DrzewaBST

2

Wygenerowano za pomocą Doxygen 1.12.0

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdujÄ siÄ klasy, struktury, unie i interfejsy wraz z ich krÃ^{3}tkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

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

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja klasy BST

Klasa implementuj¹ca binarne drzewo poszukiwañ (BST).

`#include <BST.h>`

Komponenty

- class Node

  Wêze[3] *drzewa BST przechowuj¹cy wartoœæ oraz wskaniki do lewego i prawego dziecka.*

Metody publiczne

- BST ()
- ∼BST ()
- void add (int value)
- void remove (int value)
- void clear ()
- void searchPath (int value)
- void displayInorder ()
- void displayPreorder ()
- void displayPostorder ()
- void saveToFile ()

Metody prywatne

- void addNode (Node ∗&node, int value)

  Dodanie nowego wêz[3]*a do drzewa.*
- Node ∗ deleteNode (Node ∗node, int value)

  Usuniecie wezla z dana wartoscia.
- void deleteTree (Node ∗node)

  Usuniecie calego drzewa.
- void inorder (Node ∗node)

  Wypisanie drzewa inorder.

- void preorder (Node *node)
  Wypisanie drzewa preorder.
- void postorder (Node *node)
  Wypisanie drzewa postorder.
- void findPath (Node *node, int value, std::string path)
  Znalezienie cie¿ki do danego wêz^{3}a.
- void saveInOrder (Node *node, std::ofstream &outFile)
  Zapisanie drzewa do pliku in order.

Atrybuty prywatne

- Node * root

Przyjaciele

- class Files

3.1.1 Opis szczegÃ^{3}Åowy

Klasa implementuj¹ca binarne drzewo poszukiwañ (BST).

Definicja w linii 9 pliku BST.h.

3.1.2 Dokumentacja konstruktora i destruktora

3.1.2.1 BST()

```
BST::BST ()
```

Definicja w linii 3 pliku DrzewaBST.cpp.

3.1.2.2 ~BST()

```
BST::~BST ()
```

Definicja w linii 5 pliku DrzewaBST.cpp.

3.1.3 Dokumentacja funkcji skÅadowych

3.1.3.1 add()

```
void BST::add (
            int value)
```

Definicja w linii 16 pliku DrzewaBST.cpp.

3.1.3.2 addNode()

```
void BST::addNode (
            Node *& node,
            int value)  [private]
```

Dodanie nowego wêz^{3}a do drzewa.

Parametry

value	Wartosc nowego wezla
node	referencja do wskanika wêz^{3}a

Definicja w linii 9 pliku DrzewaBST.cpp.

3.1.3.3 clear()

```
void BST::clear ()
```

Definicja w linii 59 pliku DrzewaBST.cpp.

3.1.3.4 deleteNode()

```
BST::Node * BST::deleteNode (
            Node * node,
            int value)  [private]
```

Usuniecie wezla z dana wartoscia.

Parametry

value	wartosc wezla ktory ma byc usuniety
node	wskanika wêz^{3}a od ktorego szukamy

Definicja w linii 20 pliku DrzewaBST.cpp.

3.1.3.5 deleteTree()

```
void BST::deleteTree (
            Node * node)  [private]
```

Usuniecie calego drzewa.

Parametry

node	wskanika wêz^{3}a od ktorego usuwamy

Definicja w linii 50 pliku DrzewaBST.cpp.

3.1.3.6 displayInorder()

```
void BST::displayInorder ()
```

Definicja w linii 90 pliku DrzewaBST.cpp.

3.1.3.7 displayPostorder()

```
void BST::displayPostorder ()
```

Definicja w linii 116 pliku DrzewaBST.cpp.

3.1.3.8 displayPreorder()

```
void BST::displayPreorder ()
```

Definicja w linii 103 pliku DrzewaBST.cpp.

3.1.3.9 findPath()

```
void BST::findPath (
            Node * node,
            int value,
            std::string path = "")  [private]
```

Znalezienie cie¿ki do danego wêz^{3}a.

Parametry

node	wskanika wêz^{3}a w tym przypadku root
value	wartosc wezla do ktorego chcemy znalezc droge
path	^{3}añcuch znaków który przechowuje cie¿kê

Definicja w linii 64 pliku DrzewaBST.cpp.

3.1.3.10 inorder()

```
void BST::inorder (
            Node * node)  [private]
```

Wypisanie drzewa inorder.

