. Takie jak:

Klasa i struktura: zdefiniowano ‘ListaStudentow’ i ‘Student’. Klasy i struktury są podstawowym budulcem programowania obiektowego, pozwalają na enkapsulację danych i metod.

Konstruktory: w klasie Student zdefiniowano różne konstruktory, w tym konstruktor domyślny, konstruktor bezargumentowy, konstruktor z argumentami, konstruktor kopiujący, konstruktor przenoszący.

Metody Klasy: w klasie ListaStudentow zaimplementowano różne metody, takie jak dodajStudenta, usunStudenta, wyszukajStudenta, wyswietlListe, zamienMiejscami\_Studentow, odczyt\_z\_pliku, wpis\_do\_pliku, itp. Metody te operują na obiektach klasy i umożliwiają manipulację danymi.

Wskaźniki do Obiektów: w klasie ListaStudentow użyto wskaźników do zarządzania listą obiektów Student.

Dostęp do pól poprzez modyfikatory dostępu: w klasie ListaStudentow pola takie jak head i size są prywatne, a dostęp do nich odbywa się poprzez publiczne metody.

Przenoszenie zasobów (move semantics): w klasie Student zaimplementowano konstruktor przenoszący oraz operator przypisania przenoszącego, co pozwala na efektywne przenoszenie zasobów między obiektami.

Użycie przestrzeni nazw (namespace): kod jest umieszczony w przestrzeni nazw ap, co pomaga w organizacji i unikaniu konfliktów nazw.

* 1. Załadowanie danych

//Metoda do odczytu z pliku

void odczyt\_z\_pliku(const std::string& plik) {

// Strumień do odczytu z pliku

std::ifstream Odczyt(plik);

// Sprawdź, czy plik został otwarty poprawnie

if (!Odczyt.is\_open()) {

std::cerr << "Nie mozna otworzyc pliku do odczytu: " << plik << std::endl;

return;

}

std::cerr << "Udalo sie otworzyc pliku : " << plik << std::endl;

// Odczytaj zawartość pliku linia po linii

std::string linia;

while (std::getline(Odczyt, linia)) {

std::istringstream ss(linia);

std::string imie, nazwisko, przedmiot, ocena;

int oceny[5];

std::string przedmioty[5];

// Odczytaj imię i nazwisko

ss >> imie >> nazwisko;

int i = 0;

// Odczytaj przedmioty i oceny

while (ss >> przedmiot) {

size\_t pozycjaOtwarcia = przedmiot.find('[');

size\_t pozycjaZamkniecia = przedmiot.find(']');

if (pozycjaOtwarcia != std::string::npos && pozycjaZamkniecia != std::string::npos) {

std::string nazwaPrzedmiotu = przedmiot.substr(0, pozycjaOtwarcia);

int ocenaPrzedmiotu = std::stoi(przedmiot.substr(pozycjaOtwarcia + 1, pozycjaZamkniecia));

przedmioty[i] = nazwaPrzedmiotu;

oceny[i++] = ocenaPrzedmiotu;

}

}

dodajStudenta(Student(imie, nazwisko, oceny, przedmioty));

}

// Zamykamy plik po zakończeniu operacji odczytu

Odczyt.close();

}

Zakładamy, że plik tekstowy z danymi do wykonania nad nimi różne operacji jest zbudowany tak, że w każdej linijce tekstu znajdują się imię, nazwisko, przedmiot i ocena studentów w takiej postaci, np.:

Tomasz Nowicki Matematyka[79] Angielski[85] KUC[72] TUC[89] SK[81]

Ponieważ liczba wyrazów, które mogą wystąpić w pliku wejściowym nie jest z góry narzucona, należy je przechować w dynamicznie rezerwowanym obszarze pamięci operacyjnej. Odbywa się to przy użyciu inteligentnych wskaźników ‘std::shared\_ptr’. Dla każdego studenta utworzony jest obiekt typu Student, a jego wskaźnik jest przechowywany w liście jednokierunkowej, każdy element listy będzie wskazywał na jeden wyraz (dzięki zastosowaniu wskaźnika na łańcuch znakowy unikamy ograniczenia długości np. sortowanych danych) oraz na element następny, razem z tym zwiększa rozmiar listy, realizuje to metoda ‘dodajStudenta()’.

