

Politechnika Śląska  
Wydział Automatyki, Elektroniki i Informatyki

## **Podstawy Programowania Komputerów**

Temat: Dwudzielny (10)

---

Autor	Piotr Janowski
Prowadzący	Dr inż. Krzysztof Simiński
Rok akademicki	2019/2020
Kierunek	Informatyka
Rodzaj studiów	SSI
Semestr	1
Termin laboratorium	Poniedziałek, 10:15 - 12:45
Sekcja	1
Termin oddania sprawozdania	2020-01-18

---



## Treść zadania

Napisać program, do sprawdzania, czy graf nieskierowany jest dwudzielny. Plik z grafem ma następującą postać:

- Każda krawędź jest podana w osobnej linii; podane są dwa wierzchołki, które łączy krawędź.
- W pliku mogą wystąpić puste linie.
- W linii mogą wystąpić dodatkowe (nadmiarowe) znaki białe.

Program wypisuje do pliku wyjściowego zadany graf i komunikat, czy jest to graf dwudzielny, czy nie. Jeżeli zadany graf jest dwudzielny, program wypisuje wierzchołki obu grup grafu.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna): -i plik wejściowy z krawędziami grafu -o plik wyjściowy z wynikami

## 1. Analiza zadania

Problem z zagadnieniu polega na stworzeniu struktury dowolnego grafu nieskierowanego, którego krawędzie zostały podane w zadaniu. Następnie należy sprawdzić jego dwudzielność, czyli możliwości nadania dwóch różnych parametrów każdemu węzłowi, tak aby sąsiadujące węzły miały go różny.

### 1.1. Struktury danych

W programie wykorzystana jest lista jednokierunkowa zawierająca struktury węzłów grafu. Każdy węzeł zawiera wskaźnik na listę jednokierunkową struktur reprezentujących krawędzie. Taka struktura pozwala na wierne odwzorowanie grafu przy pozostawieniu możliwości korzystania z niej jak ze zwykłej listy jednokierunkowej.

### 1.2. Algorytmy

Po załadowaniu danych z pliku, program poprzez kilkukrotne przejście przez listę węzłów przyporządkowuje, sprawdza czy graf jest dwudzielny. Algorytm realizuje tę funkcję poprzez sprawdzenie, do której grupy należą sąsiedzi i jeżeli nadal istnieje możliwość wystąpienia dwudzielności, przyporządkowuje ich do grupy przeciwnej niż aktualnie sprawdzany węzeł. W przypadku optymistycznym złożoność obliczeniowa wynosić będzie  $O(1)$ , natomiast w pesymistycznym  $O(W+a)$ , gdzie  $W$  reprezentuje liczbę węzłów w grafie,  $a$  liczbę grafów spójnych wchodzących w skład grafu niespójnego.

## 2. Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: wejściowego i wyjściowego po odpowiednich przełącznikach (odpowiednio: -i, -o) np.

```
Program.exe -i dane_wejsciowe.txt -o wynik.txt
```

```
Program.exe -o wyjscie.txt -i dane.txt
```

Pliki są plikami tekstowymi w rozszerzeniu txt. Przełączniki mogą być podane w dowolnej kolejności. Uruchomienie programu bez żadnego parametru lub z niewłaściwymi parametrami

```
Program.exe
```

```
Program.exe -p dane.txt -f wyniki.txt
```

powoduje wyświetlenie informacji, jak powinna wyglądać komenda z poprawnymi parametrami. Podanie dwa razy pliku wejściowego lub wyjściowego skutkuje wyświetleniem odpowiednio komunikatu

```
Podano dwa razy plik wejsciowy/wyjsciowy.
```

Podanie niewłaściwych nazw pliku powoduje otrzymanie komunikatów:

```
Podany plik wejsciowy nie istnieje!
```

```
Podany plik wyjsciowy nie istnieje!
```

W każdym przypadku podania niewłaściwych danych, program kończy swoje działanie.

## 3. Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym.

### 3.1. Ogólna struktura programu

Funkcja `main` ma za zadanie uporządkowanie kolejności wywoływania pozostałych funkcji. Pierwszą z nich jest `controlling`, która pobiera dane z klawiatury i zapisuje nazwy pliku wejściowego i wyjściowego. Funkcja zwraca **false** w wypadku podania błędnych parametrów, co skutkuje zakończeniem programu ze zwróceniem błędu. Następna w kolejności funkcja `download`, odpowiada za sczytanie danych z pliku oraz utworzenie struktury grafu. W przypadku uszkodzonego pliku wejściowego, funkcja zwraca **false**, co skutkuje zakończeniem całego programu ze zwróceniem błędu. Następnie, jeżeli plik wejściowy nie był pusty uruchamiane jest funkcja `assignNeigh`. Przypisuje ona wskaźniki węzłów sąsiadujących w każdej liście krawędzi. Funkcja `printGraph` zapisuje w pliku strukturę grafu (wypisuje węzeł, a w następnej linii węzły z nim sąsiadujące). Następnie program poprzez funkcję `coloring` sprawdza, czy zadany graf jest dwudzielny. Funkcja działa w sposób następujący: po przyporządkowaniu do

grupy pierwszego wężła, koloruje na kolor przeciwny jego sąsiadów. Następnie przechodzi do następnego, zakolorowanego wierzchołka i sprawdza czy jego sąsiedzi są w przeciwnym do niego kolorze. Jeżeli tak, koloruje swoich sąsiadów i przechodzi do następnego zakolorowanego wężła. Jeżeli graf jest niespójny, powyższy algorytm stosowany jest do każdego grafu składowego. Jeżeli graf jest dwudzielny, funkcja `coloring` zwraca **true** i uruchamiana jest funkcja `printByColor`, która zapisuje w pliku wyjściowym informacje o dwudzielności grafu oraz wężły z podziałem na dwie grupy dwudzielności. Ostatnia funkcja `deleteGraph` usuwa rekurencyjnie utworzoną listę.

