Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Plan

autor	Daniel Wiszowaty
prowadzący	dr inż. Krzysztof Simiński
rok akademicki	2019/2020
kierunek	informatyka
rodzaj studiów	SSI
semestr	1
termin laboratorium	poniedziałek, 08:30 – 10:00
sekcja	22
termin oddania sprawozdania	2019-MM-DD

1 Treść zadania

W pliku zawarte są dane zajęć w następującym formacie:

(godzina rozpoczęcia)-(godzina zakończenia) (dzień) (grupa) (prowadzący) (przedmiot)

Godzina jest podana w formacie: `hh:mm`, dzień przyjmuje wartości: `pn`, `wt`, `sr`, `cz`, `pt`, `sb`, `nd`. Grupa, prowadzący i przedmiot to pojedyncze wyrazy. Przykładowy plik:

```
08:30-10:00 pt gr1 Kowalski Programowanie
10:15-11:45 wt gr2 Nowak Fizyka
14:34-15:43 sr gr2 Kowalski Java
07:23-19:34 cz gr1 Nowak Astronomia
```

W wyniku działania programu powstają pliki dla każdego prowadzącego (nazwa pliku jest tożsama z nazwiskiem prowadzącego) zawierający plan zajęć dla prowadzącego. Kolejne wpisu planu są posortowane chronologicznie. Przykładowy plik `Kowalski.txt`:

```
14:34-15:43 sr gr2 Java
08:30-10:00 pt gr1 Programowanie
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następującego przełącznika:

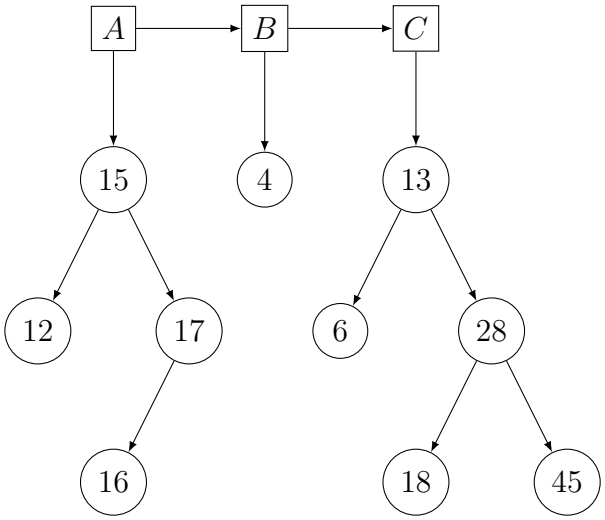
`-i` plik wejściowy

2 Analiza zadania

Zagadnienie przedstawia problem sortowania planu zajęć zapisanych w pliku wejściowym oraz przydzielanie posortowanych planów odpowiadającym prowadzącym.

2.1 Struktury danych

W programie wykorzystano listę podwieszaną. Lista nadrzędna przechowuje informacje o nazwisku prowadzącego. Lista nadrzędna zawiera wskaźnik na drzewo binarne, które przechowuje informacje o godzinie, dniu, grupie i przedmiocie. Drzewo posortowane jest według daty tj. dnia i godziny. Taka struktura umożliwia łatwe posortowanie i wypisanie planu każdego prowadzącego.



Rysunek 1: Przykład listy podwieszanej.

2.2 Algorytmy

Program sortuje zajęcia poprzez umieszczenie ich w drzewie binarnym. Wypisanie zajęć realizowane jest przez rekurencyjne przejście przez drzewo. Wypisanie planu dla każdego prowadzącego realizowane jest przez rekurencyjne przejście przez listę.

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Przy wywoływaniu programu możliwe jest użycie przełączników `-h`, `-i` oraz `-g`
Wykorzystanie przełącznika `-h` wyświetla instrukcje dla użytkownika obsługi programu. Po wykorzystaniu przełącznika `-i` należy przekazać do programu nazwę pliku wejściowego. Przełącznik `-g` generuje zadaną ilość wierszy do pliku wyjściowego. Domyślnie jest to format `.txt`

Przykładowe wywołanie programu:

```
./main -h
./main -g plik 100
./main -i plik.txt
```

Program zapisuje plan zajęć w pliku tekstowym w folderze zewnętrznym `pliki`. Plik tekstowy dla każdego prowadzącego jest nazwany nazwiskiem

prowadzącego. Pliki wejściowe mogą mieć dowolne rozszerzenie (lub go nie mieć.).