Dzięki temu podejściu, program samodzielnie zarządza pamięcią przy użyciu inteligentnych wskaźników, eliminując potrzebę ręcznego zarządzania alokacją i dealokacją pamięci.

* 1. Dodanie

// Metoda dodająca nowego studenta na początku listy

void dodajStudenta(const Student& student) {

std::shared\_ptr<Student> nowyStudent = std::make\_shared<Student>(student);

nowyStudent->next = head;

head = nowyStudent;

size++;

}

void dodanie() {

std::string imie, nazwisko;

int oceny[5];

std::string przedmioty[5];

std::cout << "Podaj imie studenta: ";

std::cin >> imie;

std::cout << "Podaj nazwisko studenta: ";

std::cin >> nazwisko;

std::cout << "Podaj przedmioty i ich oceny (np. Matematyka 90): ";

for (int i = 0; i < 5; ++i) {

std::cout << "Podaj przedmiot " << i + 1 << " ";

std::cin >> przedmioty[i];

while (true) {

if (std::cin >> oceny[i]) {

break;

}

std::cout << "\nWprowadzono nie koreknie dane. Prosze jeszcze raz podac ocene:\n";

// Wyczyść błąd wejścia i zignoruj nieprawidłowe znaki

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

}

dodajStudenta(Student(imie, nazwisko, oceny, przedmioty));

std::cout << "Student dodany.\n";

}

4.2.1. Metoda dodajStudenta():

Funkcja przyjmuje referencję do obiektu klasy Student, który ma zostać dodany do listy. Tworzony jest nowy obiekt ‘std::shared\_ptr<Student>’ o nazwie ‘nowyStudent’, który inicjalizowany jest przy użyciu funkcji ‘std::make\_shared<Student>(student)’. Ta funkcja alokuje dynamicznie pamięć dla obiektu klasy Student i zwraca inteligentny wskaźnik ‘std::shared\_ptr’ do tego obiektu. Wskaźnik ‘nowyStudent->next’ (człon ‘next’ w strukturze Student) ustawiany jest na obecny wskaźnik głowy listy (‘head’), co oznacza, że nowy student będzie wskazywał na poprzedniego pierwszego studenta na liście. Główny wskaźnik ‘head’ zostaje zaktualizowany, aby wskazywał na nowego studenta, dzięki czemu ten staje się nowym pierwszym elementem listy. Zmienna ‘size’ (przechowująca rozmiar listy) zwiększa się o 1.

4.2.2. Metoda dodanie:

Pobierane są dane od użytkownika, takie jak imię, nazwisko, przedmioty i oceny. Dane te są przechowywane w lokalnych zmiennych. Tworzony jest obiekt klasy Student o nazwie ‘Student(imie, nazwisko, oceny, przedmioty)’, który jest bezpośrednio przekazywany do metody ‘dodajStudenta()’. Metoda ‘dodajStudenta()’ dynamicznie alokuje pamięć dla nowego studenta i dodaje go na początek listy. Po dodaniu studenta, program informuje użytkownika, że student został dodany.

* 1. Wyszukiwanie

// Metoda wyszukująca studenta o podanym nazwisku std::shared\_ptr<Student> wyszukajStudenta(const std::string& nazwisko) const {

std::shared\_ptr<Student> current = head;

while (current != nullptr) {

if (current->nazwisko == nazwisko) {

return current;

}

current = current->next;

}

return nullptr; // Nie znaleziono studenta

}

void szukanie() const {

std::string nazwisko;

std::cout << "Podaj nazwisko studenta do wyszukania: ";

std::cin >> nazwisko;

std::shared\_ptr<Student> szukanyStudent = wyszukajStudenta(nazwisko);

if (szukanyStudent) {

std::cout << "\nZnaleziono studenta o nazwisku " << nazwisko << ":\n";

std::cout << std::setw(11) << szukanyStudent->imie << std::setw(11) << szukanyStudent->nazwisko;

for (int i = 0; i < 5; ++i) {

std::cout << std::setw(11) << szukanyStudent->przedmioty[i] << "[" << szukanyStudent->oceny[i] << "]";

}

std::cout << std::endl;

}

else {

std::cout << "\nNie znaleziono studenta o nazwisku " << nazwisko << std::endl;

}

}

4.3.1. Metoda wyszukajStudenta():

Funkcja przyjmuje jako argument ‘const’ referencję do ‘std::string’, reprezentującą nazwisko studenta, którego szukamy. Utworzony jest wskaźnik ‘current’ i zainicjowany jako wskaźnik na głowę listy (‘head’). Następnie program przechodzi przez listę w poszukiwaniu studenta o podanym nazwisku. Jeśli nazwisko studenta jest zgodne z podanym, funkcja zwraca wskaźnik ‘current’ wskazujący na znalezionego studenta. Jeśli student o podanym nazwisku nie zostanie znaleziony, funkcja zwraca ‘nullptr’.