### 3.2. Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

## 4. Testowanie

Program został przetestowany na różnego rodzaju plikach, zawierających liczby. Wczytanie pliku zawierającego dane inne niż liczby, powoduje zgłoszenie błędu. Jeżeli w pliku znajduje się nieparzysta ilość liczb, program ignoruje ostatnią. Pusty plik wejściowy nie powoduje zgłoszenia błędu, jednak nie zostanie utworzony plik wyjściowy. Jeżeli w pliku wystąpi podanie samego wierzchołka (krawędź między tym samym wężłem), program poinformuje, że plik wejściowy jest uszkodzony. Maksymalny rozmiar pliku, na którym program został przetestowany to 10.8MB

## 5. Wnioski

Programy sprawdzające dwudzielność grafu korzystają zwykle z algorytmu przeszukiwania w głąb, realizowanego rekurencyjnie. Przy podejściu iteracyjnym jednym z najtrudniejszych problemów okazało się przystosowanie programu do działania na grafach niespójnych. Sporym wyzwaniem okazało się również uruchamianie programu z poziomu wiersza poleceń.

## Źródła

[https://pl.wikipedia.org/wiki/Teoria\\_graf%C3%B3w](https://pl.wikipedia.org/wiki/Teoria_graf%C3%B3w)

<https://docs.microsoft.com/pl-pl/visualstudio/debugger/finding-memory-leaks-using-the-crt-library?view=vs-2019>

<http://www.cplusplus.com/articles/DEN36Up4/>

---

<http://rab.ict.pwr.wroc.pl/~kreczmer/wds/prezentacje/10-doxygen.pdf>

Dwudzielnny (10)

Generated by Doxygen 1.8.17





<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 krawedz Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	6
3.1.2.1 pNeighbour	6
3.1.2.2 pNext	6
3.1.2.3 wyjscie	6
3.2 wezel Struct Reference	6
3.2.1 Detailed Description	7
3.2.2 Member Data Documentation	7
3.2.2.1 done	7
3.2.2.2 grupa	7
3.2.2.3 pNeighbour	7
3.2.2.4 pNext	8
3.2.2.5 wartosc	8
<b>4 File Documentation</b>	<b>9</b>
4.1 funkcje.cpp File Reference	9
4.1.1 Function Documentation	10
4.1.1.1 addKrawedz()	10
4.1.1.2 assignNeigh()	11
4.1.1.3 checkOut()	12
4.1.1.4 coloring()	13
4.1.1.5 controlling()	14
4.1.1.6 deleteGraph()	15
4.1.1.7 deleteKrawedz()	15
4.1.1.8 download()	16
4.1.1.9 findKrawedz()	17
4.1.1.10 findWezel()	18
4.1.1.11 ifPossible()	19
4.1.1.12 makeColors()	19
4.1.1.13 printByColor()	20
4.1.1.14 printGraph()	20
4.2 funkcje.h File Reference	21
4.2.1 Function Documentation	22
4.2.1.1 addKrawedz()	22
4.2.1.2 assignNeigh()	23

4.2.1.3 checkOut()	24
4.2.1.4 coloring()	24
4.2.1.5 controlling()	25
4.2.1.6 deleteGraph()	26
4.2.1.7 deleteKrawedz()	27
4.2.1.8 download()	28
4.2.1.9 findKrawedz()	29
4.2.1.10 findWezel()	29
4.2.1.11 ifPossible()	30
4.2.1.12 makeColors()	31
4.2.1.13 printByColor()	31
4.2.1.14 printGraph()	32
4.3 main.cpp File Reference	32
4.3.1 Function Documentation	33
4.3.1.1 main()	33
4.4 struktury.h File Reference	34
4.4.1 Enumeration Type Documentation	35
4.4.1.1 kolor	35
<b>Index</b>	<b>37</b>

# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">krawedz</a> . . . . .	5
<a href="#">wezel</a> . . . . .	6



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">funkcje.cpp</a>	9
<a href="#">funkcje.h</a>	21
<a href="#">main.cpp</a>	32
<a href="#">struktury.h</a>	34



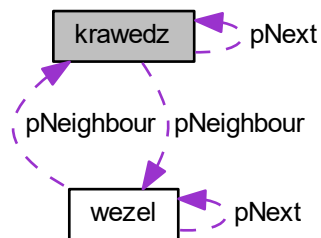
## Chapter 3

# Class Documentation

### 3.1 krawedz Struct Reference

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

Collaboration diagram for krawedz:



#### Public Attributes

- `int wyjście`  
*wartość węzła, do którego wchodzi podana krawędź*
- `wezel * pNeighbour`  
*wskaźnik na węzeł, do którego wchodzi podana krawędź*
- `krawedz * pNext`  
*wskaźnik na następną krawędź w liście*

#### 3.1.1 Detailed Description

Krawędź grafu.