Parametry

node	wskanika wêz^{3}a w tym przypadku root

Definicja w linii 82 pliku DrzewaBST.cpp.

3.1.3.11 postorder()

```
void BST::postorder (
            Node * node)  [private]
```

Wypisanie drzewa postorder.

Parametry

node	wskanika wêz[3]a w tym przypadku root

Definicja w linii 108 pliku DrzewaBST.cpp.

3.1.3.12 preorder()

```
void BST::preorder (
            Node * node)  [private]
```

Wypisanie drzewa preorder.

Parametry

node	wskanika wêz[3]a w tym przypadku root

Definicja w linii 95 pliku DrzewaBST.cpp.

3.1.3.13 remove()

```
void BST::remove (
            int value)
```

Definicja w linii 46 pliku DrzewaBST.cpp.

3.1.3.14 saveInOrder()

```
void BST::saveInOrder (
            Node * node,
            std::ofstream & outFile)  [private]
```

Zapisanie drzewa do pliku in order.

Parametry

node	wskanik na wêze[3], od którego zaczynamy zapisywanie
outfile	referencja do strumienia wyjsciowego pliku ktory jest otwarty

Definicja w linii 133 pliku DrzewaBST.cpp.

3.1.3.15 saveToFile()

```
void BST::saveToFile ()
```

Definicja w linii 121 pliku DrzewaBST.cpp.

3.1.3.16 searchPath()

```
void BST::searchPath (
            int value)
```

Definicja w linii 78 pliku DrzewaBST.cpp.

3.1.4 Dokumentacja przyjaciÃ³Å i powiÄzanych symboli

3.1.4.1 Files

```
friend class Files  [friend]
```

Definicja w linii 82 pliku BST.h.

3.1.5 Dokumentacja atrybutÃ³w skÅadowych

3.1.5.1 root

```
Node* BST::root  [private]
```

Definicja w linii 22 pliku BST.h.

Dokumentacja dla tej klasy zostaÅa wygenerowana z plikÃ³w:

- DrzewaBST/BST.h
- DrzewaBST/DrzewaBST.cpp

3.2 Dokumentacja klasy Files

Klasa statyczna do obsÅugi operacji plikowych na drzewie BST.

```
#include <BSTfiles.h>
```

Statyczne metody publiczne

- static void Load_from_text_file (BST &tree, const std::string &filename, bool clearTree)

  Wczytuje drzewo BST z pliku tekstowego.
- static void Save_to_text_file (std::ofstream &file, BST::Node *node)

  Zapisuje drzewo BST do pliku tekstowego w kolejnoci in-order.
- static void Save_to_binary_file (BST &tree, const std::string &filename)

  Zapisuje drzewo BST do pliku binarnego.
- static void Load_from_binary_file (BST &tree, const std::string &filename)

  Wczytuje drzewo BST z pliku binarnego.

3.2.1 Opis szczegÃ³Åowy

Klasa statyczna do obs³ugi operacji plikowych na drzewie BST.

Klasa Files udostêpnia metody do zapisu i odczytu drzewa BST z plików tekstowych i binarnych. Wszystkie metody s¹ statyczne i operuj¹ na podanym drzewie BST.

Definicja w linii 12 pliku BSTfiles.h.

3.2.2 Dokumentacja funkcji skÅadowych

3.2.2.1 Load_from_binary_file()

```
void Files::Load_from_binary_file (
            BST & tree,
            const std::string & filename)  [static]
```

Wczytuje drzewo BST z pliku binarnego.

Metoda odczytuje dane z pliku binarnego i odbudowuje drzewo BST w tej samej strukturze.

Parametry

tree	Referencja do drzewa BST, do którego zostan¹ dodane wczytane wartoci.
filename	Nazwa pliku binarnego z zapisanym drzewem BST.

Definicja w linii 54 pliku BSTfiles.cpp.

3.2.2.2 Load_from_text_file()

```
void Files::Load_from_text_file (
            BST & tree,
            const std::string & filename,
            bool clearTree)  [static]
```

Wczytuje drzewo BST z pliku tekstowego.

Metoda odczytuje wartoci z podanego pliku tekstowego i dodaje je do drzewa BST. Jeli parametr `clearTree` jest ustawiony na `true`, metoda wyczyci drzewo przed wczytaniem nowych danych.