4.3.2. Metoda szukanie:

Pobierane jest nazwisko studenta od użytkownika za pomocą ‘std::cin’. Wywoływana jest metoda ‘wyszukajStudenta()’ z podanym nazwiskiem, a wynik przypisywany jest do wskaźnika ‘szukanyStudent()’. Program następnie sprawdza, czy ‘szukanyStudent()’ nie jest równy ‘nullptr’. Jeśli student o podanym nazwisku zostanie znaleziony, wypisywane są informacje o tym studencie, takie jak imię, nazwisko, przedmioty i oceny. W przeciwnym razie program informuje użytkownika, że nie znaleziono studenta o podanym nazwisku.

* 1. Usuwanie

4.4.1. Metoda usunStudenta()

// Metoda usuwająca studenta o podanym nazwisku z listy

std::shared\_ptr<Student> usunStudenta(const std::string& nazwisko) {

std::shared\_ptr<Student> current = head;

std::shared\_ptr<Student> prev = nullptr;

while (current != nullptr && current->nazwisko != nazwisko) {

prev = current;

current = current->next;

}

if (current != nullptr) {

if (prev != nullptr) {

prev->next = current->next;

}

else {

head = current->next;

}

current->next = nullptr;

}

size--;

return current;

}

void usuwanie() {

std::string nazwisko;

std::cout << "Podaj nazwisko studenta do usuniecia: ";

std::cin >> nazwisko;

std::shared\_ptr<Student> usunietyStudent = usunStudenta(nazwisko);

if (usunietyStudent) {

std::cout << "\nUsunieto studenta o nazwisku " << nazwisko << ":\n";

std::cout << "Imie: " << usunietyStudent->imie << ", Nazwisko: " << usunietyStudent->nazwisko;

std::cout << ", Przedmioty[Oceny]: ";

for (int i = 0; i < 5; ++i) {

std::cout << " " << usunietyStudent->przedmioty[i] << "[" << usunietyStudent->oceny[i] << "]";

}

std::cout << std::endl;

}

else {

std::cout << "\nNie znaleziono studenta o nazwisku " << nazwisko << std::endl;

}

}

Funkcja przyjmuje jako argument ‘const’ referencję do ‘std::string’, reprezentującą nazwisko studenta, który ma zostać usunięty. Utworzony jest wskaźnik ‘current’ i zainicjowany jako wskaźnik na głowę listy (‘head’), oraz wskaźnik ‘prev’ zainicjowany jako ‘nullptr’. Następnie program przechodzi przez listę w poszukiwaniu studenta o podanym nazwisku. Jeśli student o podanym nazwisku zostanie znaleziony, to wskaźnik ‘prev’ wskazuje na poprzednika aktualnego studenta (‘current’). Aktualny student (‘current’) jest usuwany z listy przez ustawienie odpowiedniego wskaźnika ‘next’ poprzednika (‘prev’). Jeśli usuwany student był na początku listy, to głowa listy (‘head’) jest aktualizowana. Wskaźnik ‘next’ usuwanego studenta ustawiany jest na ‘nullptr’, aby nie wskazywał na żaden inny element. Zmniejszana jest wartość zmiennej ‘size’, która przechowuje ilość elementów w liście. Jeśli student o podanym nazwisku nie zostanie znaleziony, funkcja zwraca ‘nullptr’.

4.4.2. Metoda usuwanie:

Pobierane jest nazwisko studenta do usunięcia od użytkownika za pomocą ‘std::cin’. Wywoływana jest metoda ‘usunStudenta()’ z podanym nazwiskiem, a wynik przypisywany jest do wskaźnika ‘usunietyStudent()’. Program następnie sprawdza, czy ‘usunietyStudent()’ nie jest równy ‘nullptr’. Jeśli student o podanym nazwisku zostanie znaleziony, wypisywane są informacje o usuniętym studencie, takie jak imię, nazwisko, przedmioty i oceny. W przeciwnym razie program informuje użytkownika, że nie znaleziono studenta o podanym nazwisku.