### 3.1.2 Member Data Documentation

#### 3.1.2.1 pNeighbour

```
wezel* krawedz::pNeighbour
```

wskaźnik na węzeł, do którego wchodzi podana krawędź

#### 3.1.2.2 pNext

```
krawedz* krawedz::pNext
```

wskaźnik na następną krawędź w liście

#### 3.1.2.3 wyjście

```
int krawedz::wyjście
```

wartość węzła, do którego wchodzi podana krawędź

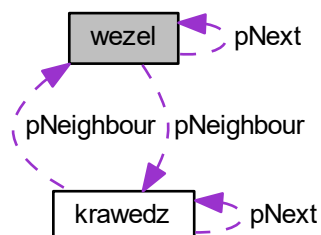
The documentation for this struct was generated from the following file:

- [struktury.h](#)

## 3.2 wezel Struct Reference

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

Collaboration diagram for wezel:





## Public Attributes

- int `wartosc`  
*wartość węzła*
- `kolor grupa`  
*kolor grupy, do której należy węzeł*
- `krawiedz * pNeighbour`  
*wskaźnik na listę krawędzi wychodzących z węzła*
- `wezel * pNext`  
*wskaźnik na następny węzeł w liście*
- int `done`  
*parametr, określający czy dla danego węzła została wykonana funkcja coloring*

### 3.2.1 Detailed Description

Węzeł grafu.

### 3.2.2 Member Data Documentation

#### 3.2.2.1 done

```
int wezel::done
```

parametr, określający czy dla danego węzła została wykonana funkcja coloring

#### 3.2.2.2 grupa

```
kolor wezel::grupa
```

kolor grupy, do której należy węzeł

#### 3.2.2.3 pNeighbour

```
krawiedz* wezel::pNeighbour
```

wskaźnik na listę krawędzi wychodzących z węzła

#### 3.2.2.4 pNext

```
wezel* wezel::pNext
```

wskaźnik na następny węzeł w liście

#### 3.2.2.5 wartosc

```
int wezel::wartosc
```

wartość węzła

The documentation for this struct was generated from the following file:

- [struktury.h](#)

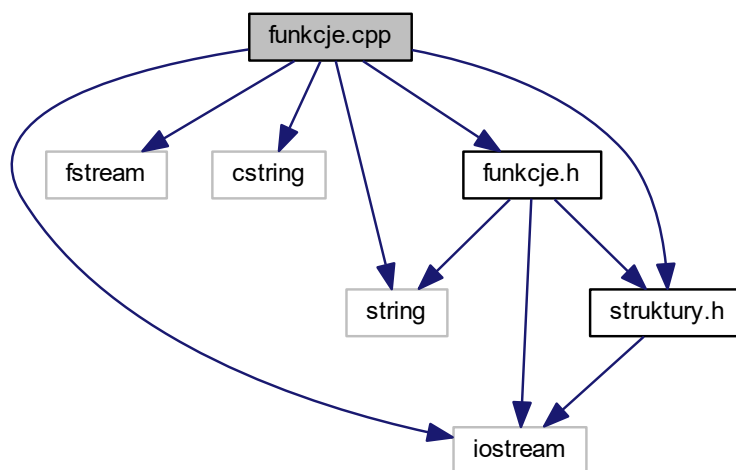
## Chapter 4

# File Documentation

### 4.1 funkcje.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <string>
#include "funkcje.h"
#include "struktury.h"
```

Include dependency graph for funkcje.cpp:



### Functions

- bool `controlling` (int argc, char \*argv[], string &input, string &output)
- `wezel * findWezel` (`wezel *pHead`, int szukana)
- `krawedz * findKrawedz` (`krawedz *pHead`, int szukana)

- `krawedz * addKrawedz (krawedz *pHead, int a)`
- `bool download (wezel *&pHead, string input)`
- `void deleteKrawedz (krawedz *&pHead)`
- `void deleteGraph (wezel *&pHead)`
- `void printGraph (wezel *pHead, string output)`
- `void assignNeigh (wezel *pHead)`
- `bool ifPossible (krawedz *pHead, kolor current)`
- `void makeColors (krawedz *pHead, kolor current)`
- `bool checkOut (wezel *pHead)`
- `bool coloring (wezel *pHead)`
- `void printByColor (wezel *pHead, string output)`

## 4.1.1 Function Documentation

### 4.1.1.1 addKrawedz()

```
krawedz* addKrawedz (
    krawedz * pHead,
    int a )
```

Funkcja dodaje krawędź, do aktualnego węzła

#### Parameters

<i>pHead</i>	adres pierwszego elementu listy krawędzi
<i>a</i>	wartość węzła, do którego krawędź wchodzi

#### Returns

adres nowo powstałej krawędzi

Here is the caller graph for this function:



#### 4.1.1.2 assignNeigh()

```
void assignNeigh (
    wezel * pHead )
```

Funkcja dodaje dok każdej krawędzi adres węzła, do którego dana krawędź wchodzi

**Parameters**

<i>pHead</i>	adres pierwszego elementu listy węzłów
--------------	--

Here is the call graph for this function:



Here is the caller graph for this function:

**4.1.1.3 checkOut()**

