Sprawozdanie 07: Implementacja i zastosowanie drzew dowolnego rzędu

Dmytro Sakharskyi - L02 - 31259 16.05.2025

Metody Programowania

Informatyka 1 rok. 2 semestr

Wstęp

Drzewa dowolnego rzędu są strukturą danych, w której każdy węzeł może posiadać dowolną liczbę dzieci. W przeciwieństwie do **drzew binarnych**, które **ograniczają** liczbę potomków **do dwóch**, drzewa n-ary pozwalają na bardziej elastyczne odwzorowywanie danych o złożonej strukturze, takich jak systemy plików, hierarchie organizacyjne czy składnie języków programowania.

Celem ćwiczenia było zapoznanie się z implementacją takiego drzewa w języku C++ z wykorzystaniem nowoczesnych technik programowania, takich jak wskaźniki inteligentne **(shared_ptr)**. Zadanie polegało na edycji i rozbudowie podanego kodu, a także na stworzeniu drzewa odwzorowującego rzeczywisty schemat organizacyjny wybranej instytucji. Dodatkowo, należało przetestować kod w praktyce, wyjaśnić jego działanie.

Implementacja programistyczna drzewa

Zadanie 1

Drzewo jest strukturą rekurencyjną, ponieważ każdy jego węzeł może zawierać w sobie inne węzły – czyli dzieci – które same również mogą mieć swoje dzieci, i tak dalej. Oznacza to, że struktura drzewa może być zdefiniowana **względem samej siebie**.

Można to zobaczyć w praktyce:

- Węzeł składa się z danych oraz listy dzieci,
- Każde dziecko jest również węzłem, który ma swoją własną listę dzieci,
- Aby przetworzyć (np. wyświetlić) całe drzewo, musimy rekurencyjnie odwiedzać każdy węzeł i jego dzieci.

Przykładowo, funkcja printTree jest funkcją rekurencyjną, bo dla każdego dziecka wywołuje samą siebie. Dzięki temu jesteśmy w stanie przejść przez wszystkie poziomy drzewa bez względu na jego głębokość czy rozmiar.

Zadanie 2

```cpp
#include <iostream>
#include <vector>

using namespace std;

struct Node
{
    string data;
    vector<shared_ptr<Node>> children;

    Node(const string& d) : data(d) {}

};

void addChild(shared_ptr<Node> parent, shared_ptr<Node> child){
    parent->children.push_back(child);
}
void printTree(const shared_ptr<Node>& node, int depth = 0) {
    for (int i = 0; i < depth; ++i) {
        cout << " ";
    }
    cout << node->data << "\n";
    for (const shared_ptr<Node>& child : node->children) {
        printTree(child, depth + 1);
    }
}
```

Struktura Node:
Reprezentuje jeden węzeł drzewa. Zawiera pole data typu **string** (przechowuje nazwę/etykietę węzła) oraz wektor wskaźników **children**, który przechowuje dzieci danego węzła.

Funkcja addChild:
Odpowiada za dodanie nowego dziecka do węzła. Po prostu dodaje wskaźnik **shared_ptr<Node>** do wektora **children**.

Funkcja printTree:
Wyświetla całe drzewo w sposób hierarchiczny, zachowując wcięcia, które zależą od głębokości aktualnie przetwarzanego węzła. Dzięki rekurencji odwiedza każde dziecko danego węzła, niezależnie od głębokości drzewa.

Funkcja main:
Tworzy kilka przykładowych węzłów, buduje między nimi relację rodzic–dziecko, a następnie wyświetla całą strukturę drzewa.

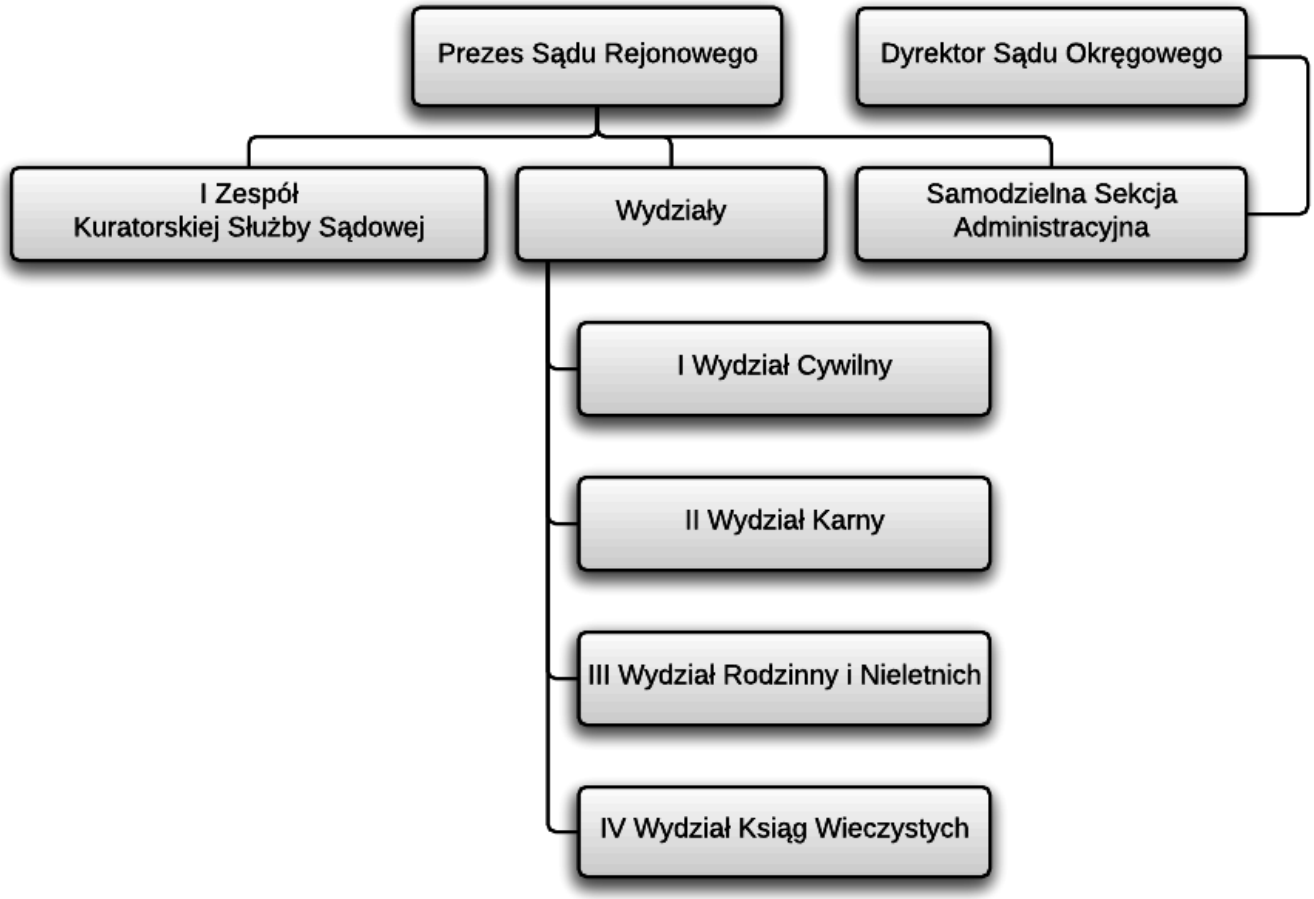
Wynik działania:

```
Prezes Sadu Rejonowego
 Samodzielna Sekcja Administracyjna
 I Zespol Kuratorskiej Sluzby Sadowej
Dyrektor Sadu Okregowego
 Samodzielna Sekcja Administracyjna
```

