

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

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

Autor:
Mateusz Basiaga

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań	4
1.1. Projekt wieloplikowy	4
1.2. Użycie systemu kontroli wersji Git i platformy GitHub	4
1.3. Dokumentacja projektowa	4
1.4. Funkcjonalność listy dwukierunkowej	5
1.5. Interfejs CLI użytkownika	5
1.6. Obsługa platform	5
2. Analiza problemu	6
2.1. Zastosowanie algorytmu listy dwukierunkowej[1]	6
2.2. Działanie listy dwukierunkowej	7
2.3. Wykorzystanie systemu kontroli wersji Git i platformy GitHub	8
2.3.1. Systemy kontroli wersji	8
2.3.2. Czym jest Git	8
2.3.3. Platformy hostowania kodu źródłowego	9
3. Projektowanie	10
3.1. Narzędzia i technologie	10
3.2. Ustawienia kompilatora	10
3.3. Schemat działania programu	11
3.4. Użycie narzędzi	12
3.5. Przykładowa konfiguracja CMake	13
4. Implementacja	14
4.1. Struktura Node	14
4.2. Klasa DoublyLinkedList	14
4.3. Klasa App	15
4.4. Wyniki implementacji	16
4.4.1. Obsługa programu	16
4.5. Wykorzystanie systemu Git i platformy GitHub	17
5. Wnioski	18

5.1. Wnioski dotyczące implementacji	18
5.2. Perspektywy rozwoju	19
Literatura	20
Spis rysunków	21
Spis tabel	22
Spis listingów	23

1. Ogólne określenie wymagań

W ramach realizacji projektu zostały zdefiniowane następujące wymagania dotyczące struktury, funkcjonalności oraz środowiska pracy aplikacji implementującej listę dwukierunkową:

1.1. Projekt wieloplikowy

Projekt powinien być zorganizowany w sposób modularny, z rozdzieleniem odpowiedzialności na różne pliki źródłowe i nagłówkowe. W szczególności:

- Kod implementujący logikę listy dwukierunkowej powinien być umieszczony w osobnym pliku źródłowym (`doubly_linked_list.cpp`) oraz odpowiadającym mu pliku nagłówkowym (`doubly_linked_list.hpp`).
- Funkcje obsługujące interfejs aplikacji, takie jak menu oraz obsługa interakcji z użytkownikiem, powinny być umieszczone w dedykowanym pliku (`app.cpp`).

1.2. Użycie systemu kontroli wersji Git i platformy GitHub

Projekt powinien być zarządzany za pomocą systemu kontroli wersji Git, a repozytorium projektu powinno być umieszczone na platformie GitHub, co pozwoli na:

- Śledzenie zmian w kodzie i historii projektu.
- Współpracę z innymi członkami zespołu.
- Zastosowanie mechanizmu Continuous Integration (CI) w celu automatycznej weryfikacji poprawności kodu.

1.3. Dokumentacja projektowa

Dokumentacja projektu powinna być dostępna w dwóch formach:

- **Dokumentacja zautomatyzowana** – generowana automatycznie z kodu za pomocą narzędzia `Doxygen`. Powinna zawierać szczegółowe informacje o strukturze klas, funkcjach oraz parametrach.
- **Dokumentacja teoretyczna** – opracowana ręcznie w formacie \LaTeX , obejmująca między innymi opis wymagań, architekturę systemu oraz instrukcję użytkownika.

1.4. Funkcjonalność listy dwukierunkowej

Implementacja listy dwukierunkowej powinna oferować następujące funkcje:

- Dodawanie elementów na początku i końcu listy.
- Wstawianie elementów na dowolnym indeksie.
- Usuwanie elementów z początku, końca oraz z wybranego indeksu.
- Wyświetlanie wszystkich elementów listy w porządku od początku do końca oraz w odwrotnej kolejności.
- Usuwanie wszystkich elementów z listy.