```
bool checkOut (
    wezel * pHead )
```

Funkcja sprawdza, czy dla wszystkich węzłów została wykonana funkcja coloring

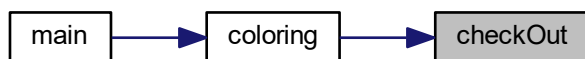
**Parameters**

<i>pHead</i>	adres pierwszego elementu listy węzłów
--------------	--

**Returns**

w wypadku wykonania funkcji coloring dla wszystkich węzłów zwraca TRUE, w przeciwnym wypadku FALSE

Here is the caller graph for this function:

**4.1.1.4 coloring()**

```
bool coloring (  
    wezel * pHead )
```

Funkcja sprawdza, czy graf jest dwudzielny

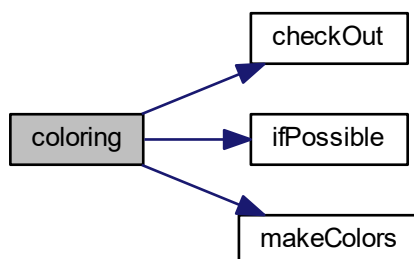
**Parameters**

<i>pHead</i>	adres pierwszego elementu listy węzłów
--------------	--

**Returns**

w wypadku dwudzielności funkcja zwraca TRUE, w przeciwnym wypadku FALSE

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.1.5 controlling()

```
bool controlling (
    int argc,
    char * argv[],
    string & input,
    string & output )
```

Funkcja przypisuje wprowadzone w konsoli parametry plików i sprawdza ich poprawność

##### Parameters

	<i>argc</i>	ilość przyjmowanych parametrów z konsoli
	<i>argv[]</i>	wartości parametrów
<i>in, out</i>	<i>input</i>	zmienna do zapisania nazwy pliku z danymi wejściowymi
<i>in, out</i>	<i>output</i>	zmienna do zapisania nazwy pliku wyjściowego

##### Returns

Funkcja zwraca TRUE w wypadku gdy wszystkie parametry zostały podane poprawnie, a pliki istnieją, w przeciwnym wypadku zwraca FALSE

Here is the caller graph for this function:





#### 4.1.1.6 deleteGraph()

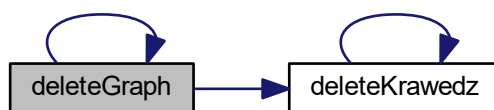
```
void deleteGraph (
    wezel *& pHead )
```

Funkcja rekurencyjnie usuwa listę węzłów z pamięci

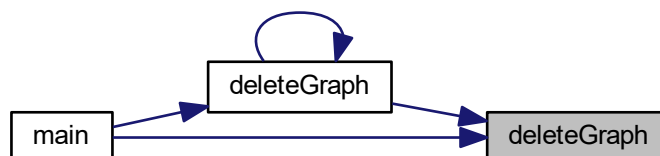
##### Parameters

in, out	<i>pHead</i>	adres pierwszego elementu listy węzłów
---------	--------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.1.7 deleteKrawedz()

```
void deleteKrawedz (
    krawedz *& pHead )
```

Funkcja rekurencyjnie usuwa listę krawędzi dla danego węzła z pamięci

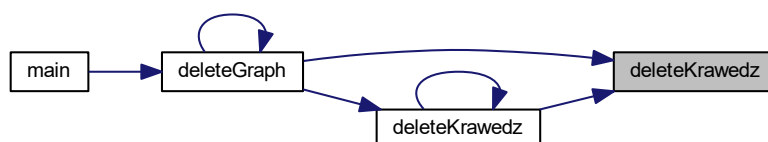
##### Parameters

in, out	<i>pHead</i>	adres pierwszego elementu listy krawędzi
---------	--------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.1.8 download()

```

bool download (
    wezel *& pHead,
    string input )
  
```

Funkcja tworzy listę jednokierunkową złożoną z węzłów grafu, dane pobierane są z pliku wejściowego

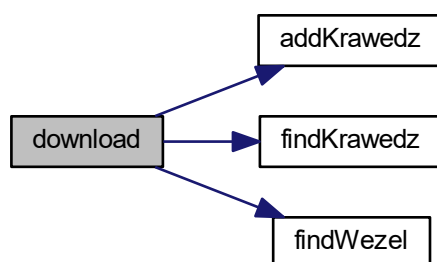
##### Parameters

<code>in, out</code>	<code>pHead</code>	adres pierwszego elementu listy węzłów
	<code>input</code>	nazwa pliku z danym wejściowymi

### Returns

Funkcja zwraca FALSE, jeśli plik wejściowy jest uszkodzony, TRUE jeżeli wczytanie danych przebiegło pomyślnie

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.1.9 findKrawedz()

```
krawedz* findKrawedz (  
    krawedz * pHead,  
    int szukana )
```

Funkcja sprawdza, czy z aktualnego węzła istnieją już krawędzie do węzła sąsiadującego

### Parameters

<i>pHead</i>	adres pierwszego elementu listy krawędzi
<i>szukana</i>	wartość węzła, do którego połączenie jest sprawdzane

**Returns**

adres krawędzi

Here is the caller graph for this function:

**4.1.1.10 findWezel()**