```
./main
./main -h
```

powoduje wyświetlenie krótkiej pomocy. Instrukcja wyświetlania jest również w wyniku podania niepoprawnych danych.

```
./main -g plik ilośćwierszy
```

Uruchomienie programu z parametrem `-g` powoduje wygenerowanie pliku `plik.txt` w folderze `pliki` zawierający losowy plan zajęć o zadanej przez użytkownika ilości wierszy który następnie można posortować. Gdy generowanie się powiedzie na ekranie pojawi się komunikat:

```
Wygenerowano plik <plik.txt> w folderze pliki
```

.
Uruchomienie programu z nieprawidłowymi parametrami powoduje wyświetlenie komunikatu

```
Podano zle argumenty do programu!
```

i wyświetlenie pomocy.

Podanie nieprawidłowej nazwy pliku powoduje wyświetlenie odpowiedniego komunikatu:

```
Plik plik.txt nie istnieje lub jest wadliwy
```

Podanie za dużej ilości argumentów powoduje wyświetlenie komunikatu:

```
Podano za dużo argumentów do programu
```

3.1 Ogólna struktura programu

Ogólna struktura programu, żeby czytelnik miał rozeznanie, co się w programie dzieje, jak program jest skonstruowany.

W funkcji głównej wywołana jest funkcja pobierzParametry. Funkcja ta sprawdza, czy program został wywołany w prawidłowy sposób. Gdy program nie został wywołany prawidłowo, zostaje wypisany stosowny komunikat i program się kończy. Następnie wywoływana jest funkcja wczytajDoDrzewa. Funkcja ta otwiera plik wejściowy, sczytuje liczby i umieszcza je w drzewie binarnym. Po sczytaniu wszystkich liczb funkcja zamyka plik. W razie wystąpienia błędu funkcja zwraca puste drzewo, w przeciwnym wypadku

– poprawną wartość korzenia drzewa. Następnie wywoływana jest funkcja zapiszDrzewoDoPliku. Funkcja przechodzi rekurencyjnie drzewo i zapisuje posortowane wartości do pliku wyjściowym. Po zapisaniu liczb funkcja zamyka plik. W razie wystąpienia błędu funkcja zwraca **false**, w przeciwnym wypadku – **true**. Ostatnią funkcją programu jest funkcja zwalniająca pamięć usunDrzewo.

3.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

4 Testowanie

Należy opisać jak program był testowany (na zbiorach poprawnych i typowych, na zbiorach poprawnych, ale nietypowych i wreszcie na zbiorach niepoprawnych). Należy opisać zbiory testowe.

Program został przetestowany na różnego rodzaju plikach. Pliki niepoprawne (niezawierające liczb, zawierający liczby w niepoprawnym formacie, niezgodne ze specyfikacją) powodują zgłoszenie błędu. Plik pusty nie powoduje zgłoszenia błędu, ale utworzenie pustego pliku wynikowego (został podany pusty zbiór liczb i pusty zbiór został zwrócony). Maksymalna liczba akceptowana w pliku zależy od kompilatora (typ **int** może być realizowany jako zmienna dwu- lub czterobajtowa). Maksymalna wielkość pliku, dla której udało się poprawnie uruchomić program, to 1.57 GB. Większe pliki wejściowe powodują błąd alokacji pamięci.

Program został sprawdzony pod kątem wycieków pamięci.

5 Wnioski