1.5. Interfejs CLI użytkownika

Aplikacja powinna oferować interfejs linii poleceń (CLI) umożliwiający użytkownikowi korzystanie z funkcji listy w sposób intuicyjny. Interfejs powinien obejmować:

- Wyświetlanie menu wyboru operacji.
- Obsługę dodawania, usuwania i wyświetlania elementów.
- Możliwość przeglądania listy w obu kierunkach, zarówno od początku do końca, jak i w odwrotnej kolejności.
- Opcję zakończenia pracy z aplikacją.

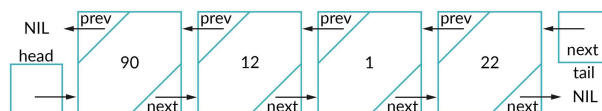
1.6. Obsługa platform

Aplikacja powinna działać poprawnie na następujących platformach:

- **Windows** – wsparcie dla kompilacji i uruchomienia na systemach z rodziny Windows.
- **Linux** – wsparcie dla kompilacji i uruchomienia na systemach operacyjnych z rodziny Linux.

2. Analiza problemu

Lista dwukierunkowa to lista, w której możemy poruszać się w dwóch kierunkach. Każdy element zawiera co najmniej trzy pola: klucz, prev oraz next. W rezultacie elementy wskazują zarówno swoje poprzedniki, jak i następniki. Strukturę logiczną listy przedstawiono na rysunku 2.1¹



Rys. 2.1. Lista dwukierunkowa

2.1. Zastosowanie algorytmu listy dwukierunkowej[1]

Listy dwukierunkowe są szeroko stosowane w różnych dziedzinach informatyki, gdy konieczne jest przechowywanie i manipulowanie dynamicznie zmieniającą się liczbą elementów, przy jednoczesnym zachowaniu możliwości poruszania się w obu kierunkach — do przodu i do tyłu. Typowe zastosowania list dwukierunkowych obejmują:

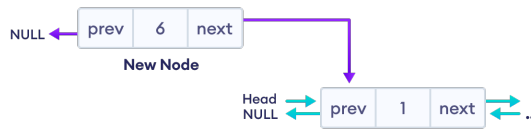
- Implementację struktur takich jak deki (kolejki dwustronne), które pozwalają na dodawanie i usuwanie elementów z obu końców.
- Systemy zarządzania pamięcią, gdzie listy dwukierunkowe są używane do śledzenia dostępnych i zajętych bloków pamięci.
- Nawigację w przeglądarkach internetowych (przycisk „wstecz” i „dalej”), gdzie konieczne jest poruszanie się w obu kierunkach między odwiedzionymi stronami.
- Implementację buforów cyklicznych, gdzie struktura umożliwia płynne przechodzenie pomiędzy początkowym a końcowym elementem listy.
- Różne algorytmy wyszukiwania i sortowania, które wymagają przemieszczania się po danych w obu kierunkach.

¹Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

2.2. Działanie listy dwukierunkowej

- **Wstawianie elementów** na początku, na końcu lub na podanym indeksie.

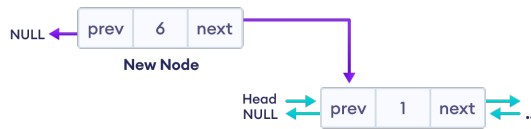
Na rysunku 2.2² pokazano dodawanie elementu na początek listy.



Rys. 2.2. Dodawanie elementu na początek listy

- **Usuwanie elementów** z początku, z końca lub z dowolnej pozycji w liście oraz czyszczenie listy.

Na rysunku 2.3³ pokazano usuwanie elementu z początku listy.



Rys. 2.3. Usuwanie elementu z początku listy

- **Wyświetlanie elementów** zarówno w kolejności od początku do końca, jak i w odwrotnej kolejności.
- **Przeglądanie listy**, zaczynając od określonego indeksu, w obu kierunkach (do przodu lub do tyłu).

²Zdjęcie ze strony <https://www.programiz.com/dsa/doubly-linked-list>[2] (Dostęp: 24.10.2024r.)

³Zdjęcie ze strony <https://www.programiz.com/dsa/doubly-linked-list>[2] (Dostęp: 24.10.2024r.)