```
wezel* findWezel (  
    wezel * pHead,  
    int szukana )
```

Funkcja sprawdza, czy węzeł o wartości czytanej z pliku już istnieje

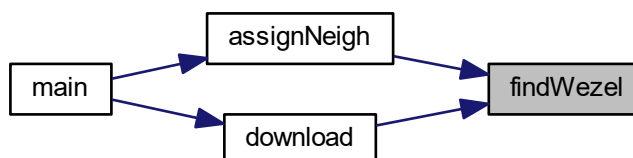
**Parameters**

<i>pHead</i>	adres pierwszego elementu listy węzłów
<i>szukana</i>	wartosc szukanego węzła

**Returns**

adres węzła o szukanej wartości

Here is the caller graph for this function:



#### 4.1.1.11 ifPossible()

```
bool ifPossible (
    krawedz * pHead,
    kolor current )
```

Funkcja sprawdza, czy dla danego zakolorowanie wężła i jego sąsiadów może wystąpić dwudzielność całego grafu

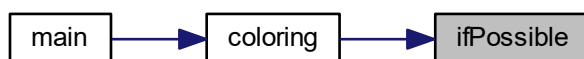
##### Parameters

<i>pHead</i>	adres pierwszego elementu listy krawędzi
<i>current</i>	kolor wężła, dla którego wykonywana jest funkcja

##### Returns

w wypadku nie przeczącym dwudzielności zwraca TRUE, w przeciwnym wypadku FALSE

Here is the caller graph for this function:



#### 4.1.1.12 makeColors()

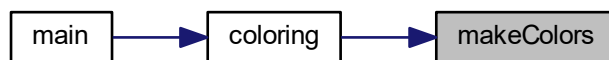
```
void makeColors (
    krawedz * pHead,
    kolor current )
```

Funkcja nadaje kolory sąsiednim węzłom

##### Parameters

<i>pHead</i>	adres pierwszego elementu listy krawędzi
<i>current</i>	kolor wężła, dla ktorego wykonywana jest funkcja

Here is the caller graph for this function:



#### 4.1.1.13 printByColor()

```
void printByColor (
    wezel * pHead,
    string output )
```

Funkcje, w wypadku dwudzielności grafu, wypisuje do pliku informację o dwudzielności oraz wartości węzłów dla obu grup

##### Parameters

<i>pHead</i>	adres pierwszego elementu listy węzłów
<i>output</i>	nazwa pliku wyjściowego

Here is the caller graph for this function:



#### 4.1.1.14 printGraph()

```
void printGraph (
    wezel * pHead,
    string output )
```

Funkcja wypisuje do pliku graf (węzły oraz ich sąsiadów)

## Parameters

<i>pHead</i>	adres pierwszego elementu listy węzłów
<i>nazwa</i>	pliku wyjściowego

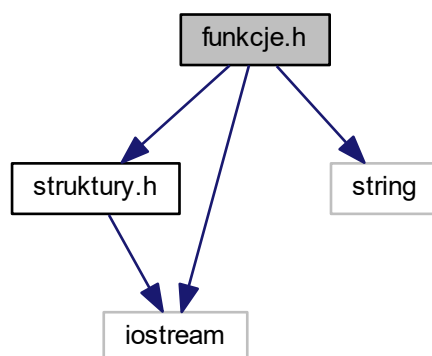
Here is the caller graph for this function:



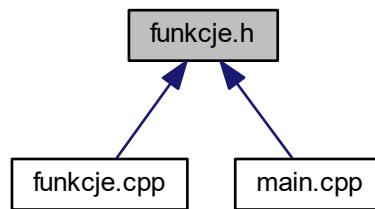
## 4.2 funkcje.h File Reference

```
#include "struktury.h"  
#include <iostream>  
#include <string>
```

Include dependency graph for funkcje.h:



This graph shows which files directly or indirectly include this file:



## Functions

- bool `controlling` (int argc, char \*argv[], string &input, string &output)
- bool `download` (wezel \*&pHead, string input)
- wezel \* `findWezel` (wezel \*pHead, int szukana)
- krawedz \* `findKrawedz` (krawedz \*pHead, int szukana)
- krawedz \* `addKrawedz` (krawedz \*pHead, int a)
- void `deleteGraph` (wezel \*&pHead)
- void `deleteKrawedz` (krawedz \*&pHead)
- void `printGraph` (wezel \*pHead, string output)
- void `assignNeigh` (wezel \*pHead)
- bool `coloring` (wezel \*pHead)
- bool `ifPossible` (krawedz \*pHead, kolor current)
- void `makeColors` (krawedz \*pHead, kolor current)
- bool `checkOut` (wezel \*pHead)
- void `printByColor` (wezel \*pHead, string output)

## 4.2.1 Function Documentation

### 4.2.1.1 addKrawedz()

```

krawedz* addKrawedz (
    krawedz * pHead,
    int a )
  
```

Funkcja dodaje krawędź, do aktualnego wężła

#### Parameters

<i>pHead</i>	adres pierwszego elementu listy krawędzi
<i>a</i>	wartość wężła, do którego krawędź wchodzi



**Returns**

adres nowo powstałej krawędzi

Here is the caller graph for this function:

**4.2.1.2 assignNeigh()**

```
void assignNeigh (  
    wezel * pHead )
```

Funkcja dodaje dok każdej krawędzi adres węzła, do którego dana krawędź wchodzi

**Parameters**

<i>pHead</i>	adres pierwszego elementu listy węzłów
--------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.2.1.3 checkOut()