Proszę napisać, czy zrealizowali Państwo zadanie. Jeśli nie, to dlaczego się to nie powiodło. Warto napisać, czego się Państwo nauczyli podczas tworzenia programu, czy laboratorium wniosło coś konstruktywnego..., można dodać uwagi dotyczące przedmiotu, sposobu prowadzenia, sugerowane zmiany, czego Państwo się spodziewali po ćwiczeniach, a nie zostało to spełnione. Wnioski nie muszą być pisane lanym tekstem, mogą być wypunktowane.

Program do sortowania liczb jest programem prostym, chociaż wymaga samodzielnego zarządzania pamięcią. Najbardziej wymagające okazało się usunięcie wycieków pamięci. Szczególnie trudne było zapewnienie prawidłowego zwolnienia zaalokowanej pamięci, gdy część danych została wczytana do drzewa, po czym w pliku pojawiały się nieprawidłowe dane. Wtedy program powinien przerwać wczytywanie danych i wyświetlić komunikat, niezaniedbując zwolnienia pamięci.

Dla pewnych danych program wykonywał się poprawnie na niektórych komputerach, podczas gdy na innych maszynach generował niepoprawne wyniki. Było to spowodowane tym, że w zależności od kompilatora typ **int** ma 2 albo 4 bajty. Kompilacja typu jako dwubajtowego skutkowała przepełnieniem zakresu zmiennej w czasie wykonania.

[1].

Literatura

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. *Wprowadzenie do algorytmów.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2001.

Przed oddaniem sprawozdania proszę sprawdzić, czy jest poprawne pod względem:

- merytorycznym,
- językowym: błędy ortograficzne (np. pisownia łączna i rozdzielna, wielkie i małe litery), fleksyjne, składniowe, interpunkcyjne, stylistyczne (np. kolokwializmy, pleonazmy); pomocne tutaj mogą być strony [1], [2] lub [3].
- formalnym: m. in.: krój szeryfowy pisma, dzielenie wyrazów, wcięcia akapitów, wyjustowanie tekstu do lewego i prawego marginesu, brak spacji przed znakiem przestankowym, poprawne użycie apostrofu, „cudzysłowów", dywizu '-' i pauzy '–'.

[1] `http://sjp.pwn.pl`
[2] `http://sjp.pwn.pl/zasady`
[3] `https://pl.wikipedia.org/wiki/Pomoc:Powszechne_błędy_językowe`

Dodatek
Szczegółowy opis typów i funkcji

Projekt zaliczeniowy z PPK-SSI

Wygenerowano przez Doxygen 1.8.17

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury Godzina

`#include <struktury.h>`

Atrybuty publiczne

- int **Godzinka**
- int **Minuta**

3.1.1 Opis szczegółowy

Struktura reprezentująca godzinę

Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/struktury.h

3.2 Dokumentacja struktury Prowadzacy

`#include <struktury.h>`

Atrybuty publiczne

- string NazwiskoProwadzacego

  nazwisko przechowywane w liście
- Prowadzacy * pNastepnyProwadzacy

  adres następnego prowadzącego
- Zajecia * pKorzenListyZajec

  adres węzła drzewa poszukiwań binarnych

3.2.1 Opis szczegółowy

Lista jednokierunkowa

Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/struktury.h

3.3 Dokumentacja struktury Zajecia