Parametry

tree	Referencja do drzewa BST, do którego zostan¹ dodane wczytane wartoci.
filename	Nazwa pliku tekstowego z liczbami.
clearTree	Flaga okrelaj¹ca, czy drzewo ma zostaæ wyczyszczone przed wczytaniem danych.

Definicja w linii 4 pliku BSTfiles.cpp.

3.2.2.3 Save_to_binary_file()

```
void Files::Save_to_binary_file (
            BST & tree,
            const std::string & filename)  [static]
```

Zapisuje drzewo BST do pliku binarnego.

Metoda zapisuje strukturê drzewa BST do pliku binarnego, umo¿liwiaj¹c póniejsze wczytanie drzewa w tej samej strukturze.

Parametry

tree	Referencja do drzewa BST, które ma zostaæ zapisane do pliku.
filename	Nazwa pliku binarnego, do którego zostanie zapisane drzewo.

Definicja w linii 37 pliku BSTfiles.cpp.

3.2.2.4 Save_to_text_file()

```
void Files::Save_to_text_file (
            std::ofstream & file,
            BST::Node * node)  [static]
```

Zapisuje drzewo BST do pliku tekstowego w kolejnoci in-order.

Metoda zapisuje wartoci wêz^{3}ów drzewa BST do pliku tekstowego w kolejnoci in-order.

Parametry

file	Obiekt pliku wyjciowego, do którego zapisywane s¹ wartoci.
node	Wskanik do bie¿¹cego wêz^{3}a BST do zapisania.

Definicja w linii 27 pliku BSTfiles.cpp.

Dokumentacja dla tej klasy zostaÅa wygenerowana z plikÃ^{3}w:

- DrzewaBST/BSTfiles.h
- DrzewaBST/BSTfiles.cpp

3.3 Dokumentacja klasy BST::Node

Wêze^3 drzewa BST przechowuj¹cy wartoæ oraz wskaniki do lewego i prawego dziecka.

Metody publiczne

- Node (int value)

Atrybuty publiczne

- int data
- Node * left
- Node * right

3.3.1 Opis szczegÃ^{3}Åowy

Wêze^3 drzewa BST przechowuj¹cy wartoæ oraz wskaniki do lewego i prawego dziecka.

Definicja w linii 15 pliku BST.h.

3.3.2 Dokumentacja konstruktora i destruktora

3.3.2.1 Node()

```
BST::Node::Node (
            int value)  [inline]
```

Definicja w linii 20 pliku BST.h.

3.3.3 Dokumentacja atrybutÃ³w skÅadowych

3.3.3.1 data

```
int BST::Node::data
```

Definicja w linii 17 pliku BST.h.

3.3.3.2 left

```
Node* BST::Node::left
```

Definicja w linii 18 pliku BST.h.

3.3.3.3 right

```
Node* BST::Node::right
```

Definicja w linii 19 pliku BST.h.

Dokumentacja dla tej klasy zostaÅa wygenerowana z pliku:

- DrzewaBST/BST.h

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku DrzewaBST/BST.h

```
#include <iostream>
#include <fstream>
#include <string>
```

Komponenty

- class BST

  Klasa implementuj¹ca binarne drzewo poszukiwañ (BST).
- class BST::Node

  Wêze[3] *drzewa BST przechowuj¹cy wartoœæ oraz wskaniki do lewego i prawego dziecka.*