2.3. Wykorzystanie systemu kontroli wersji Git i platformy GitHub

2.3.1. Systemy kontroli wersji [3]

Czym są systemy kontroli wersji? System kontroli wersji — oprogramowanie służące do śledzenia zmian głównie w kodzie źródłowym oraz pomocy programistom w łączeniu zmian dokonanych w plikach przez wiele osób w różnym czasie.

Dlaczego warto znać systemy kontroli wersji? Proces tworzenia i rozwoju oprogramowania polega na nieustannym wprowadzaniu zmian w kodzie. Nawet jeśli pracujemy w pojedynkę z czasem zarządzanie takimi zmianami przy pomocy naiwnych metod (np. tworzenie nowego katalogu opisanego datą i nazwą danej wersji) staje się problematyczne, czasochłonne i na ogół nieefektywne. Nie mamy również łatwej możliwości archiwizacji czy cofania zmian.

Problem jest dodatkowo potęgowany kiedy mamy do czynienia z zespołem programistów pracującym nad projektem. Oprócz oczywistej konieczności ciągłego ręcznego przekazywania sobie zmian pojawia się kwestia rozwiązywania nieuniknionych konfliktów.

Wtedy właśnie z pomocą przychodzą systemy kontroli wersji.

2.3.2. Czym jest Git [3]

Git — system kontroli wersji utworzony w 2005 roku przez społeczność rozwijającą jądro systemu Linux (a w szczególności Linusa Torvaldsa, twórcę Linuksa) po tym jak pogorszyły się jej relacje z firmą odpowiedzialną za stworzenie systemu kontroli BitKeeper, którego wówczas używali.

Charakteryzują go przede wszystkim:

- Szybkość
- Prosta konstrukcja
- Silne wsparcie dla nieliniowego rozwoju (tysięcy równoległych gałęzi)
- Pełne rozproszenie
- Wydajna obsługa dużych projektów, takich jak jądro systemu Linux (szybkość i rozmiar danych)

2.3.3. Platformy hostowania kodu źródłowego

Najpopularniejsze platformy do hostowania kodu to

- [GitHub](#)
- [GitLab](#)
- [BitBucket](#)
- [SourceForge](#)

W tym przypadku skupimy się na platformie GitHub. Warto wspomnieć, że GitHub oferuje studentom darmową subskrypcję Pro do celów edukacyjnych jak również inne pakiety. Więcej na ten temat [pod adresem](#).

3. Projektowanie

3.1. Narzędzia i technologie

W projekcie wykorzystano następujące narzędzia i technologie:

- **Język programowania:** C++.
- **Kompilator:** GCC (GNU Compiler Collection) dla systemów Linux oraz MINGW-w64 dla systemu Windows.
- **System budowania:** CMake z użyciem Ninja jako generatora systemu budowania.
- **Dokumentacja:** Narzędzie Doxygen do generowania dokumentacji technicznej oraz szablon \LaTeX dostosowany przez mgr inż. Dawida Kotlarskiego.
- **Kontrola wersji:** Git oraz platforma GitHub.
- **Git Hooks:** Lefthook do automatyzacji zadań związanych z Git.
- **Continuous Integration:** GitHub Actions do automatyzacji testów i budowy projektu.
- **Walidacja commitów:** Narzędzie Commitlint do sprawdzania wiadomości commitów zgodnie z konwencją Conventional Commits.