* 1. Wyświetlanie

// Metoda wyświetlająca zawartość listy

void wyswietlListe() const {

std::shared\_ptr<Student> current = head;

while (current != nullptr) {

std::cout << std::setw(11) << current->imie << "\t" << std::setw(11) << current->nazwisko << "\t";

// Wyświetl oceny i przedmi

for (int i = 0; i < 5; ++i) {

std::cout << std::setw(11) << current->przedmioty[i] << "[" << current->oceny[i] << "]";

}

std::cout << std::endl;

current = current->next;

}

}

4.5.1. Inicjalizacja wskaźnika ‘current’ jako wskaźnik na głowę listy (‘head’). Pętla ‘while’ przechodzi przez listę, dopóki wskaźnik ‘current’ nie osiągnie wartości ‘nullptr’. Wewnątrz pętli wyświetlane są imię i nazwisko bieżącego studenta, używając ‘std::setw’ do odpowiedniego wyrównania kolumn. Następnie w pętli for wyświetlane są oceny i przedmioty studenta. Każdy przedmiot i ocena są wyświetlane z odpowiednim wyrównaniem kolumn. Wskaźnik current jest przesuwany na następnego studenta przez przypisanie ‘current = current->next’.

* 1. Sortowanie

// Sortowanie quick\_sort

int partition(std::shared\_ptr<Student> arr[], int start, int end) {

std::string pivot = arr[start]->nazwisko;

int count = 0;

for (int i = start + 1; i <= end; i++) {

if (arr[i]->nazwisko <= pivot) {

count++;

}

}

// Nadanie elementowi pivot właściwej pozycji

int pivotIndex = start + count;

std::swap(arr[pivotIndex], arr[start]);

// Sortowanie lewej i prawej części elementu pivot

int i = start, j = end;

while (i < pivotIndex && j > pivotIndex) {

while (arr[i]->nazwisko <= pivot) {

i++;

}

while (arr[j]->nazwisko > pivot) {

j--;

}

if (i < pivotIndex && j > pivotIndex) {

std::swap(arr[i], arr[j]);

}

}

return pivotIndex;

}

void quickSort(std::shared\_ptr<Student> arr[], int start, int end) {

// przypadek podstawowy

if (start >= end) {

return;

}

//podział tablicy

int podzial = partition(arr, start, end);

//sortowanie lewej strony

quickSort(arr, start, podzial - 1);

//sortowanie prawej strony

quickSort(arr, podzial + 1, end);

}

// Wywołanie quickSort

void sortujPoNazwisku() {

if (head != nullptr) {

std::shared\_ptr<Student>\* arr = new std::shared\_ptr<Student>[size];

int i = 0;

// Przepisanie wskaźników z listy do tablicy

std::shared\_ptr<Student> current = head;

while (current != nullptr && i < size) {

arr[i++] = current;

current = current->next;

}

//wywołanie sortowania

quickSort(arr, 0, i - 1);

// Przepisz posortowane elementy z powrotem do listy

head = arr[0];

current = head;

for (int j = 1; j < i; ++j) {

current->next = arr[j];

current = current->next;}

current->next = nullptr; // Ustaw next z ostatniego elementu na nullptr

delete[] arr; // Zwolnij zaalokowaną pamięć } }

4.6.1. sortujPoNazwisku()

Algorytm sortowania QuickSort dla tablicy obiektów klasy ‘Student’. Metoda sortująca listę studentów w kolejności alfabetycznej nazwisk, korzystając z algorytmu quicksort na tablicy wskaźników. Tworzy dynamiczną tablicę wskaźników na studentów i przepisuje do niej wskaźniki z listy. Wywołuje metodę 'quicksort()' na tablicy wskaźników. Aktualizuje odpowiednio połączoną listę, aby odzwierciedlić posortowany porządek. Po posortowaniu tablicy zmienia rozmieszczenie kolejnych wskaźników połączonych węzłów listy, aby dopasować je do nowej kolejności. Zwalnia zaalokowaną pamięć.