4.2 BST.h

IdÅº do dokumentacji tego pliku.

```
00001 #pragma once
00002 #include <iostream>
00003 #include <fstream>
00004 #include <string>
00009 class BST {
00010 private:
00015     class Node {
00016     public:
00017         int data; // <--- Wartoœæ przechowywane w wêle
00018         Node* left;  // <--- Wskanik na lewy korzen
00019         Node* right;  // <--- Wskanik na prawy korzen
00020         Node(int value) : data(value), left(nullptr), right(nullptr) {}  // <--- Konstruktor tworz¹cy
      wezel z podana wartoscia
00021     };
00022     Node* root; // <--- Wskanik na korzen drzewa
00028     void addNode(Node*& node, int value);
00034     Node* deleteNode(Node* node, int value);
00039     void deleteTree(Node* node);
00044     void inorder(Node* node);
00049     void preorder(Node* node);
00054     void postorder(Node* node);
00061     void findPath(Node* node, int value, std::string path);
00067     void saveInOrder(Node* node, std::ofstream& outFile);
00068
00069 public:
00070
00071     BST(); // <--- Konstruktor drzewa
```

```
00072     ~BST(); // <--- Destruktor drzewa drzewa
00073     void add(int value); // <--- Wywo3anie poprzez referencje fukcji addNode
00074     void remove(int value); // <--- Wywo3anie poprzez referencje fukcji deleteNode
00075     void clear(); // <--- Wywo3anie poprzez referencje fukcji deleteTree
00076     void searchPath(int value); // <--- Wywo3anie poprzez referencje fukcji findPath
00077     void displayInorder(); // <--- Wywo3anie poprzez referencje fukcji Inorder
00078     void displayPreorder(); // <--- Wywo3anie poprzez referencje fukcji Preorder
00079     void displayPostorder(); // <--- Wywo3anie poprzez referencje fukcji Postorder
00080     void saveToFile(); // <--- Wywo3anie poprzez referencje fukcji saveInOrder
00081
00082     friend class Files; // <--- ustawienie friend class ¿eby klasa Files ¿eby metody tej klasy mog3y
      korzystaæ z prywatnej klasy node
00083 };
```

4.3 Dokumentacja pliku DrzewaBST/BSTfiles.cpp

```
#include "BSTfiles.h"
#include <fstream>
```

4.4 BSTfiles.cpp

IdÅº do dokumentacji tego pliku.

```
00001 #include "BSTfiles.h"
00002 #include<fstream>
00003
00004 void Files::Load_from_text_file(BST& tree, const std::string& filename, bool clearTree) {
00005
00006     int value;  // <--- Zmienna przechowuj¹ca wartoæ odczytan¹ z pliku
00007
00008     std::ifstream file(filename); // <---  Otwarcie podanego pliku do odczytu
00009
00010     if (!file) {  // <--- Sprawdzanie czy plik zosta3 poprawnie otwarty
00011
00012         std::cout « "Nie mozna otworzyc pliku tekstowego: " « filename « std::endl;
00013         return;
00014
00015     }
00016
00017     while (file » value) { // <--- Pêtla odczytuj¹ca liczby z pliku
00018
00019         tree.add(value);  // <--- Przekazywanie liczb do drzewa
00020
00021     }
00022
00023     file.close(); // <--- Zamykanie pliku
00024
00025 }
00026
00027 void Files::Save_to_text_file(std::ofstream& file, BST::Node* node) {
00028     if (node == nullptr) {  // <--- Sprawdzanie czy istnieje wêze3
00029         return;
00030     }
00031
00032     file « node->data « " ";  // <--- Zapisywanie wartoci bie¿¹cego wêz3a
00033     Save_to_text_file(file, node->left);  // <--- Wywo3anie funkcji dla lewego korzenia
00034     Save_to_text_file(file, node->right); // <--- Wywo3anie funkcji dla prawego korzenia
00035 }
00036
00037 void Files::Save_to_binary_file(BST& tree, const std::string& filename) {
00038
00039     std::ofstream file(filename, std::ios::binary); // <--- Otwiera plik do zapisu w trybie binarnym
00040
00041     if (!file) { // <--- Sprawdzanie czy plik zosta3 poprawnie otwarty
00042
00043         std::cout « "Nie mo¿na otworzyæ pliku do zapisu: " « filename « std::endl;
00044         return;
00045
00046     }
00047
00048     Save_to_text_file(file, tree.root); // <--- Wywo3uje funkcjê aby zapisaæ wartoci
00049
00050     file.close(); // <--- Zamykanie pliku
00051
```

```
00052 }
00053
00054 void Files::Load_from_binary_file(BST& tree, const std::string& filename) {
00055
00056     std::ifstream file(filename, std::ios::binary); // <---  Otwarcie podanego pliku do odczytu
00057
00058     if (!file) { // <--- Sprawdzanie czy plik zosta³ poprawnie otwarty
00059         std::cout « "Nie mo¿na otworzyæ pliku do wczytania: " « filename « std::endl;
00060         return;
00061     }
00062
00063     int value;
00064
00065     while (file.read(reinterpret_cast<char*>(&value), sizeof(value))) { // <--- Odczytywanie funkcj¹
      read w pêtli while i zapisanie do zmiennej value. Rzutuje wskanik do value na char* poniewa¿ funkcja
      read tego wymaga
00066         tree.add(value); // <--- Dodanie wartosci do drzewa
00067     }
00068
00069     file.close();// <--- Zamykanie pliku
00070
00071 }
```

4.5 Dokumentacja pliku DrzewaBST/BSTfiles.h

```
#include <iostream>
#include <fstream>
#include "BST.h"
```

Komponenty

- class Files

  Klasa statyczna do obs³ugi operacji plikowych na drzewie BST.