3.2. Ustawienia kompilatora

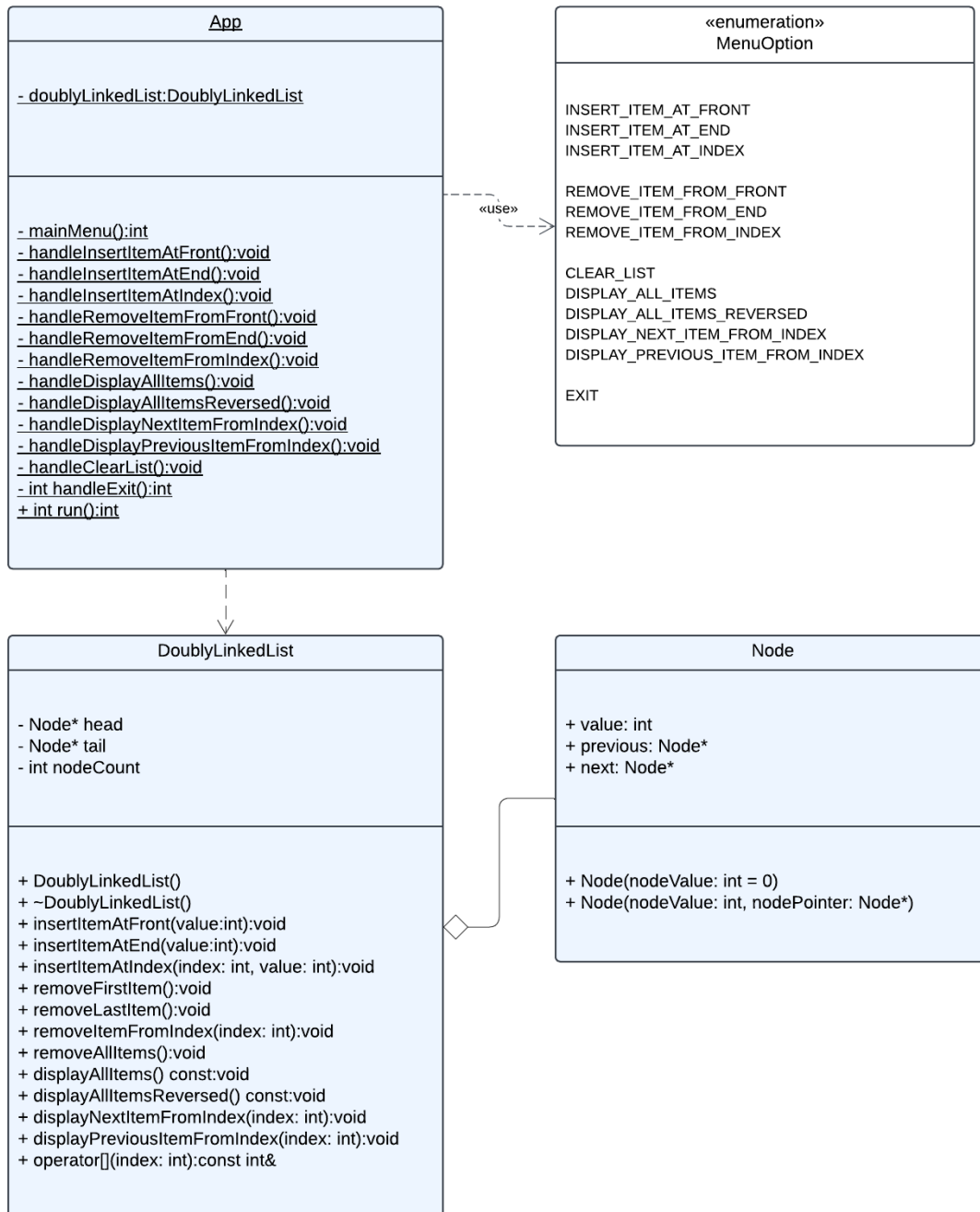
Projekt korzysta z systemu CMake, który umożliwia konfigurację i zarządzanie ustawieniami kompilatora dla różnych platform. Ustawienia są dostosowane do pracy z:

- **Windows:** MINGW-w64 z MSYS2 jako środowiskiem terminalowym.
- **Linux:** GCC, który jest standardowym kompilatorem na większości dystrybucji systemu Linux.

Dzięki zastosowaniu CMake projekt jest w stanie automatycznie generować odpowiednie pliki Makefile dla Ninja, co umożliwia szybkie i efektywne budowanie aplikacji.

3.3. Schemat działania programu

Poniżej przedstawiono diagram UML klas, który ilustruje strukturę i relacje pomiędzy klasami w projekcie.



