Sprawozdanie 17.03.2024

Sebastian Tarczyński niest. 1 rok, LK4

Implementacja i operacje na listach jedno- i dwukierunkowych

1. Struktura plików

Projekt składa się z trzech głównych plików:

SinglyLinkedList.h: Definicja i implementacja klasy SinglyLinkedList, która zarządza listą jednokierunkową.

DoublyLinkedList.h: Definicja i implementacja klasy DoublyLinkedList, która zarządza listą dwukierunkową.

main.cpp: Plik główny, który wykonuje operacje na obiektach obu list, prezentując ich funkcjonalności.

2. Funkcjonalności listy jednokierunkowej (SinglyLinkedList)

Klasa SinglyLinkedList oferuje następujące funkcje:

- push_front(int val): Dodaje element na początek listy.
- push_back(int val): Dodaje element na koniec listy.
- remove_front(): Usuwa element z początku listy.
- remove_back(): Usuwa element z końca listy.
- remove(int val): Usuwa element o określonej wartości z listy; może to być element z początku, końca lub ze środka.
- print(): Wyświetla wszystkie elementy listy od początku do końca.
- search(int val): Wyszukuje element o danej wartości w liście i zwraca true, jeśli element jest obecny.

3. Funkcjonalności listy dwukierunkowej (DoublyLinkedList)

Klasa DoublyLinkedList zapewnia podobne funkcjonalności do listy jednokierunkowej, ale z dodatkowymi możliwościami wynikającymi z jej dwukierunkowej struktury:

- push_front(int val): Dodaje element na początek listy.
- push_back(int val): Dodaje element na koniec listy.
- remove_front(): Usuwa pierwszy element listy, dbając o odpowiednie zarządzanie wskaźnikiem prev w następnym węźle.
- remove_back(): Usuwa ostatni element listy, aktualizując wskaźnik prev poprzedniego węzła.

- remove(int val): Usuwa element o danej wartości, zarządzając wskaźnikami next i prev dla zachowania spójności listy.

- print(): Wyświetla wszystkie elementy listy.

- search(int val): Wyszukuje element o danej wartości w liście.

4. Demonstracja użycia w main.cpp

Plik main.cpp demonstruje wykorzystanie obu list poprzez:

- Dodanie elementów na początek i koniec list.

- Wyszukiwanie specyficznych wartości w obu listach.

- Usunięcie wybranych elementów.

- Wyświetlenie zawartości list po wykonaniu operacji.

- Dodanie elementów na koniec listy po serii operacji, pokazując elastyczność list w zarządzaniu danymi.

5. Wnioski

Implementacja list jedno- i dwukierunkowych w C++ demonstruje jak struktury danych mogą być wykorzystywane do efektywnego zarządzania danymi w sposób dynamiczny. Możliwość dodawania, usuwania i wyszukiwania elementów w obu typach list jest kluczowa w wielu aplikacjach, od algorytmów po aplikacje zarządzające danymi. Listy dwukierunkowe oferują dodatkową elastyczność w manipulacji danymi, co jest przydatne w bardziej złożonych scenariuszach zarządzania danymi.

6. Zdjęcia kodu z komentarzami i działania programu

17-03 - SinglyLinkedList.h

```cpp
// SinglyLinkedList.h
#ifndef SINGLY_LINKED_LIST_H
#define SINGLY_LINKED_LIST_H

#include <iostream>

struct Node
{
    int value;
    Node *next;

    Node(int val) : value(val), next(nullptr) {}
};

class SinglyLinkedList
{
public:
    Node *head;

    SinglyLinkedList() : head(nullptr) {}

    void push_front(int val)
    {
        Node *newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    void push_back(int val)
    {
        Node *newNode = new Node(val);
        if (head == nullptr)
        {
            head = newNode;
        }
        else
        {
            Node *current = head;
            while (current->next != nullptr)
            {
                current = current->next;
            }
            current->next = newNode;
        }
    }

    void print()
    {
        Node *current = head;
        while (current != nullptr)
        {
            std::cout << current->value << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }

    bool search(int val)
    {
        Node *current = head;
        while (current != nullptr)
        {
            if (current->value == val)
            {
                return true;
            }
            current = current->next;
        }
        return false;
    }
```

17-03 - SinglyLinkedList.h

```cpp

    void remove_front()
    {
        if (head != nullptr)
        {
            Node *temp = head;
            head = head->next;
            delete temp;
        }
    }

    void remove_back()
    {
        if (head == nullptr)
        {
            return;
        }
        else if (head->next == nullptr)
        {
            delete head;
            head = nullptr;
        }
        else
        {
            Node *current = head;
            while (current->next->next != nullptr)
            {
                current = current->next;
            }
            delete current->next;
            current->next = nullptr;
        }
    }

    void remove(int val)
    {
        Node *current = head;
        Node *previous = nullptr;
        while (current != nullptr)
        {
            if (current->value == val)
            {
                if (previous == nullptr)
                {
                    head = current->next;
                }
                else
                {
                    previous->next = current->next;
                }
                delete current;
                return;
            }
            previous = current;
            current = current->next;
        }
    }
};

#endif // SINGLY_LINKED_LIST_H