4.6 BSTfiles.h

Idź do dokumentacji tego pliku.

```
00001 #pragma once
00002 #include <iostream>
00003 #include <fstream>
00004 #include "BST.h"
00012 class Files {
00013 public:
00024         static void Load_from_text_file(BST& tree, const std::string& filename, bool clearTree);
00033         static void Save_to_text_file(std::ofstream& file,BST::Node* node);
00043         static void Save_to_binary_file(BST& tree, const std::string& filename);
00052         static void Load_from_binary_file(BST& tree, const std::string& filename);
00053     };
```

4.7 Dokumentacja pliku DrzewaBST/DrzewaBST.cpp

```
#include "BST.h"
```

4.8 DrzewaBST.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "BST.h"
00002
00003 BST::BST() : root(nullptr) {} // <--- Przekazywanie nullptr do root oznacza że tworzymy drzewo ale
      jest puste
00004
00005 BST::~BST() {
00006     this->clear(); // <--- Destruktor wywołanie funkcji clear która usuwa drzewo
00007 }
00008
00009 void BST::addNode(Node*& node, int value) {
00010      if (!node) { node = new Node(value); } // <--- Sprawdzanie czy istnieje jakikolwiek węzeł jeżeli
      nie isntieje to tworzenie nowego węzła z wartością value
00011      else if (value < node->data) { addNode(node->left, value); } // <--- jeżeli wartość nowego wezła
      jest mniejsza od wartosci obecnego to wywoływane jest znowu ta funkcja ale od lewego dziecka węzła
00012      else { addNode(node->right, value); } // <--- jeżeli wartość nowego wezła jest większa od
      wartosci obecnego to wywoływane jest znowu ta funkcja ale od prawego dziecka węzła
00013  }
00014
00015
00016 void BST::add(int value) {
00017         addNode(root, value); // <--- Referencyjne wywołane addNode
00018 }
00019
00020 BST::Node* BST::deleteNode(Node* node, int value) {
00021         if (!node) return nullptr;  // <--- Sprawdzanie czy istnieje jakikolwiek węzeł jeżeli nie to
      zwracany jest nullptr
00022         if (value < node->data) { node->left = deleteNode(node->left, value); } // <--- jeżeli wartość
      nowego wezła jest mniejsza od root to wywoływane jest znowu ta funkcja ale od lewego dziecka węzła
00023         else if (value > node->data) { node->right = deleteNode(node->right, value); }// <--- jeżeli
      wartość nowego wezła jest większa od root to wywoływane jest znowu ta funkcja ale od prawego dziecka
      węzła
00024         else {
00025             if (!node->left) { // <--- Jeżeli węzeł nie ma lewego dziecka, zastępujemy go jego prawym
      dzieckiem.
00026                 Node* temp = node->right;
00027                 delete node;
00028                 return temp;
00029             }
00030             else if (!node->right) { // <--- Jeżeli węzeł nie ma prawego dziecka, zastępujemy go jego
      lewym dzieckiem.
00031                 Node* temp = node->left;
00032                 delete node;
00033                 return temp;
00034             }
00035             else {
00036                 Node* temp = node->right;
00037                 while (temp->left) temp = temp->left; // <--- Znajdujemy najmniejszy węzeł w prawym
      poddrzewie
00038                 node->data = temp->data;  // <--- Zastępujemy dane bieżącego węzła danymi węzła
      najmniejszego
00039                 node->right = deleteNode(node->right, temp->data);  // <--- Rekurencyjnie usuwamy
      najmniejszy węzeł prawego poddrzewia
00040             }
00041         }
00042         return node;  // <--- zwracamy węzeł
00043 }
00044
00045
00046 void BST::remove(int value) {
00047     root = deleteNode(root, value);  // <--- Referencyjne wywołanie deleteNode
00048 }
00049
00050 void BST::deleteTree(Node* node) {
00051     if (node) {
00052         deleteTree(node->left);    // <--- Rekurencyjne wywołanie deleteTree w lewym dziecku
00053         deleteTree(node->right);   // <--- Rekurencyjne wywołanie deleteTree w prawym dziecku
00054         delete node;
00055     }
00056
00057 }
00058
00059 void BST::clear() {
00060     deleteTree(root);  // <--- Referencyjne wywołanie deleteTree
00061     root = nullptr;
00062 }
00063
00064 void BST::findPath(Node* node, int value, std::string path = ""){
00065     if (!node) {   // <--- Sprawdzanie czy isntieje jakikolwiek węzeł w drzewie
00066         std::cout « "Brak elementu w drzewie." « std::endl;
00067         return;
00068     }
00069     path += std::to_string(node->data) + " ";   // <--- Dodanie do łańcucha znaków wartości obecnego
      węzła i spacji
```