4.6.2 partiotion()

Metoda odpowiedzialna za wyznaczanie punktu podziału w tablicy wskaźników na studentów do sortowania quicksort. Wybiera pivot, czyli nazwisko pierwszego studenta w fragmencie tablicy. Przesuwa elementy tablicy, która jest tablicą wskaźników na studentów, tak, aby elementy mniejsze lub równe pivot były po lewej stronie, a większe po prawej, sortując wskaźniki do obiektów 'Student', czyli zamieniamy ze sobą wskaźniki, a nie same obiekty. Przestawia pivot na właściwą pozycję.

4.6.3. quickSort()

Najpierw rozpatrywany przypadek podstawowy, jeśli zakres tablicy jest mniejszy lub równy 1, zakończ sortowanie. Jeśli tak owym nie jest to wywołujemy funkcję 'partition()' i otrzymujemy indeks pivota ('podzial'). Po jego uzyskaniu wywołujemy rekurencyjnie 'quickSort()' dla lewej i prawej części tablicy wokół pivota. W rezultacie funkcja 'quickSort()' wykorzystuje rekurencję do podziału tablicy na mniejsze fragmenty, sortuje każdy fragment oddzielnie, a następnie łączy je w posortowaną całość.

* 1. Zamiana miejscami studentów

void zamienMiejscami\_Studentow() { // (w klasie ListaStudentow):

std::string nazwisko1, nazwisko2;

std::cout << "Podaj nazwisko pierwszego studenta: ";

std::cin >> nazwisko1;

std::cout << "Podaj nazwisko drugiego studenta: ";

std::cin >> nazwisko2;

std::shared\_ptr<Student> student1 = wyszukajStudenta(nazwisko1);

std::shared\_ptr<Student> student2 = wyszukajStudenta(nazwisko2);

if (student1 && student2) {

// Używamy metody zamienMiejscami z klasy Student

student1->zamienMiejscami(student1, student2, head);

}

else {

std::cout << "Nie znaleziono jednego lub obu studentow o podanych nazwiskach." << std::endl;

}

}

void zamienMiejscami(std::shared\_ptr<Student> student1, std::shared\_ptr<Student> student2, std::shared\_ptr<Student> listaHead) {

// (w strukturze Student)

// Sprawdź, czy oba wskaźniki są różne od nullptr

if (student1 && student2) {

// Szukamy poprzedników obu studentów

std::shared\_ptr<Student> poprzednik1 = nullptr;

std::shared\_ptr<Student> poprzednik2 = nullptr;

std::shared\_ptr<Student> obecny = listaHead;

while (obecny) {

if (obecny->next == student1) {

poprzednik1 = obecny; }

if (obecny->next == student2) {

poprzednik2 = obecny }

obecny = obecny->next;}

// Zamień miejscami tylko jeśli oba znaleziono poprzedniki

if (poprzednik1 && poprzednik2) {

// Jeśli studenty nie są obok siebie, a jedynie jeden z nich jest głową listy

if (poprzednik1->next != student2 && poprzednik2->next != student1) {

std::shared\_ptr<Student> temp = student1->next;

student1->next = student2->next;

student2->next = temp;

poprzednik1->next = student2;

poprzednik2->next = student1;}

// Jeśli studenty są obok siebie

else {

// Obsłuż zamianę miejscami głowy listy

if (poprzednik1->next == student2) {

student1->next = student2->next;

student2->next = student1;

poprzednik2->next = student1;

listaHead = student2;

}

else if (poprzednik2->next == student1) {

student2->next = student1->next;

student1->next = student2;

poprzednik1->next = student2;

listaHead = student1;}}}}}};

4.7.1. zamienMiejscami\_Studentow() w klasie ListaStudentow

Użytkownik podaje nazwisko pierwszego i drugiego studenta. Wywoływana jest metoda 'wyszukajStudenta(nazwisko)' dla obu podanych nazwisk, aby wyszukać studentów w liście. Jeśli obie operacje wyszukiwania zwróciły poprawne wskaźniki na studentów ('student1' i 'student2'), przechodzi do zamiany miejscami. Wywoływana jest metoda 'zamienMiejscami(student1, student2, head)' z klasy 'Student'. Metoda ta znajduje poprzedników obu studentów na liście i dokonuje zamiany ich miejscami.