Rys. 3.1. Diagram UML klas dla projektu

3.4. Użycie narzędzi

Na potrzeby tego i przyszłych projektów o podobnym charakterze został utworzony szablon repozytorium na platformie GitHub. Znajduje jest ono pod adresem: <https://github.com/Me-Phew/programowanie-zaawansowane-template>^[4] (Dostęp: 24.10.2024r.). Zawiera następujące funkcjonalności:

- **Generator systemu budowania:** CMake do zarządzania konfiguracją projektu.
- **Kompleksowa konfiguracja dla GCC (MINGW-w64) oraz Linux.**
- **Automatyzacja budowy projektu:** Ninja jako system budowania.
- **Dokumentacja:** Generowana automatycznie z użyciem Doxygen oraz szablonu LaTeX.
- **Zarządzanie tematami:** Temat Doxygen Awesome z kolorami inspirowanymi stroną Nuxt.
- **Git Hooks:** Lefthook do automatyzacji i walidacji wiadomości commitów.
- **Continuous Integration:** GitHub Actions do automatyzacji procesów budowy i testowania.
- **Generowanie strony dokumentacji automatycznej:** GitHub Pages.
- **Tworzenie wydań:** Zawierające pliki wykonywalne dla systemów Windows i Linux oraz pliki dokumentacji.

3.5. Przykładowa konfiguracja CMake

Przykładowa konfiguracja pliku Zawartość pliku CMakeLists.txt dla projektu została przedstawiona na listingu 3.1

```
1 cmake_minimum_required(VERSION 3.10)
2
3 project(DoublyLinkedList)
4
5 include_directories(src/app src/doubly_linked_list)
6
7 set(SOURCES src/app/app.cpp src/doubly_linked_list/
8     doubly_linked_list.cpp src/main.cpp)
9
10 add_executable(DoublyLinkedList ${SOURCES})
11
12 # Compile with all warnings, treat warnings as errors
13 if (MSVC)
14     add_compile_options(/W4 /WX)
15 else()
16     add_compile_options(-Wall -Wextra -pedantic -Werror)
17 endif()
```

Listing 1. CMakeLists.txt

Powyższa konfiguracja definiuje minimalną wersję CMake, ustawia standard C++ oraz określa pliki źródłowe projektu.

4. Implementacja

4.1. Struktura Node

Pierwszym istotnym elementem jest struktura Node, która reprezentuje pojedynczy węzeł

w podwójnie powiązanej liście. Definicję klasy Node przedstawiono na listingu 4.1:

```
1 struct Node {
2     int value;
3     Node* previous;
4     Node* next;
5
6     explicit Node(int nodeValue = 0);
7
8     Node(int nodeValue, Node* nodePointer);
9
10    friend std::ostream& operator<<(std::ostream& stream, const
11    DoublyLinkedList::Node* node);
12 };
13
```

Listing 2. Struktura Node

Struktura ta definiuje trzy podstawowe atrybuty: wartość przechowywaną w węźle, wskaźnik na następny węzeł oraz wskaźnik na poprzedni węzeł. Konstruktor inicjalizuje te atrybuty.

4.2. Klasa DoublyLinkedList

Główna klasa, DoublyLinkedList, zawiera metody do dodawania, usuwania oraz przeszukiwania węzłów. Jej strukturę przedstawiono na listingu 4.2

```
1 class DoublyLinkedList {
2 private:
3     Node* head;
4     Node* tail;
5     int nodeCount;
6
7 public:
8     void insertItemAtFront(int value);
9
10    void insertItemAtEnd(int value);
11
12    void insertItemAtIndex(int index, int value);
13
```

```
14 void removeFirstItem();
15
16 void removeLastItem();
17
18 void removeItemFromIndex(int index);
19
20 void removeAllItems();
21
22 void displayAllItems() const;
23
24 void displayAllItemsReversed() const;
25
26 void displayNextItemFromIndex(int index) const;
27
28 void displayPreviousItemFromIndex(int index) const;
29 };
```

Listing 3. Klasa DoublyLinkedList

4.3. Klasa App

Klasa App, posiadająca jedynie składowe statyczne odpowiada za uruchomienie programu i wyświetlanie interfejsu użytkownika. Jej strukturę przedstawiono na listingu 4.3

```
1 class App {
2     static DoublyLinkedList doublyLinkedList;
3
4     static int mainMenu();
5
6     static void handleInsertItemAtFront();
7
8     static void handleInsertItemAtEnd();
9
10    static void handleInsertItemAtIndex();
11
12    static void handleRemoveItemFromFront();
13
14    static void handleRemoveItemFromEnd();
15
16    static void handleRemoveItemFromIndex();
17
18    static void handleDisplayAllItems();
19
20    static void handleDisplayAllItemsReversed();
```