```
00070     if (node->data == value) { // <--- Jeżeli trafiliśmy na dobry węzeł wypisywana jest wartość do
      niego
00071         std::cout « "Sciezka do " « value « " : " « path « std::endl;
00072         return;
00073     }
00074     if (value < node->data) { findPath(node->left, value, path); } // <--- Rekurencyjne wywołanie tej
      samej funkcji ale od lewego dziecka
00075     else { findPath(node->right, value, path); } // <--- Rekurencyjne wywołanie tej samej funkcji ale
      od prawego dziecka
00076 }
00077
00078 void BST::searchPath(int value) {
00079     findPath(root, value);   // <--- Referencyjne wywołanie findPath
00080 }
00081
00082 void BST::inorder(Node* node) {
00083     if (node) { // <--- Jeżeli isntieje jakikolwiek węzeł w drzewie
00084         inorder(node->left); // <--- Rekurencyjne wywołanie tej samej funkcji ale od lewego dziecka
00085         std::cout « node->data « " "; // <--- wypisanie wartości obecnego węzła
00086         inorder(node->right); // <--- Rekurencyjne wywołanie tej samej funkcji ale od prawego dziecka
00087     }
00088 }
00089
00090 void BST::displayInorder() {
00091     inorder(root);   // <--- Referencyjne wywołanie inorder
00092     std::cout « std::endl;
00093 }
00094
00095 void BST::preorder(Node* node) {
00096     if (node) { // <--- Jeżeli isntieje jakikolwiek węzeł w drzewie
00097         std::cout « node->data « " ";// <--- wypisanie wartości obecnego węzła
00098         preorder(node->left); // <--- Rekurencyjne wywołanie tej samej funkcji ale od lewego dziecka
00099         preorder(node->right); // <--- Rekurencyjne wywołanie tej samej funkcji ale od prawego dziecka
00100     }
00101 }
00102
00103 void BST::displayPreorder() {
00104     preorder(root);   // <--- Referencyjne wywołanie preorder
00105     std::cout « std::endl;
00106 }
00107
00108 void BST::postorder(Node* node) {
00109     if (node) {
00110         postorder(node->left); // <--- Rekurencyjne wywołanie tej samej funkcji ale od lewego dziecka
00111         postorder(node->right);// <--- Rekurencyjne wywołanie tej samej funkcji ale od prawego dziecka
00112         std::cout « node->data « " ";// <--- wypisanie wartości obecnego węzła
00113     }
00114 }
00115
00116 void BST::displayPostorder() {
00117     postorder(root);  // <--- Referencyjne wywołanie postorder
00118     std::cout « std::endl;
00119 }
00120
00121 void BST::saveToFile() {
00122     std::ofstream outFile("plik.txt"); // <--- Otwarcie pliku
00123     if (outFile.is_open()) { // <--- sprawdzanie czy plik się otwarł poprawnie
00124         saveInOrder(root, outFile); // <--- Referencyjne wywołanie saveInorder
00125         outFile.close(); // <--- Zamknięcie pliku
00126         std::cout « "Drzewo zapisane w plik.txt" « std::endl;
00127     }
00128     else { // <--- przypadek gdy plik się nie otwarł
00129         std::cerr « "Brak mozliwosci otwarcia pliku." « std::endl;
00130     }
00131 }
00132
00133 void BST::saveInOrder(Node* node, std::ofstream& outFile) {
00134     if (node) {
00135         saveInOrder(node->left, outFile);// <--- Rekurencyjne wywołanie tej samej funkcji ale od
      lewego dziecka
00136         outFile « node->data « " "; // <--- zapisanie wartości obecnego węzła do pliku
00137         saveInOrder(node->right, outFile);// <--- Rekurencyjne wywołanie tej samej funkcji ale od
      prawego dziecka
00138     }
00139 }
```

4.9 Dokumentacja pliku DrzewaBST/main.cpp

```
#include <iostream>
#include "BST.h"
```

```
#include "BSTfiles.h"
```

Funkcje

- void displayMenu ()
- int main ()

4.9.1 Dokumentacja funkcji

4.9.1.1 displayMenu()

```
void displayMenu ()
```

Definicja w linii 5 pliku main.cpp.

4.9.1.2 main()

```
int main ()
```

Definicja w linii 21 pliku main.cpp.

4.10 main.cpp

Idź do dokumentacji tego pliku.

