AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

...Algorytm listy dwukierunkowej z zastosowaniem GitHub...

Autor:
Mateusz Magiera

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2025

Spis treści

1. Ogólne określenie wymagań

Celem projektu było zaprojektowanie i zaimplementowanie w języku C++ listy dwukierunkowej działającej na stercie, zgodnie z zasadami programowania obiektowego. Każda klasa miała znajdować się w osobnym pliku źródłowym. Program miał również wykorzystywać wzorzec projektowy Factory, a do poruszania się po liście — własnoręcznie zaimplementowany wzorzec Iterator. Implementacja listy dwukierunkowej musiała umożliwiać wykonanie wszystkich wymaganych operacji, w tym:

- dodanie elementu na początek listy,
- dodanie elementu na koniec listy,
- dodanie elementu pod wskazany indeks,
- usunięcie elementu z początku,
- usunięcie elementu z końca,
- usunięcie elementu pod wskazanym indeksem,
- wyświetlenie całej listy,
- wyświetlenie listy w odwrotnej kolejności,
- wyświetlenie następnego elementu (iterator → next),
- wyświetlenie poprzedniego elementu (iterator → prev),
- wyczyszczenie całej listy.

1.1. Instalacja

Program uruchamia się z pliku main.cpp, w którym testowane są wszystkie funkcjonalności listy.

Kolejne operacje wykonywane w main.cpp:

Utworzenie listy za pomocą fabryki (ListFactory::createList()).

Dodanie kilku elementów na koniec listy (10, 20, 30).

Dodanie elementów na początek (5, 0).

Wstawienie elementu 15 pod indeks 3.

Usuwanie z początku i końca listy.

Usuwanie elementu spod indeksu 1.

Wyświetlenie listy w obu kierunkach.

Test działania iteratorów:

iteracja od początku do końca (next),

iteracja od końca do początku (prev),

wyświetlenie elementu bieżącego, następnego i poprzedniego.

Wyczyszczenie całej listy i ponowne wyświetlenie jej stanu.

Program prezentuje działanie wszystkich wymaganych metod, a także potwierdza poprawność implementacji iteratora.

Następnym krokiem jest ustawienie w /TeXStudio kolejności budowania projektu. Należy wybrać zakładkę: „Opcje/Konfiguruj /TeXstudio...". W otwartym oknie przechodzimy na zakładkę „Zbuduj". Na rysunku /refrys:ustawienia (s. /pagerefrys:ustawienia) pokazany jest zrzut ekranu z konfiguracją. W linijce „Zbuduj i pokaż" klikamy ikonę klucza, żeby przejść do konfiguracji polecenia. W otwartym oknie ustawić kolejność tak jak pokazano na rysunku.

1.2. Instalacja i uruchamianie

Wymagane narzędzia:

Kompilator C++ (MSVC, MinGW lub g++)

Visual Studio / Visual Studio Code

Doxygen (do dokumentacji)

Opcjonalnie: MiKTeX / TeX Live, jeśli potrzebny jest PDF dokumentacji

Sposób kompilacji:

W Visual Studio: uruchomić projekt i zbudować solution.

W VS Code: użyć g++ *.cpp -o lista.exe.

Generowanie dokumentacji Doxygen:

Wygenerowanie pliku Doxyfile:

doxygen -g

Edycja podstawowych ustawień (INPUT, HTML$_{OUTPUT}, TREEVIEW, EXTRACT_{ALL}$).

Generowanie dokumentacji:

doxygen Doxyfile

Otworzenie docs/html/index.html.

2. Analiza problemu

Lista dwukierunkowa jest dynamiczną strukturą danych, w której każdy węzeł przechowuje:

wartość typu int,

wskaźnik na poprzedni element,

wskaźnik na następny element.

Struktura ta pozwala na:

łatwe usuwanie i dodawanie elementów w dowolnym miejscu listy,

przechodzenie listy w obu kierunkach,

efektywne operacje na początku i końcu.