```
#include <struktury.h>
```

Atrybuty publiczne

- Godzina PoczatekZajec
  godzina początku zajęć przechowywana w węźle
- Godzina KoniecZajec
  godzina końca zajęć przechowywana w węźle
- Dzien DzienZajec
  dzien przechowywany w węźle
- string Grupa
  grupa przechowywana w węźle
- string Przedmiot
  przedmioty przechowywany w węźle
- Zajecia * pLewy
  adres lewego potomka
- Zajecia * pPrawy
  adres prawego potomka

3.3.1 Opis szczegółowy

Węzeł drzewa poszukiwań binarnych

Dokumentacja dla tej struktury została wygenerowana z pliku:

- kod/struktury.h

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku kod/funkcje.h

```
#include <string>
```

Funkcje

- bool mniejsza (const Zajecia &pLewy, const Zajecia &pPrawy)
- string stringNaEnum (Dzien DzienZajec)
- Dzien enumNaString (const string &dzien)
- Prowadzacy * znajdzProwadzacegoRekurencyjnie (Prowadzacy *pGlowaListyProwadzacych, string nazwisko)
- Prowadzacy * dodajProwadzacegoNaKoniecListy (Prowadzacy *&pGlowaListyProwadzacych, string nazwisko)
- void dodajZajeciaProwadzacemu (Zajecia *&pKorzen, Godzina &PoczatekZajec, Godzina &KoniecZajec, Dzien &DzienZajec, string &grupa, string &przedmiot)
- void wypiszZajeciaProwadzacego (Zajecia *pKorzen, ofstream &strumien)
- void wypiszWszystkieZajecia (Prowadzacy *&pGlowaListyProwadzacych)
- void usunDrzewo (Zajecia *&pKorzen)
- void usunWszystko (Prowadzacy *&pGlowaListyProwadzacych)
- void wczytajZajeciaProwadzacemu (Prowadzacy *&pGlowaListyProwadzacych, string nazwisko, Godzina PoczatekZajec, Godzina KoniecZajec, Dzien DzienZajec, string grupa, string przedmiot)
- bool odczytajZPliku (Prowadzacy *&pGlowaListyProwadzacych, string &nazwaPlikuWejsciowego)
- bool sprawdzPlik (string &nazwaPlikuWejsciowego)
- int Silnik (size_t min, size_t max)
- void generujPlik (int ile, string &nazwaPlikuWyjsciowego)
- void instrukcja ()
- int pobierzArgumenty (int argc, char *argv[], string &nazwaPlikuWejsciowego, int &ile)

4.1.1 Dokumentacja funkcji

4.1.1.1 dodajProwadzacegoNaKoniecListy()

```
Prowadzacy* dodajProwadzacegoNaKoniecListy (
            Prowadzacy *& pGlowaListyProwadzacych,
            string nazwisko )
```

Funkcja dodaje rekurencyjnie element na koniec listy. Funkcja alokuje pamięć.

Parametry

pGlowaListyProwadzących	pierwszy element listy
nazwisko	element który chcemy dodać do listy

Zwraca

wskaźnik na ostatni element drzewa

4.1.1.2 dodajZajeciaProwadzacemu()

```
void dodajZajeciaProwadzacemu (
            Zajecia *& pKorzen,
            Godzina & PoczatekZajec,
            Godzina & KoniecZajec,
            Dzien & DzienZajec,
            string & grupa,
            string & przedmiot )
```

Funkcja dodaje iteracyjnie element do drzewa. Funkcja alokuje pamięć.

Parametry

pKorzen	korzen drzewa do którego dodawane są elementy
PoczatekZajec	godzina początkowa która ma być wstawiona
KoniecZajec	godzina końcowa która ma być wstawiona
grupa	grupa która ma być wstawiona
przedmiot	przedmiot który ma być wstawiony

4.1.1.3 enumNaString()

```
Dzien enumNaString (
            const string & dzien )
```

Funkcja konwertuje string na typ wyliczeniowy.

Parametry

DzienZajec	dzień zajęć

Zwraca

funkcja zwraca typ wyliczeniowy

4.1.1.4 generujPlik()

```
void generujPlik (
            int ile,
            string & nazwaPlikuWyjsciowego )
```

Funkcja generuje przykładowy plik do posortowowania.

Parametry

	ile	ilość wierszy w docelowym pliku tekstowym
out	*nazwapliku*	odczytana przez funkcję nazwa pliku wyjściowego

4.1.1.5 instrukcja()

```
void instrukcja ( )
```

Funkcja wypisująca instrukcję programu

4.1.1.6 mniejsza()

```
bool mniejsza (
            const Zajecia & pLewy,
            const Zajecia & pPrawy )
```

Funkcja porównuje datę(dzień, godzinę początkową a następnie godzinę końcową) w drzewie.

Parametry

pLewy	lewy potomek drzewa
pPrawy	prawy potomek drzewa

Zwraca

funkcja zwraca true jeżeli porównywana data jest mniejsza

4.1.1.7 odczytajZPliku()

```
bool odczytajZPliku (
            Prowadzacy *& pGlowaListyProwadzacych,
            string & nazwaPlikuWejsciowego )
```

Funkcja wczytuje zajęcia z pliku do struktury

Parametry

	pGlowaListyProwadzacych	element do którego dodawane są elementy
in	*nazwapliku*	odczytana przez funkcję nazwa pliku wejściowego

Zwraca

funkcja zwraca true jeśli uda się wczytać cały plik

4.1.1.8 pobierzArgumenty()

```
int pobierzArgumenty (
            int argc,
            char * argv[],
            string & nazwaPlikuWejsciowego,
            int & ile )
```

Funkcja pobiera parametry programu i sprawdza ich poprawność.

Parametry

	argc	liczba parametrów uruchomienia programu
	argv	tablica wskaźników na parametry uruchomienia programu
out	*nazwaPlikuWejsciowego*	odczytana przez funkcję nazwa pliku wyjściowego

Zwraca

funkcja zwraca liczbę kodu wyjścia

4.1.1.9 Silnik()