```
00001 #include <iostream>
00002 #include "BST.h"
00003 #include "BSTfiles.h"
00004
00005 void displayMenu() {
00006     std::cout « "\nBST Menu:\n";
00007     std::cout « "1. Dodaj element\n";
00008     std::cout « "2. Usuń element\n";
00009     std::cout « "3. Wyświetl drzewo preorder\n";
00010     std::cout « "4. Wyświetl drzewo inorder\n";
00011     std::cout « "5. Wyświetl drzewo postorder\n";
00012     std::cout « "6. Szukaj drogi do elementu\n";
00013     std::cout « "7. Usuń całe drzewo\n";
00014     std::cout « "8. Zapisz drzewo do pliku\n";
00015     std::cout « "9. Wczytaj drzewo z pliku tekstowego\n";
00016     std::cout « "10. Zapisz drzewo z pliku binarnego\n";
00017     std::cout « "11. Wczytaj drzewo z pliku binarnego\n";
00018     std::cout « "12. Wyjdź\n";
00019 }
00020
00021 int main() {
00022
00023     char choice = 'n';
00024     int option;
00025     int value;
00026     BST tree;
00027
00028     Files file;
00029
00030     do {
00031         displayMenu();
00032         std::cin » option;
00033
00034     switch (option) {
00035     case 1:
00036         do {
00037             std::cout « "Podaj wartosc nowego elementu drzewa BST: ";
00038             std::cin » value;
```

```
00039
00040             tree.add(value);
00041
00042             std::cout « "Czy dodac nastepny element ? (t/n): ";
00043             std::cin » choice;
00044
00045         } while (choice == 't' || choice == 'T');
00046         break;
00047     case 2:
00048
00049         std::cout « "Podaj wartosc elementu drzewa BST, ktora chcesz usunac: ";
00050         std::cin » value;
00051
00052         tree.remove(value);
00053
00054         break;
00055     case 3:
00056         tree.displayPreorder();
00057         std::cout « std::endl;
00058         break;
00059
00060     case 4:
00061         tree.displayInorder();
00062         std::cout « std::endl;
00063         break;
00064
00065     case 5:
00066         tree.displayPostorder();
00067         std::cout « std::endl;
00068         break;
00069     case 6:
00070         do {
00071             std::cout « "Podaj wartosc elementu drzewa BST, ktora chcesz znalezc: ";
00072             std::cin » value;
00073
00074             tree.searchPath(value);
00075
00076             std::cout « "Czy chcesz znalezc inny element ? (t/n): ";
00077             std::cin » choice;
00078
00079         } while (choice == 't' || choice == 'T');
00080         break;
00081     case 7:
00082         tree.clear();
00083         std::cout « "Cale drzewo zostalo usuniete" « std::endl;
00084         break;
00085
00086     case 8:
00087         tree.saveToFile();
00088         std::cout « "Drzewo zapisano do plik.txt" « std::endl;
00089         break;
00090
00091     case 9:
00092     {
00093         std::string filename = "Wczytanie z pliku tekstowego";
00094         file.Load_from_text_file(tree, filename, true);
00095     }
00096     break;
00097
00098
00099     case 10:
00100     {
00101         std::string binFilename = "tree.bin";
00102         file.Save_to_binary_file(tree, binFilename);
00103         std::cout « "Konczenie programu..." « std::endl;
00104     }
00105     break;
00106
00107     case 11:
00108     {
00109         std::string binFilename = "tree.bin";
00110         file.Load_from_binary_file(tree, binFilename);
00111         std::cout « "Wczytano tree z pliku binarnego: " « binFilename « std::endl;
00112     }
00113     break;
00114
00115     }
00116
00117 } while (option != 12);
00118
00119 }
```

Skorowidz