4.7.2. zamienMiejscami() w strukturze Student

Metoda przyjmuje dwa wskaźniki na studentów ('student1' i 'student2') oraz wskaźnik na głowę listy ('listaHead'). Sprawdza, czy oba wskaźniki studentów są różne od 'nullptr'. Szuka poprzedników obu studentów w liście, aby móc prawidłowo zamienić ich miejscami. Jeśli znaleziono poprzedników dla obu studentów, wykonuje zamianę miejscami. Jeśli oba studenci nie są obok siebie, to zamienia ich miejscami na liście. Jeśli są obok siebie, obsługuje zamianę miejscami głowy listy, jeśli jedno z nich jest głową.

Umiejscowienie tych metod:

Struktura Student jest odpowiedzialna za przechowywanie informacji o jednym studencie oraz dostarcza specyficznych operacji na pojedynczym studencie, jak metoda 'zamienMiejscami()'. Lista 'ListaStudentow' jest odpowiedzialna za zarządzanie kolekcją studentów, umożliwiając dynamiczne dodawanie, usuwanie i sortowanie studentów. Metoda 'zamienMiejscami\_Studentow()' w klasie ListaStudentow wykorzystuje metodę 'zamienMiejscami()' z strukturze Student, aby realizować operację zamiany miejscami dwóch studentów w liście.

* 1. Zapisywanie wyników do pliku

void wpis\_do\_pliku() {

std::ofstream zapis("listaStudentow.txt");

std::shared\_ptr<Student> current = head;

if (zapis.is\_open()) {

while (current != nullptr) {

zapis << std::fixed << std::setw(11) << current->imie << "\t " << std::setw(11) << current->nazwisko << "\t";

for (int i = 0;i < 5; ++i) {

zapis << " " << std::setw(11) << current->przedmioty[i] << "[" << current->oceny[i] << "]\t";

}

zapis << std::endl;

current = current->next;

}

}

else {

std::cout << "Nie udalo sie otworzyc pliku." << std::endl;

}

zapis.close();

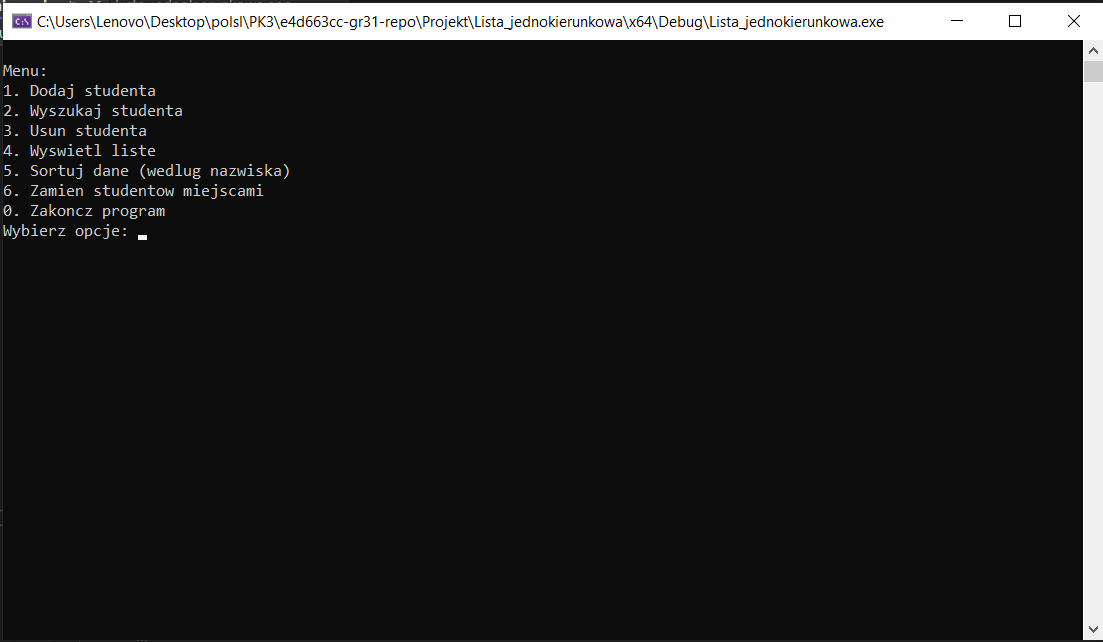
}

4.8.1. wpis\_do\_pliku()

Otwarcie pliku "listaStudentow.txt" do zapisu za pomocą strumienia ‘std::ofstream’. Sprawdzenie, czy plik został otwarty poprawnie. Iteracja przez listę studentów, zaczynając od głowy listy (‘head’). Zapisanie informacji o studencie, włączając imię, nazwisko, przedmioty i oceny, do pliku. Dodanie nowej linii po zapisie informacji o jednym studencie. Przejście do kolejnego studenta w liście. Zamknięcie pliku po zakończeniu operacji zapisu.