```
bool checkOut (
    wezel * pHead )
```

Funkcja sprawdza, czy dla wszystkich węzłów została wykonana funkcja coloring

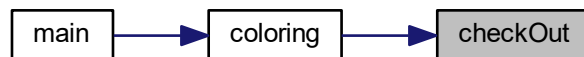
##### Parameters

<i>pHead</i>	adres pierwszego elementu listy węzłów
--------------	--

##### Returns

w wypadku wykonania funkcji coloring dla wszystkich węzłów zwraca TRUE, w przeciwnym wypadku FALSE

Here is the caller graph for this function:



#### 4.2.1.4 coloring()

```
bool coloring (
    wezel * pHead )
```

Funkcja sprawdza, czy graf jest dwudzielny

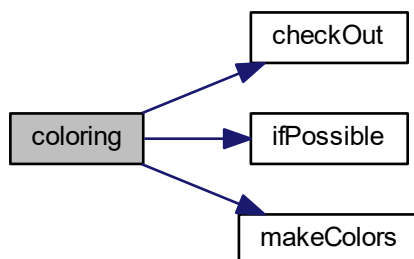
##### Parameters

<i>pHead</i>	adres pierwszego elementu listy węzłów
--------------	--

**Returns**

w wypadku dwudzielności funkcja zwraca TRUE, w przeciwnym wypadku FALSE

Here is the call graph for this function:



Here is the caller graph for this function:

**4.2.1.5 controlling()**

```
bool controlling (  
    int argc,  
    char * argv[],  
    string & input,  
    string & output )
```

Funkcja przypisuje wprowadzone w konsoli parametry plików i sprawdza ich poprawność

**Parameters**

	<i>argc</i>	ilość przyjmowanych parametrów z konsoli
	<i>argv[]</i>	wartości parametrów
<i>in, out</i>	<i>input</i>	zmienna do zapisania nazwy pliku z danymi wejściowymi
<i>in, out</i>	<i>output</i>	zmienna do zapisania nazwy pliku wyjściowego

### Returns

Funkcja zwraca TRUE w wypadku gdy wszystkie parametry zostały podane poprawnie, a pliki istnieją, w przeciwnym wypadku zwraca FALSE

Here is the caller graph for this function:



#### 4.2.1.6 deleteGraph()

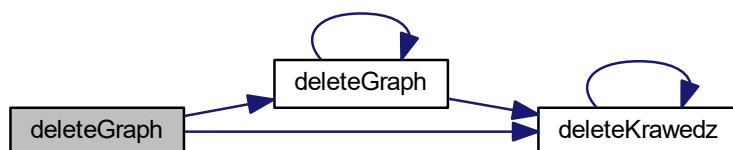
```
void deleteGraph (
    wezel *& pHead )
```

Funkcja rekurencyjnie usuwa listę węzłów z pamięci

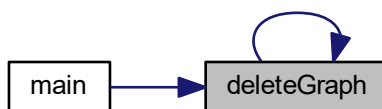
### Parameters

in, out	<i>pHead</i>	adres pierwszego elementu listy węzłów
---------	--------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.2.1.7 deleteKrawedz()

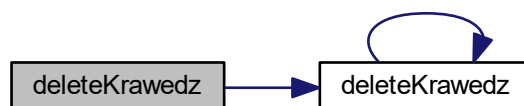
```
void deleteKrawedz (
    krawedz *& pHead )
```

Funkcja rekurencyjnie usuwa listę krawędzi dla danego wężła z pamięci

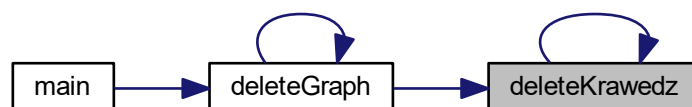
##### Parameters

<code>in, out</code>	<code>pHead</code>	adres pierwszego elementu listy krawędzi
----------------------	--------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.2.1.8 download()

```
bool download (
    wezel *& pHead,
    string input )
```

Funkcja tworzy listę jednokierunkową złożoną z węzłów grafu, dane pobierane są z pliku wejściowego

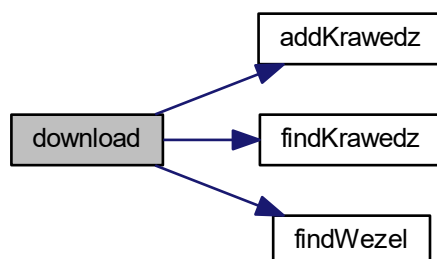
##### Parameters

<i>in, out</i>	<i>pHead</i>	adres pierwszego elementu listy węzłów
	<i>input</i>	nazwa pliku z danym wejściowymi

##### Returns

Funkcja zwraca FALSE, jeśli plik wejściowy jest uszkodzony, TRUE jeżeli wczytanie danych przebiegło pomyślnie

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.2.1.9 findKrawedz()

```
krawedz* findKrawedz (
    krawedz * pHead,
    int szukana )
```

Funkcja sprawdza, czy z aktualnego węzła istnieją już krawędzie do węzła sąsiadującego

##### Parameters

<i>pHead</i>	adres pierwszego elementu listy krawędzi
<i>szukana</i>	wartość węzła, do którego połączenie jest sprawdzane

##### Returns

adres krawędzi

Here is the caller graph for this function:



#### 4.2.1.10 findWezel()