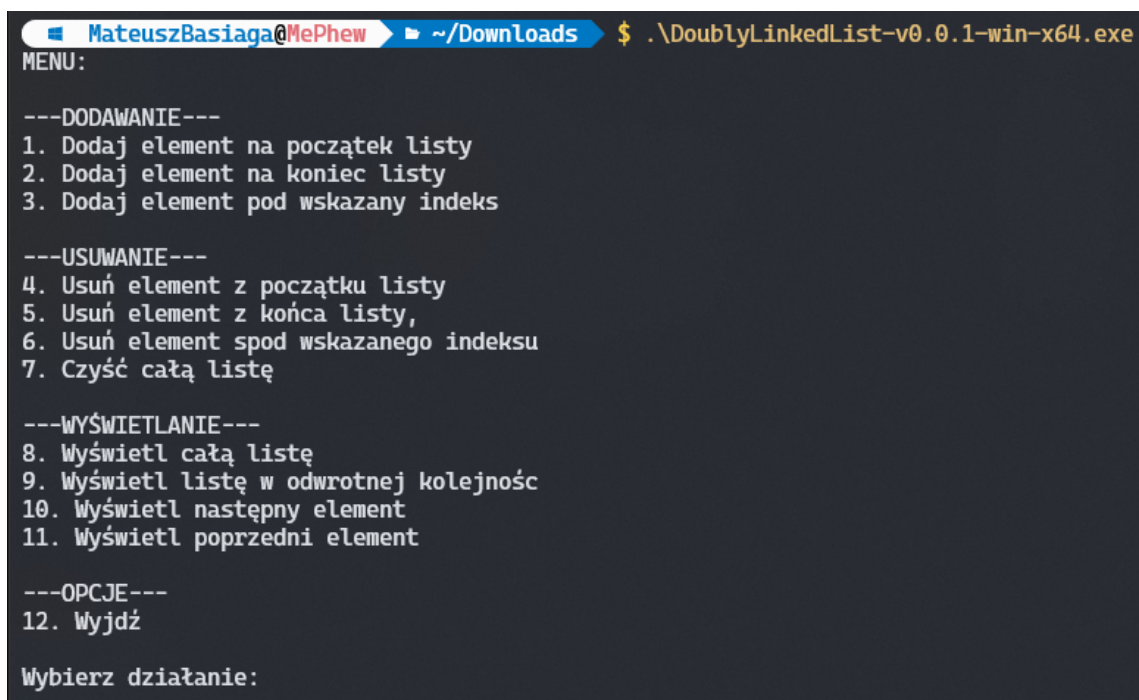
```
21
22     static void handleDisplayNextItemFromIndex();
23
24     static void handleDisplayPreviousItemFromIndex();
25
26     static void handleClearList();
27
28     static int handleExit();
29
30 public:
31     static int run();
32 };
```