```
int Silnik (
            size_t min,
            size_t max )
```

Funkcja generuje liczbę z zadanego przedziału w oparciu o random_int_distribution.

Parametry

min	minimalna liczba
max	maksymalna liczba

Zwraca

funkcja zwraca losowa liczbę z przedziału

4.1.1.10 sprawdzPlik()

```
bool sprawdzPlik (
            string & nazwaPlikuWejsciowego )
```

Funkcja sprawdza poprawność danych w pliku przy użyciu wyrażen regularnych.

Parametry

in	*nazwaPlikuWejsciowego*	odczytana przez funkcję nazwa pliku wejściowego

Zwraca

funkcja zwraca true jeśli cały plik jest poprawny

4.1.1.11 stringNaEnum()

```
string stringNaEnum (
            Dzien DzienZajec )
```

Funkcja konwertuje typ wyliczeniowy na string.

Parametry

DzienZajec	dzień zajęć

Zwraca

funkcja zwraca string

4.1.1.12 usunDrzewo()

```
void usunDrzewo (
            Zajecia *& pKorzen )
```

Funkcja usuwa rekurencyjnie całe drzewo z pamięci.

Parametry

pKorzen	korzeń drzewa do usunięcia

4.1.1.13 usunWszystko()

```
void usunWszystko (
            Prowadzacy *& pGlowaListyProwadzacych )
```

Funkcja usuwa rekurencyjnie każde drzewo i całą listę.

Parametry

pGlowaListyProwadzacych	pierwszy element listy

4.1.1.14 wczytajZajeciaProwadzacemu()

```
void wczytajZajeciaProwadzacemu (
            Prowadzacy *& pGlowaListyProwadzacych,
            string nazwisko,
            Godzina PoczatekZajec,
            Godzina KoniecZajec,
            Dzien DzienZajec,
            string grupa,
            string przedmiot )
```

Funkcja tworzy unikalny element listy i do przypisanego mu drzewa dodaje zajęcia. Funkcja alokuje pamięć.

Parametry

pGlowaListyProwadzacych	pierwszy element listy
nazwisko	nazwisko które ma być wstawione (do listy)
PoczatekZajec	godzina początkowa która ma być wstawiona (do drzewa)
KoniecZajec	godzina końcowa która ma być wstawiona
grupa	grupa która ma być wstawiona
przedmiot	przedmiot który ma być wstawiony

4.1.1.15 wypiszWszystkieZajecia()

```
void wypiszWszystkieZajecia (
            Prowadzacy *& pGlowaListyProwadzacych )
```

Funkcja wypisuje rekurencyjnie posortowane drzewo dla każdego elementu listu jednokierunkowej.

Parametry

pGlowaListyProwadzacych	pierwszy element listy

4.1.1.16 wypiszZajeciaProwadzacego()

```
void wypiszZajeciaProwadzacego (
            Zajecia * pKorzen,
            ofstream & strumien )
```

Funkcja wypisuje rekurencyjnie drzewo do danego strumienia.

Parametry

pKorzen	korzeń drzewa z którego elementy mają być wypisane
strumien	strumień do którego elementy mają być wypisane

4.1.1.17 znajdzProwadzacegoRekurencyjnie()

```
Prowadzacy* znajdzProwadzacegoRekurencyjnie (
            Prowadzacy * pGlowaListyProwadzacych,
            string nazwisko )
```

Funkcja znajduje rekurencyjnie poszukiwany element w liście.

Parametry

pGlowaListyProwadzacych	pierwszy element listy
nazwisko	element w liście którego poszukujemy

Zwraca

wskaźnik na znaleziony element

4.2 Dokumentacja pliku kod/struktury.h

```
#include <string>
```

Komponenty

- struct Godzina
- struct Zajecia
- struct Prowadzacy

Wyliczenia

- enum Dzien {
  pn, **wt**, **sr**, **cz**,
  pt, **sb**, **nd** }

4.2.1 Dokumentacja typów wyliczanych

4.2.1.1 Dzien

`enum Dzien`

Typ wyliczeniowy reprezentujacy dni planu zajec

Indeks