```
wezel* findWezel (
    wezel * pHead,
    int szukana )
```

Funkcja sprawdza, czy węzeł o wartości czytanej z pliku już istnieje

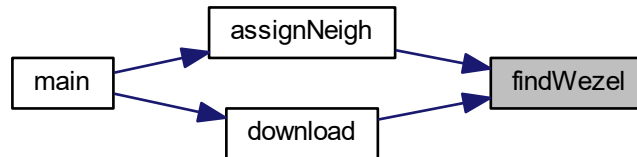
##### Parameters

<i>pHead</i>	adres pierwszego elementu listy węzłów
<i>szukana</i>	wartość szukanego węzła

**Returns**

adres węzła o szukanej wartości

Here is the caller graph for this function:

**4.2.1.11 ifPossible()**

```

bool ifPossible (
    krawedz * pHead,
    kolor current )
  
```

Funkcja sprawdza, czy dla danego zakolorowanie węzła i jego sąsiadów może wystąpić dwudzielność całego grafu

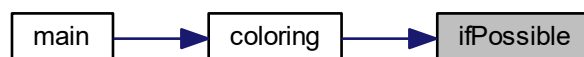
**Parameters**

<i>pHead</i>	adres pierwszego elementu listy krawędzi
<i>current</i>	kolor węzła, dla którego wykonywana jest funkcja

**Returns**

w wypadku nie przeczącym dwudzielności zwraca TRUE, w przeciwnym wypadku FALSE

Here is the caller graph for this function:





#### 4.2.1.12 makeColors()

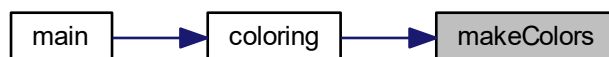
```
void makeColors (
    krawedz * pHead,
    kolor current )
```

Funkcja nadaje kolory sąsiednim węzłom

##### Parameters

<i>pHead</i>	adres pierwszego elementu listy krawędzi
<i>current</i>	kolor węzła, dla którego wykonywana jest funkcja

Here is the caller graph for this function:



#### 4.2.1.13 printByColor()

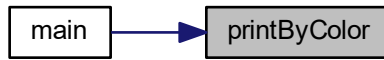
```
void printByColor (
    wezel * pHead,
    string output )
```

Funkcje, w wypadku dwudzielności grafu, wypisuje do pliku informację o dwudzielności oraz wartości węzłów dla obu grup

##### Parameters

<i>pHead</i>	adres pierwszego elementu listy węzłów
<i>output</i>	nazwa pliku wyjściowego

Here is the caller graph for this function:



#### 4.2.1.14 printGraph()

```
void printGraph (
    wezel * pHead,
    string output )
```

Funkcja wypisuje do pliku graf (węzły oraz ich sąsiadów)

##### Parameters

<i>pHead</i>	adres pierwszego elementu listy węzłów
<i>nazwa</i>	pliku wyjściowego

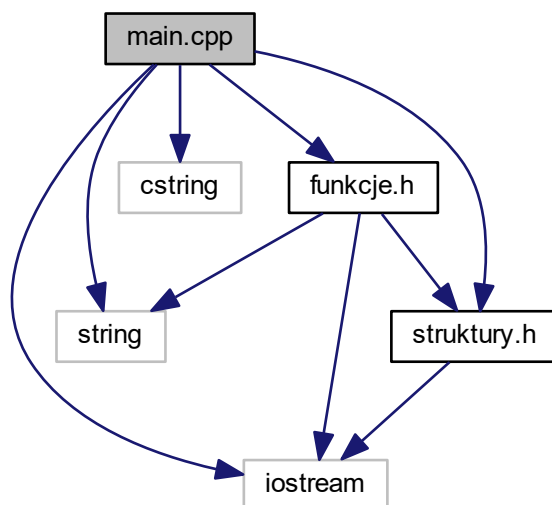
Here is the caller graph for this function:



## 4.3 main.cpp File Reference

```
#include <iostream>
#include <string>
#include <cstring>
#include "funkcje.h"
#include "struktury.h"
```

Include dependency graph for main.cpp:



## Functions

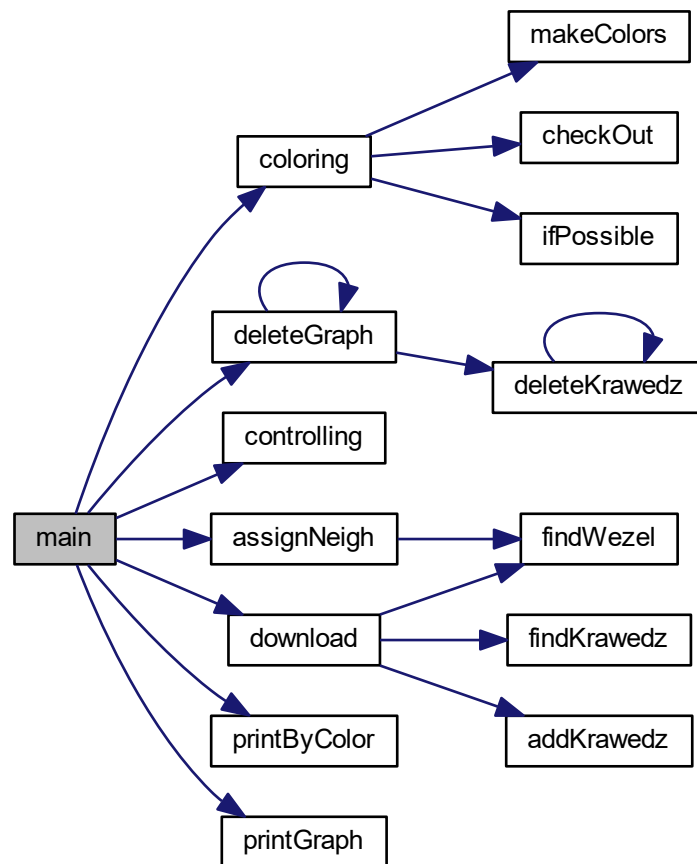
- `int main (int argc, char *argv[])`

### 4.3.1 Function Documentation

#### 4.3.1.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

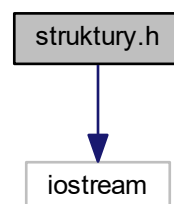
Here is the call graph for this function:



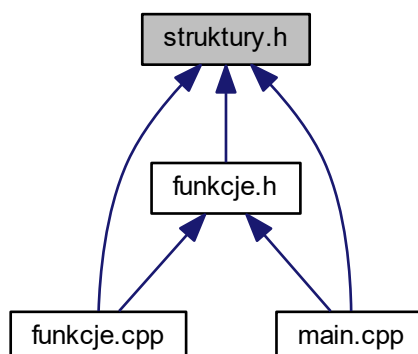
## 4.4 struktury.h File Reference