```

17-03 - DoublyLinkedList.h

```cpp
// DoublyLinkedList.h
#ifndef DOUBLY_LINKED_LIST_H
#define DOUBLY_LINKED_LIST_H

#include <iostream>

struct DNode
{
    int value;
    DNode *next;
    DNode *prev;

    DNode(int val) : value(val), next(nullptr), prev(nullptr) {}
};

class DoublyLinkedList
{
public:
    DNode *head;
    DNode *tail;

    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    void push_front(int val)
    {
        DNode *newNode = new DNode(val);
        if (head == nullptr)
        {
            head = tail = newNode;
        }
        else
        {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    void push_back(int val)
    {
        DNode *newNode = new DNode(val);
        if (tail == nullptr)
        {
            head = tail = newNode;
        }
        else
        {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void print()
    {
        DNode *current = head;
        while (current != nullptr)
        {
            std::cout << current->value << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }

    bool search(int val)
    {
        DNode *current = head;
        while (current != nullptr)
        {
            if (current->value == val)
            {
                return true;
            }
            current = current->next;
        }
        return false;
    }
```

17-03 - DoublyLinkedList.h

```cpp

    void remove_front()
    {
        if (head != nullptr)
        {
            DNode *temp = head;
            head = head->next;
            if (head != nullptr)
            {
                head->prev = nullptr;
            }
            else
            {
                tail = nullptr;
            }
            delete temp;
        }
    }

    void remove_back()
    {
        if (tail != nullptr)
        {
            DNode *temp = tail;
            tail = tail->prev;
            if (tail != nullptr)
            {
                tail->next = nullptr;
            }
            else
            {
                head = nullptr;
            }
            delete temp;
        }
    }

    void remove(int val)
    {
        DNode *current = head;
        while (current != nullptr)
        {
            if (current->value == val)
            {
                if (current->prev != nullptr)
                {
                    current->prev->next = current->next;
                }
                else
                {
                    head = current->next;
                }
                if (current->next != nullptr)
                {
                    current->next->prev = current->prev;
                }
                else
                {
                    tail = current->prev;
                }
                delete current;
                return;
            }
            current = current->next;
        }
    }
};

#endif // DOUBLY_LINKED_LIST_H

```

17-03 - main.cpp

```cpp
// main.cpp
#include "SinglyLinkedList.h"
#include "DoublyLinkedList.h"

int main()
{
    SinglyLinkedList sList;
    DoublyLinkedList dList;

    // Dodawanie elementów do listy jednokierunkowej
    sList.push_front(1);
    sList.push_back(2);
    sList.push_front(3); // Lista: 3, 1, 2

    // Dodawanie elementów do listy dwukierunkowej
    dList.push_front(1);
    dList.push_back(2);
    dList.push_front(3); // Lista: 3, 1, 2

    // Wyszukiwanie wartości w liście jednokierunkowej
    int searchValue = 2;
    std::cout << "Lista jednokierunkowa: ";
    sList.print();
    if (sList.search(searchValue))
    {
        std::cout << "Znaleziono wartość " << searchValue << " w liście jednokierunkowej." << std::endl;
    }
    else
    {
        std::cout << "Nie znaleziono wartości " << searchValue << " w liście jednokierunkowej." << std::endl;
    }

    // Wyszukiwanie wartości w liście dwukierunkowej
    searchValue = 3;
    std::cout << "Lista dwukierunkowa: ";
    dList.print();
    if (dList.search(searchValue))
    {
        std::cout << "Znaleziono wartość " << searchValue << " w liście dwukierunkowej." << std::endl;
    }
    else
    {
        std::cout << "Nie znaleziono wartości " << searchValue << " w liście dwukierunkowej." << std::endl;
    }

    std::cout << "Lista jednokierunkowa przed usunięciem: ";
    sList.print();
    std::cout << "Lista dwukierunkowa przed usunięciem: ";
    dList.print();

    sList.remove_front(); // Usuwa 3
    sList.remove_back();  // Usuwa 2
    sList.push_front(5);  // Dodaje na początek
    sList.remove(1);      // Usuwa 1 (środek)

    // Usuwanie elementów z listy dwukierunkowej
    dList.remove_front(); // Usuwa 3
    dList.remove_back();  // Usuwa 2
    dList.push_front(5);  // Dodaje na początek
    dList.remove(1);      // Usuwa 1 (środek)

    // Wyświetlanie list po usunięciach
    std::cout << "Lista jednokierunkowa po usunięciu: ";
    sList.print();
    std::cout << "Lista dwukierunkowa po usunięciu: ";
    dList.print();

    // Dodawanie nowych elementów na koniec list
    sList.push_back(8); // Dodaje 8 na koniec listy jednokierunkowej
    dList.push_back(8); // Dodaje 8 na koniec listy dwukierunkowej

    // Wyświetlanie zawartości list po dodaniu elementów na koniec
    std::cout << "Lista jednokierunkowa po dodaniu na koniec: ";
    sList.print();
    std::cout << "Lista dwukierunkowa po dodaniu na koniec: ";
    dList.print();

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE SEARCH ERROR

```
dev@dev-mbp-1 17-03 % cd "/Users/dev/Documents/projects/sem2/ppc/17-03"
cd "/Users/dev/Documents/projects/sem2/ppc/17-03/"
dev@dev-mbp-1 17-03 % cd "/Users/dev/Documents/projects/sem2/ppc/17-03/" && g++ -std=c++11 main.cpp -o main && "/Users/dev/Documents/projects/sem2/ppc/17-03/"main
Lista jednokierunkowa: 3 1 2
Znaleziono wartość 2 w liście jednokierunkowej.
Lista dwukierunkowa: 3 1 2
Znaleziono wartość 3 w liście dwukierunkowej.
Lista jednokierunkowa przed usunięciem: 3 1 2
Lista dwukierunkowa przed usunięciem: 3 1 2
Lista jednokierunkowa po usunięciu: 5
Lista dwukierunkowa po usunięciu: 5
Lista jednokierunkowa po dodaniu na koniec: 5 8
Lista dwukierunkowa po dodaniu na koniec: 5 8
dev@dev-mbp-1 17-03 %
```