1. Opis testowania

Po poprawnym uruchomieniu programu w górnej części ekranu pokazuje się proste menu:



Po naciśnięciu jednego z klawiszy oznaczonych cyframi: 1, 2, 3, 4, 5, 6, 0 i potwierdzeniu klawiszem "enter" następuje uruchomienie odpowiedniej operacji. Następnie program zapisuje wykonane zmiany do pamięci, aż do naciśnięcia opcji 0, po której wszystko się zapisuje do pliku tekstowego „listaStudentow.txt” tym samym ją aktualizując.

Program może również pokazać następujące komunikaty w przypadku nieprawidłowego działania:

*Nie mozna otworzyc pliku do odczytu: "listaStudentów.txt"*

Oznacza on, że plik z danymi do posortowania nie istnieje, bądź istnieje lecz nie ma go w bieżącym katalogu.

*Nie udalo sie zapisać do pliku.*

Oznacza on, że nie można było otworzyć pliku do zapisu. Przyczyna może tkwić w braku miejsca na bieżącym dysku bądź w braku uprawnień użytkownika do zapisu na bieżącym dysku.

* 1. Opcja 1 „Dodaj studenta”

Po jej naciśnięciu użytkownik wprowadza w zadanej kolejności dane studenta:

- imię,

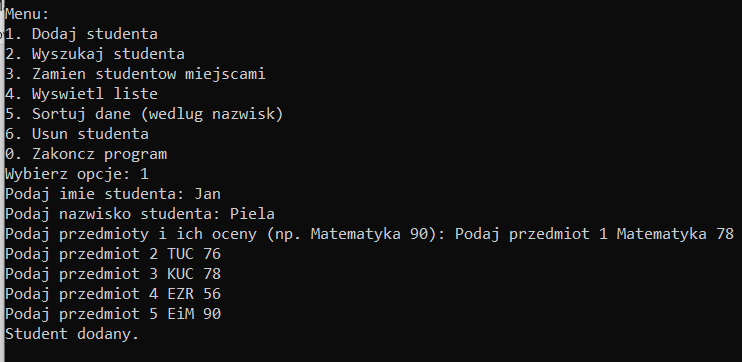
- nazwisko,

- przedmiot

- ocena

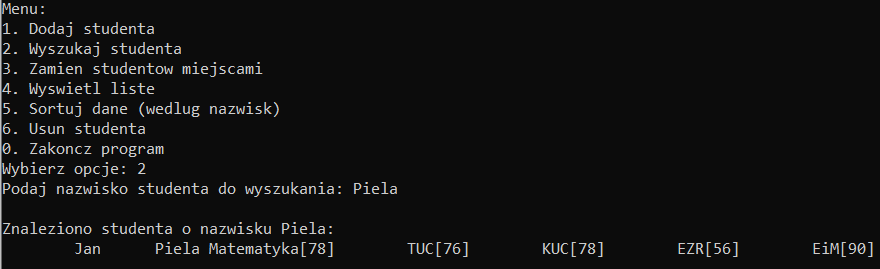
Te dwa atrybuty się podaje pięć razy, bo pięć przedmiotów i odpowiednio pięć ocen. Nazwy przedmiotów nie są określone, więc się podają ściśle na pogląd użytkownika. Po wprowadzeniu i naciśnięciu ‘enter’ wyświetla się komunikat:

*Student dodany.*

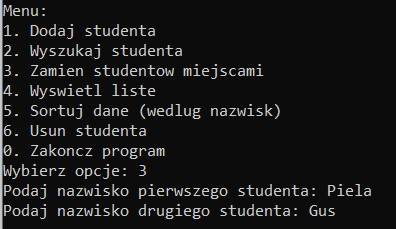


* 1. Opcja 2 ‘Wyszukaj studenta’

Po jego wyborze użytkownik wprowadza tylko nazwisko szukanego studenta i w wyniku ma wypisane jego dane:

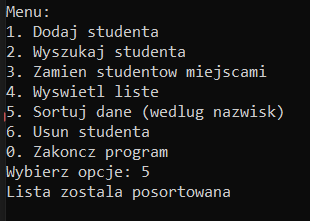


* 1. Opcja 3 ’Zamien studentow miejscami’