Listing 4. Klasa App

4.4. Wyniki implementacji

4.4.1. Obsługa programu

Na rysunku 4.1 przedstawiono menu główne programu.



```
MateuszBasiaga@MePhew ~/Downloads $ ./DoublyLinkedList-v0.0.1-win-x64.exe
MENU:

---DODAWANIE---
1. Dodaj element na początek listy
2. Dodaj element na koniec listy
3. Dodaj element pod wskazany indeks

---USUWANIE---
4. Usuń element z początku listy
5. Usuń element z końca listy,
6. Usuń element spod wskazanego indeksu
7. Czyść całą listę

---WYŚWIETLANIE---
8. Wyświetl całą listę
9. Wyświetl listę w odwrotnej kolejność
10. Wyświetl następny element
11. Wyświetl poprzedni element

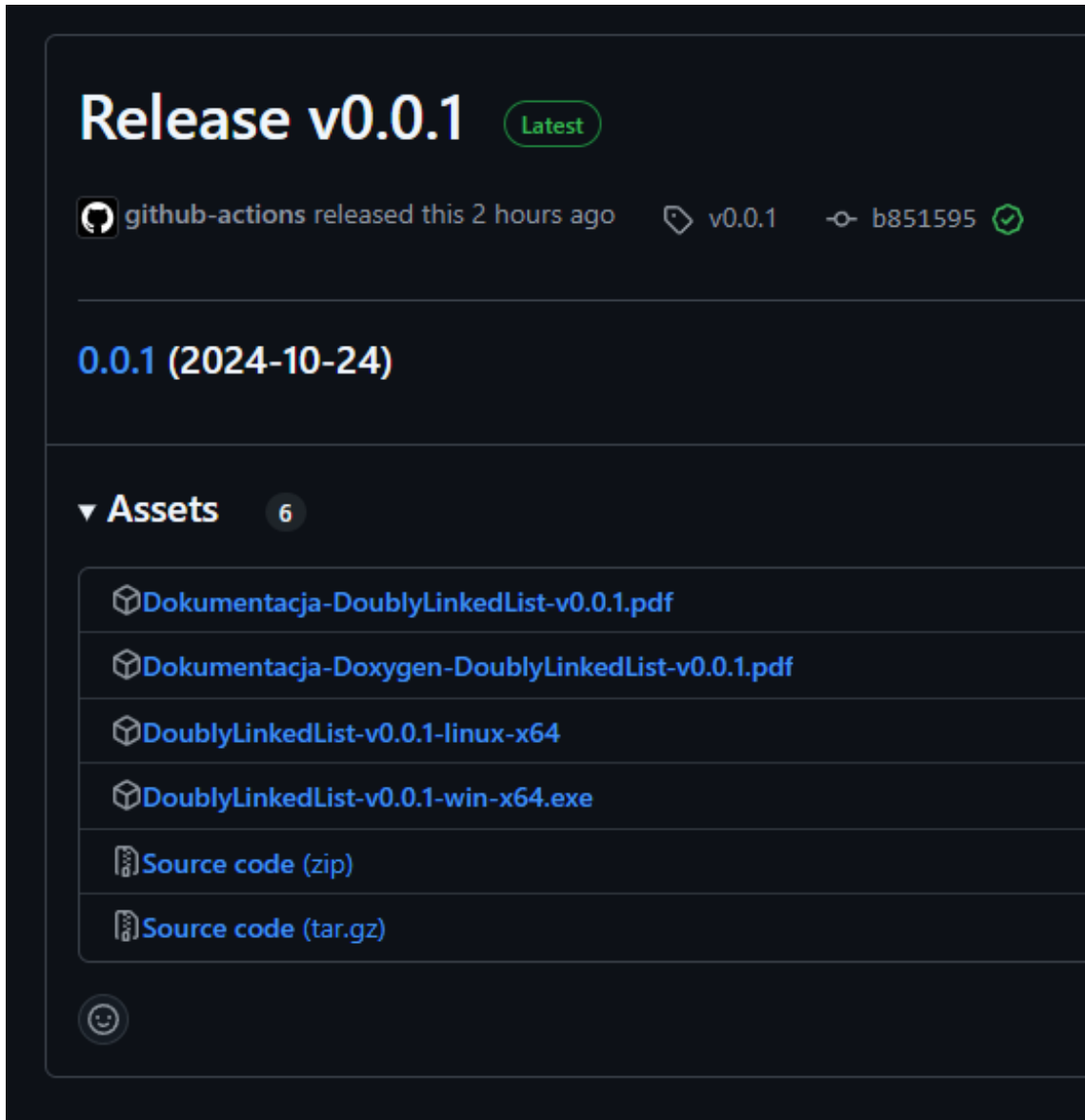
---OPCJE---
12. Wyjdź

Wybierz działanie:
```

Rys. 4.1. Menu główne programu

4.5. Wykorzystanie systemu Git i platformy GitHub

Wdrożono automatyczną publikację nowych wersji programu w ramach systemu CI. Na rysunku 4.2 pokazano pierwszą opublikowaną wersję programu.