Ponieważ w tym projekcie wymagane było również wykorzystanie wzorców projektowych, lista została rozbudowana o:

Factory – tworzenie listy na stercie (ukrycie sposobu tworzenia obiektu),

Iterator – obsługa sekwencyjnego przechodzenia po liście bez ujawniania jej struktury.

Całość podzielono na osobne klasy i pliki, zgodnie z zasadami OOP i enkapsulacji.

3. Projektowanie i implementacja

3.1. Klasa Node

Projekt składa się z następujących plików:

Node.h, Node.cpp – definicja i implementacja węzła listy,

DoublyLinkedList.h, DoublyLinkedList.cpp – właściwa lista dwukierunkowa,

ListIterator.h, ListIterator.cpp – iterator umożliwiający przechodzenie po liście,

ListFactory.h, ListFactory.cpp – wzorzec Factory tworzący listę,

main.cpp – test działania wszystkich metod.

3.2. Klasa DoublyLinkedList

Klasa Node reprezentuje pojedynczy węzeł listy. Zawiera:

- pole data,
- wskaźnik next,
- wskaźnik prev.

Jest to najprostszy elementarna struktura elementu listy.

3.3. Struktura projektu

Implementuje wszystkie wymagane funkcje:

dodawanie (push$_f$$ront, push_b ack, insert_a t$),

usuwanie (pop$_f$$ront, pop_b ack, remove_a t$),

wyświetlanie (print$_f$$orward, print_b ackward$),

czyszczenie (clear),

zwracanie iteratorów (begin, rbegin).

Lista przechowuje wskaźniki:

head,

tail,

oraz zmienną length.

Zarządza pamięcią dynamiczną poprzez new i delete.

3.4. Klasa ListIterator

Iterator zawiera wskaźnik current i pozwala na:

sprawdzanie poprawności (isValid()),

pobranie wartości (get()),

przejście w przód (next()),

przejście w tył (prev()),

sprawdzenie, czy istnieje dalszy/poprzedni element (hasNext, hasPrev).

Dzięki niemu nie trzeba „ręcznie” iterować po wskaźnikach.

3.5. Klasa ListIterator

Klasa ListFactory udostępnia metodę statyczną:

static DoublyLinkedList* createList();

Służy to ukryciu sposobu tworzenia listy i wymusza wykorzystanie alokacji dynamicznej (zgodnie z wymaganiami zadania).

4. Testy działania programu

Wszystkie funkcje zostały przetestowane w main.cpp.

Wyniki testów:

Dodawanie na początek/koniec działa poprawnie — elementy pojawiają się w odpowiedniej kolejności.

Wstawianie pod indeks działa zarówno dla środka listy, jak i indeksów skrajnych.

Usuwanie działa poprawnie dla każdego przypadku — pierwszy element, ostatni oraz elementy pośrednie.

Wyświetlanie w obu kierunkach wskazuje, że wskaźniki prev i next są poprawnie aktualizowane.

Iterator poprawnie przechodzi przez listę:

- od początku do końca,

- od końca do początku,

- poprawnie obsługuje brak poprzedniego/następnego elementu.

Czyszczenie listy usuwa wszystkie elementy i nie powoduje wycieków pamięci ani błędów dostępu.

Program działa stabilnie i poprawnie dla wszystkich testowanych operacji.

5. Wnioski

W ramach projektu udało się zrealizować wszystkie wymagania zadania:

zaprojektowano i zaimplementowano w pełni funkcjonalną listę dwukierunkową,

zastosowano wzorce projektowe Factory oraz Iterator,

zachowano podział na pliki i zasady programowania obiektowego,

wszystkie funkcje zostały szczegółowo przetestowane.

Projekt pozwolił na praktyczne utrwalenie pracy z dynamiczną pamięcią w C++, wskaźnikami, klasami oraz implementacją wzorców projektowych. Dodatkowo wygenerowano dokumentację techniczną za pomocą Doxygena, co jest zgodne z wymaganiami profesjonalnych projektów.

Spis rysunków

Spis tabel

Spis listingów