Działa poprawnie.

Zadanie 3

Struktura organizacyjna - Sąd rejonowy



https://www.strzyzow.sr.gov.pl/index.php/struktura-organizacyjna.html - link na stronę.

Zadanie 4

```cpp
int main() {
    shared_ptr<Node> root1 = make_shared<Node>("Prezes Sadu Rejonowego");
    shared_ptr<Node> root2 = make_shared<Node>("Dyrektor Sadu Okregowego");
    shared_ptr<Node> child1 = make_shared<Node>("I Zespol Kuratorskiej Sluzby Sadowej");
    shared_ptr<Node> child2 = make_shared<Node>("Wydzialy");
    shared_ptr<Node> child3 = make_shared<Node>("Samodzielna Sekcja Administracyjna");

    shared_ptr<Node> child4 = make_shared<Node>("I Wydzial Cywilny");
    shared_ptr<Node> child5 = make_shared<Node>("II Wydzial Karny");
    shared_ptr<Node> child6 = make_shared<Node>("III Wydzial Rodzinny i Nieletnich");
    shared_ptr<Node> child7 = make_shared<Node>("IV Wydzial Ksiag Wieczystych");


    addChild(root1, child3);
    addChild(root2, child3);

    addChild(root1, child1);
    addChild(root1, child2);

    addChild(child2, child4);
    addChild(child2, child5);
    addChild(child2, child6);
    addChild(child2, child7);

    printTree(root1);
    printTree(root2);

}
```

Struktura organizacyjna Sądu Rejonowego w postaci drzewa.

Cała struktura jest przykładem praktycznego zastosowania drzew dowolnego rzędu w modelowaniu organizacji.

Wynik działania:

```
Prezes Sadu Rejonowego
 Samodzielna Sekcja Administracyjna
 I Zespol Kuratorskiej Sluzby Sadowej
 Wydzialy
  I Wydzial Cywilny
  II Wydzial Karny
  III Wydzial Rodzinny i Nieletnich
  IV Wydzial Ksiag Wieczystych
Dyrektor Sadu Okregowego
 Samodzielna Sekcja Administracyjna
```

Wynik działania programu (drzewo wygenerowane przez funkcję printTree) zgadza się z oryginalnym schematem organizacyjnym przedstawionym na oryginalnej strukturze organizacyjnej sądu.

Elementy hierarchii, takie jak:

- **Prezes Sądu Rejonowego**,
- **Samodzielna Sekcja Administracyjna**,
- **Zespół Kuratorskiej Służby Sądowej**,
- **Wydziały** z odpowiednimi pododdziałami,
  zostały poprawnie odwzorowane w strukturze drzewa i wyświetlone zgodnie z ich pozycją w organizacji.

Potwierdza to, że implementacja działa zgodnie z założeniem i prawidłowo odwzorowuje rzeczywiste zależności organizacyjne.

Wszystkie zadania zostały wykonane zgodnie z instrukcją i bez żadnych problemów technicznych. Program działał poprawnie, a uzyskane wyniki były zgodne z oczekiwaniami.

Wnioski

- Drzewa dowolnego rzędu pozwalają na wygodne odwzorowanie struktur hierarchicznych, takich jak schematy organizacyjne.
- Implementacja drzewa z użyciem wskaźników **shared_ptr** upraszcza zarządzanie pamięcią i eliminuje ryzyko wycieków.
- Funkcja rekurencyjna **printTree** umożliwia prostą i czytelną prezentację drzewa niezależnie od jego głębokości.
- Dodawanie dzieci do węzłów za pomocą funkcji **addChild** pozwala na dynamiczną rozbudowę struktury.
- W praktyce programowanie z użyciem drzew **n-ary** może być przydatne przy modelowaniu złożonych danych o nieregularnej strukturze.