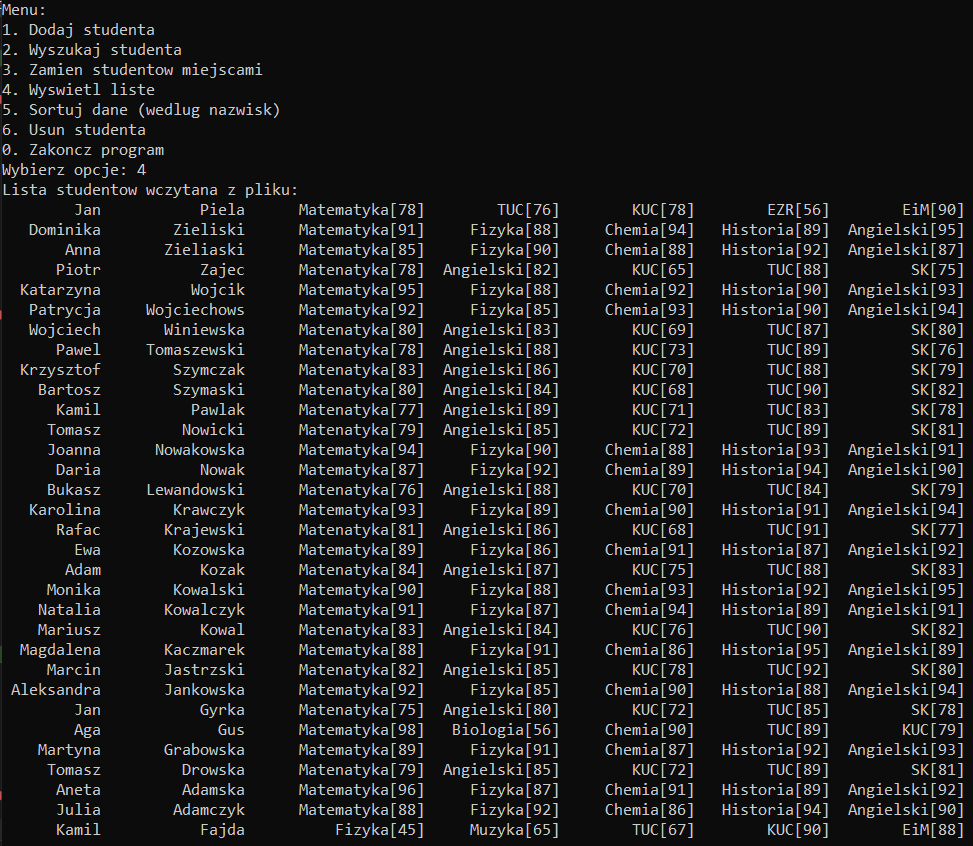
* 1. Opcja 4 ‘Sortuj dane (według nazwisk)’

Po jego naciśnięciu lista się sortuje według nazwisk:



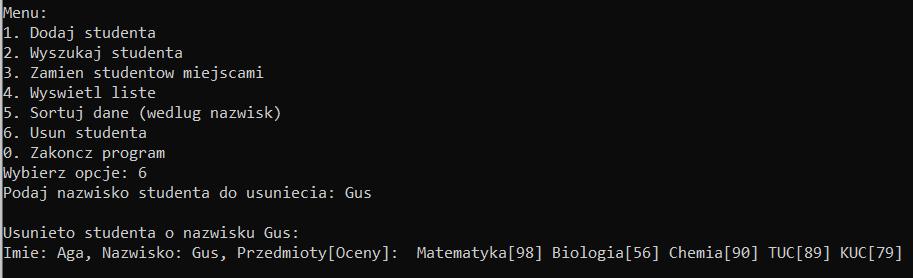
* 1. Opcja 5 ‘Wyswietl liste’

Po nadciśnieniu wyświetla się aktualna lista razem z jej modyfikacjami:



* 1. Opcja 6 ‘Usun studenta’

Po jego naciśnięciu użytkownik wprowadza nazwisko studenta, którego chce usunąć i on się usuwa z listy:



* 1. Opcja 0 ‘Zakoncz program’

Po naciśnięciu 0 program zapisuje wszystkie modyfikacje do pliku „listaStodentow.txt” i czeka na jeszcze jeden ‘enter’, aby program się skończył:

