

Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Plan

autor	Daniel Wiszowaty
prowadzący	dr inż. Artur Pasierbek
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 wpisy 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
```

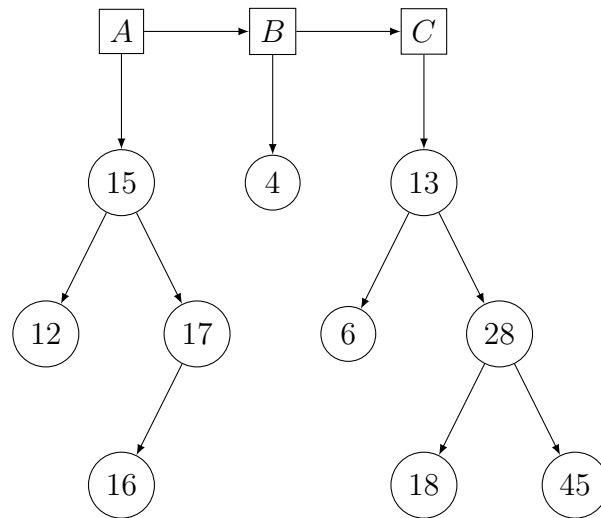
2 Analiza zadania

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 informację z `nazwiskiem 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.

2.2 Algorytmy



Rysunek 1: Przykład listy podwieszanej.

Dla elementu A zostały dodane liczby w kolejności 15, 12, 17, 16

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 i wypisanie odpowiadającego mu drzewa. [1]

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

3.1 Ogólna struktura programu

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 złe 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

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 `sprawdzPlik`. Funkcja ta przy pomocy wyrażeń regularnych sprawdza poprawność całego pliku wejściowego. Jeżeli plik jest nieodpowiedni na ekranie zostanie ukazany komunikat z numerem niepoprawnej linii. Funkcja zwraca `true` jeżeli plik jest

3.2 Szczegółowy opis typów i funkcji

poprawny a następnie zostaje wywołana funkcja `odczytajZPliku` która wczytuje kolejne linijki pliku do listy podwieszanej. Wywołana zostaje funkcja `wypiszWszystkieZajecia`. Funkcja przechodzi rekurencyjnie przez listę, wypisuje każde drzewo zapisuje posortowane zajęcia do plików wyjściowych. Po zapisaniu liczb funkcja zamyka plik. Ostatnią funkcją programu jest funkcja zwalniana pamięć `usunWszystko`.

3.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

4 Testowanie

Program testuje poprawność pliku przy użyciu **wyrażeń regularnych**. Spacje traktowane są jako osobny znak. W C++ należy stosować podwójny ukośnik wsteczny `\\`. Dane nie spełniającego podanego warunku zostaną wypisane w konsoli, ale sam program nie zostanie wykonany.

```
^(\\s*|\\t)*([01]\\d|2[0-3]):([0-5]\\d)-([01]\\d|2[0-3]):([0-5]\\d)\\t+
(pn|wt|sr|cz|pt|sb|nd)\\t+(gr[1-9])\\t+([A-Z][a-z]*)\\t+([A-Z]
[a-z]|[A-Z])*)\\s*$
```

<code>^</code>	początek linii
<code>\\s*</code>	biały znak 0 lub więcej razy
<code> </code>	lub
<code>[\\t]*</code>	spacja bądź tabulator 0 lub więcej razy
<code>[01]\\d</code>	cyfra z zakresu 0-1 i cyfra
<code>2[0-3]</code>	2 z przodu i cyfra z zakresu 0-3
<code>:</code>	dwukropek
<code>[0-5]\\d</code>	cyfra z zakresu 0-5 i cyfra
<code>-</code>	myślnik
<code>[\\t]+</code>	spacja bądź tabulator conajmniej raz
<code>(pn wt sr cz pt sb nd)</code>	odpowiedni dzień
<code>(gr[1-9])</code>	grupa i cyfra z zakresu 1-9
<code>([A-Z][a-z]+)</code>	słowo rozpoczynające się wielką literą
<code>([A-Z]([a-z] [A-Z])*)</code>	słowo rozpoczynające się wielką literą i dowolne litery
<code>\$</code>	koniec linii

Program został sprawdzony pod kątem wycieków pamięci przy użyciu biblioteki `nvwa` i `-fsanitize=leak` (biblioteki Clang).

5 Wnioski

Program do sortowania planu zajęć nie był programem trudnym, ale kluczowe było zastosowanie odpowiedniej struktury. Najtrudniejsze okazało się sprawdzenie poprawności danych w pliku wejściowym.

Literatura

- [1] Krzysztof Simiński. Wykłady z podstaw programowania komputerów.

Dodatek
Szczegółowy opis typów
i funkcji

Projekt zaliczeniowy z PPK-SSI

Wygenerowano przez Doxygen 1.8.17

1 Indeks klas	1
1.1 Lista klas	1
2 Indeks plików	3
2.1 Lista plików	3
3 Dokumentacja klas	5
3.1 Dokumentacja struktury Czas	5
3.1.1 Opis szczegółowy	5
3.2 Dokumentacja struktury Prowadzacy	5
3.2.1 Opis szczegółowy	6
3.3 Dokumentacja struktury Zajecia	6
3.3.1 Opis szczegółowy	6
4 Dokumentacja plików	7
4.1 Dokumentacja pliku kod/funkcje.h	7
4.1.1 Dokumentacja funkcji	7
4.1.1.1 dodajProwadzacegoNaKoniecListy()	7
4.1.1.2 dodajZajeciaProwadzacemu()	8
4.1.1.3 enumNaString()	8
4.1.1.4 generujPlik()	9
4.1.1.5 instrukcja()	9
4.1.1.6 mniejsza()	9
4.1.1.7 odczytajZPliku()	9
4.1.1.8 pobierzArgumenty()	10
4.1.1.9 Silnik()	10
4.1.1.10 sprawdzPlik()	11
4.1.1.11 stringNaEnum()	11
4.1.1.12 usunDrzewo()	11
4.1.1.13 usunWszystko()	12
4.1.1.14 wczytajZajeciaProwadzacemu()	12
4.1.1.15 wypiszWszystkieZajecia()	12
4.1.1.16 wypiszZajeciaProwadzacego()	13
4.1.1.17 znajdzProwadzacegoRekurencyjnie()	13
4.2 Dokumentacja pliku kod/struktury.h	13
4.2.1 Dokumentacja typów wyliczanych	14
4.2.1.1 Dzień	14
Indeks	15

Rozdział 1

Indeks klas

1.1 Lista klas

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

Czas	5
Prowadzacy	5
Zajecia	6

Rozdział 2

Indeks plików

2.1 Lista plików

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

kod/funkcje.h	7
kod/struktury.h	13

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury Czas

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

Atrybuty publiczne

- int **Godzina**
- 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

- [Czas PoczątekZajec](#)
godzina początku zajęć przechowywana w węźle
- [Czas KoniecZajec](#)
godzina końca zajęć przechowywana w węźle
- [Dzien DzieńZajec](#)
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 `mniej` (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, `Czas` &PoczątekZajec, `Czas` &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, `Czas` PoczątekZajec, `Czas` 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

<i>pGlowaListyProwadzących</i>	pierwszy element listy
<i>nazwisko</i>	element który chcemy dodać do listy

Zwraca

wskaźnik na ostatni element listy

4.1.1.2 dodajZajeciaProwadzacemu()

```
void dodajZajeciaProwadzacemu (
    Zajecia *& pKorzen,
    Czas & PoczatekZajec,
    Czas & KoniecZajec,
    Dzień & DzieńZajec,
    string & grupa,
    string & przedmiot )
```

Funkcja dodaje iteracyjnie element do drzewa. Funkcja alokuje pamięć.

Parametry

<i>pKorzen</i>	korzeń drzewa do którego dodawane są elementy
<i>PoczatekZajec</i>	godzina początkowa która ma być wstawiona
<i>KoniecZajec</i>	godzina końcowa która ma być wstawiona
<i>grupa</i>	grupa która ma być wstawiona
<i>przedmiot</i>	przedmiot który ma być wstawiony

4.1.1.3 enumNaString()

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

Funkcja konwertuje string na typ wyliczeniowy. Służy do wczytania danych z pliku.

Parametry

<i>DzieńZajec</i>	dzień zajęć
-------------------	-------------

Zwraca

funkcja zwraca typ wyliczeniowy

4.1.1.4 generujPlik()

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

Funkcja generuje przykładowy plik do posortowania.

Parametry

	<i>ile</i>	ilość wierszy w docelowym pliku tekstowym
out	<i>nazwapliku</i>	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

<i>pLewy</i>	lewy potomek drzewa
<i>pPrawy</i>	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

	<i>pGlowaListyProwadzacych</i>	element do którego dodawane są elementy
in	<i>nazwaPliku</i>	odczytana przez funkcję nazwa pliku wejściowego

Zwraca

funkcja zwraca true jeśli uda się poprawnie 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

	<i>argc</i>	liczba parametrów uruchomienia programu
	<i>argv</i>	tablica wskaźników na parametry uruchomienia programu
out	<i>nazwaPlikuWejsciowego</i>	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

<i>min</i>	minimalna liczba
<i>max</i>	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 wyrażeń regularnych.

Parametry

in	<i>nazwaPlikuWejsciowego</i>	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. Służy do wypisania planów do pliku.

Parametry

<i>DzienZajec</i>	dzień zajęć
-------------------	-------------

Zwraca

funkcja zwraca string

4.1.1.12 usunDrzewo()

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

Funkcja usuwa rekurencyjnie całe drzewo z pamięci.

Parametry

<i>pKorzen</i>	korzeń drzewa do usunięcia
----------------	----------------------------

4.1.1.13 usunWszystko()

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

Funkcja usuwa rekurencyjnie każde drzewo i całą listę.

Parametry

<i>pGlowaListyProwadzacych</i>	element listy od którego ma się zacząć usuwanie
--------------------------------	---

4.1.1.14 wczytajZajeciaProwadzacemu()

```
void wczytajZajeciaProwadzacemu (
    Prowadzacy *& pGlowaListyProwadzacych,
    string nazwisko,
    Czas PoczatekZajec,
    Czas KoniecZajec,
    Dzień DzieńZajec,
    string grupa,
    string przedmiot )
```

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

Parametry

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

4.1.1.15 wypiszWszystkieZajecia()

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

Funkcja wypisuje rekurencyjnie posortowane drzewo dla każdego elementu listy jednokierunkowej.

Parametry

<i>pGlowaListyProwadzacych</i>	element listy od którego ma się zacząć wypisywanie
--------------------------------	--

4.1.1.16 wypiszZajeciaProwadzacego()

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

Funkcja wypisuje rekurencyjnie drzewo do danego strumienia.

Parametry

<i>pKorzen</i>	korzeń drzewa z którego elementy mają być wypisane
<i>strumien</i>	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

<i>pGlowaListyProwadzacych</i>	pierwszy element listy
<i>nazwisko</i>	element w liście którego poszukujemy

Zwraca

wskaźnik na znaleziony element

4.2 Dokumentacja pliku kod/struktury.h

```
#include <string>
```

Komponenty

- struct [Czas](#)
- 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 reprezentujący dni planu zajęć

Indeks

Czas, [5](#)

dodajProwadzacegoNaKoniecListy
funkcje.h, [7](#)

dodajZajeciaProwadzacemu
funkcje.h, [8](#)

Dzien
struktury.h, [14](#)

enumNaString
funkcje.h, [8](#)

funkcje.h
dodajProwadzacegoNaKoniecListy, [7](#)
dodajZajeciaProwadzacemu, [8](#)
enumNaString, [8](#)
generujPlik, [8](#)
instrukcja, [9](#)
mniejsza, [9](#)
odczytajZPliku, [9](#)
pobierzArgumenty, [10](#)
Silnik, [10](#)
sprawdzPlik, [11](#)
stringNaEnum, [11](#)
usunDrzewo, [11](#)
usunWszystko, [12](#)
wczytajZajeciaProwadzacemu, [12](#)
wypiszWszystkieZajecia, [12](#)
wypiszZajeciaProwadzacego, [13](#)
znajdzProwadzacegoRekurencyjnie, [13](#)

generujPlik
funkcje.h, [8](#)

instrukcja
funkcje.h, [9](#)

kod/funkcje.h, [7](#)
kod/struktury.h, [13](#)

mniejsza
funkcje.h, [9](#)

odczytajZPliku
funkcje.h, [9](#)

pobierzArgumenty
funkcje.h, [10](#)
Prowadzacy, [5](#)

Silnik

funkcje.h, [10](#)

sprawdzPlik
funkcje.h, [11](#)

stringNaEnum
funkcje.h, [11](#)

struktury.h
Dzien, [14](#)

usunDrzewo
funkcje.h, [11](#)

usunWszystko
funkcje.h, [12](#)

wczytajZajeciaProwadzacemu
funkcje.h, [12](#)

wypiszWszystkieZajecia
funkcje.h, [12](#)

wypiszZajeciaProwadzacego
funkcje.h, [13](#)

Zajecia, [6](#)
znajdzProwadzacegoRekurencyjnie
funkcje.h, [13](#)