```
#include <iostream>
```

Include dependency graph for `struktury.h`:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [wezel](#)
- struct [krawedz](#)

## Enumerations

- enum [kolor](#) { [white](#), [red](#), [green](#) }

### 4.4.1 Enumeration Type Documentation

#### 4.4.1.1 kolor

```
enum kolor
```

Typ reprezentujący grupę dwudzielności, do której należą poszczególne węzły grafu.

##### Enumerator

<a href="#">white</a>	grupa neutralna, przydzielana od razu po utworzeniu węzła
<a href="#">red</a>	grupa "czerwona"
<a href="#">green</a>	grupa "zielona"



# Index

- addKrawedz
  - funkcje.cpp, [10](#)
  - funkcje.h, [22](#)
- assignNeigh
  - funkcje.cpp, [10](#)
  - funkcje.h, [23](#)
- checkOut
  - funkcje.cpp, [12](#)
  - funkcje.h, [24](#)
- coloring
  - funkcje.cpp, [13](#)
  - funkcje.h, [24](#)
- controlling
  - funkcje.cpp, [14](#)
  - funkcje.h, [25](#)
- deleteGraph
  - funkcje.cpp, [14](#)
  - funkcje.h, [26](#)
- deleteKrawedz
  - funkcje.cpp, [15](#)
  - funkcje.h, [27](#)
- done
  - wezel, [7](#)
- download
  - funkcje.cpp, [16](#)
  - funkcje.h, [28](#)
- findKrawedz
  - funkcje.cpp, [17](#)
  - funkcje.h, [28](#)
- findWezel
  - funkcje.cpp, [18](#)
  - funkcje.h, [29](#)
- funkcje.cpp, [9](#)
  - addKrawedz, [10](#)
  - assignNeigh, [10](#)
  - checkOut, [12](#)
  - coloring, [13](#)
  - controlling, [14](#)
  - deleteGraph, [14](#)
  - deleteKrawedz, [15](#)
  - download, [16](#)
  - findKrawedz, [17](#)
  - findWezel, [18](#)
  - ifPossible, [18](#)
  - makeColors, [19](#)
  - printByColor, [20](#)
  - printGraph, [20](#)
- funkcje.h, [21](#)
  - addKrawedz, [22](#)
  - assignNeigh, [23](#)
  - checkOut, [24](#)
  - coloring, [24](#)
  - controlling, [25](#)
  - deleteGraph, [26](#)
  - deleteKrawedz, [27](#)
  - download, [28](#)
  - findKrawedz, [28](#)
  - findWezel, [29](#)
  - ifPossible, [30](#)
  - makeColors, [30](#)
  - printByColor, [31](#)
  - printGraph, [32](#)
- green
  - struktury.h, [35](#)
- grupa
  - wezel, [7](#)
- ifPossible
  - funkcje.cpp, [18](#)
  - funkcje.h, [30](#)
- kolor
  - struktury.h, [35](#)
- krawedz, [5](#)
  - pNeighbour, [6](#)
  - pNext, [6](#)
  - wyjście, [6](#)
- main
  - main.cpp, [33](#)
- main.cpp, [32](#)
  - main, [33](#)
- makeColors
  - funkcje.cpp, [19](#)
  - funkcje.h, [30](#)
- pNeighbour
  - krawedz, [6](#)
  - wezel, [7](#)
- pNext
  - krawedz, [6](#)
  - wezel, [7](#)
- printByColor
  - funkcje.cpp, [20](#)
  - funkcje.h, [31](#)
- printGraph
  - funkcje.cpp, [20](#)

- funkcje.h, [32](#)
- red
  - struktury.h, [35](#)
- struktury.h, [34](#)
  - green, [35](#)
  - kolor, [35](#)
  - red, [35](#)
  - white, [35](#)
- wartosc
  - wezel, [8](#)
- wezel, [6](#)
  - done, [7](#)
  - grupa, [7](#)
  - pNeighbour, [7](#)
  - pNext, [7](#)
  - wartosc, [8](#)
- white
  - struktury.h, [35](#)
- wyjscie
  - krawedz, [6](#)