Rys. 4.2. Pierwsza opublikowana wersja programu

Jak widać powyżej do każdej wersji programu dołączane są załączniki zawierające dokumentację kodu oraz pliki wykonywalne na platformy Windows oraz Linux.

5. Wnioski

Podsumowanie projektu

Przeprowadzony projekt dotyczący implementacji listy dwukierunkowej przyniósł szereg istotnych rezultatów i wniosków. Zrealizowane zadania obejmowały analizę problemu, projektowanie, implementację oraz testowanie funkcjonalności algorytmu. Dzięki temu udało się stworzyć stabilną i wydajną strukturę danych, która może być używana w różnych zastosowaniach programistycznych.

5.1. Wnioski dotyczące implementacji

1. **Efektywność operacji:** Implementacja podwójnie powiązanej listy umożliwia szybkie dodawanie i usuwanie elementów z dowolnych miejsc w strukturze. Dzięki zastosowaniu wskaźników do poprzednich i następnych węzłów, operacje te są wykonywane w czasie stałym $O(1)$.
2. **Łatwość rozbudowy:** Struktura ta może być łatwo rozszerzana o dodatkowe funkcjonalności, takie jak wyszukiwanie czy sortowanie elementów. Modularność kodu umożliwia przyszłe modyfikacje oraz dostosowanie do nowych wymagań.
3. **Wykorzystanie narzędzi:** W projekcie zastosowano szereg narzędzi, takich jak Git i GitHub do zarządzania wersjami, a także Doxygen do automatycznego generowania dokumentacji. Narzędzia te znacznie ułatwiły współpracę i organizację pracy nad projektem.
4. **Testowanie i weryfikacja:** Przeprowadzone testy ujawniły, że implementacja działa zgodnie z założeniami. Każda z operacji na liście została dokładnie przetestowana, co potwierdziło jej niezawodność.
5. **Zastosowanie w praktyce:** Struktura danych w postaci podwójnie powiązanej listy może być wykorzystana w wielu aplikacjach, takich jak edytory tekstu, aplikacje do zarządzania zadaniami, czy systemy bazodanowe. Jej elastyczność i wydajność czynią ją atrakcyjnym rozwiązaniem w projektach programistycznych.

5.2. Perspektywy rozwoju

Na cele tego projektu utworzono szablon (pod adresem pod adresem: <https://github.com/Me-Phew/programowanie-zaawansowane-template>^[4] (Dostęp: 24.10.2024r.)), który stanowi solidną bazę dla przyszłych projektów. Repozytorium z kodem projektu znajduje się pod adresem <https://github.com/Me-Phew/doubly-linked-list>^[5]

Bibliografia

- [1] *Strona internetowa Zintegrowanej Platformy Edukacyjnej*. URL: <https://zpe.gov.pl/pdf/Puhk7mD92beginning> (term. wiz. 24.10.2024).
- [2] *Strona internetowa Programiz*. URL: <https://www.programiz.com/dsa/doubly-linked-list#insertion-at-beginning> (term. wiz. 24.10.2024).
- [3] Scott Chacon Ben Straub. *Pro Git*. 2 wyd. Apress, 2009.
- [4] *Szablon utworzony na potrzeby projektu (repozytorium GitHub)*. URL: <https://github.com/Me-PheW/programowanie-zaawansowane-template> (term. wiz. 24.10.2024).
- [5] *Repozytorium projektu na platformie GitHub*. URL: <https://github.com/Me-PheW/doubly-linked-list> (term. wiz. 24.10.2024).

Spis rysunków

2.1. Lista dwukierunkowa	6
2.2. Dodawanie elementu na początek listy	7
2.3. Usuwanie elementu z początku listy	7
3.1. Diagram UML klas dla projektu	11
4.1. Menu główne programu	16
4.2. Pierwsza opublikowana wersja programu	17

Spis tabel

Spis listingów

1.	CMakeLists.txt	13
2.	Struktura Node	14
3.	Klasa DoublyLinkedList	14
4.	Klasa App	15