Projektowanie i Analiza Algorytmów

Sprawozdanie

Projekt 1 – Zadania przypominające

Jakub Piekarek

Indeks 264202

Prowadzący dr inż. Krzysztof Halawa

Kod grupy K00-37d

Środa 11^{15} – 13^{00}

Wydział Informatyki i Telekomunikacji

1. Opis rozwiązanych zadań, wyjaśnienie kodu wraz z właściwościami zaimplementowanych metod

1.1. Zadanie 1

Polecenie tego zadania wymagało aby zaimplementować tablice dwuwymiarową alokowaną dynamicznie, która będzie spełniać funkcje taki jak:

- wypełnij tablice losowymi wartościami od 0 do X, gdzie X to parametr funkcji
- wyświetl zawartość funkcji
- znajdź maksymalna wartość
- stworzenie menu do podanych funkcji

Implementacja tablicy dwuwymiarowej alokowanej dynamicznie

```
// Funkcja alokująca pamięć dla tablicy dwuwymiarowej o rozmiarze n x m.
int** alokuj_tablice(int n, int m)
{
    int** tablica = new int* [n];
    for (int i = 0; i < n; i++)
    {
        tablica[i] = new int[m];
    }
    return tablica;
}
```

Funkcja alokuj_tablice() alokuje pamięć dla tablicy dwuwymiarowej o wymiarach n x m. Na początku funkcja tworzy dynamiczną tablicę wskaźników tablica o długości n. Następnie w pętli dla każdego elementu tej tablicy, czyli dla każdego wiersza, alokowana jest nowa dynamiczna tablica o długości m i wskaźnik do niej przypisywany jest do odpowiedniego elementu tablica. Po zakończeniu pętli, funkcja zwraca wskaźnik do utworzonej tablicy tablica.

Funkcja wypełnij tablice

```
// Funkcja wypełniająca tablicę losowymi wartościami od 0 do x.
void wypelnij_tablice(int** tablica, int n, int m, int x)
{
    srand(time(NULL));
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            tablica[i][j] = rand() % (x + 1);
        }
    }
}
```

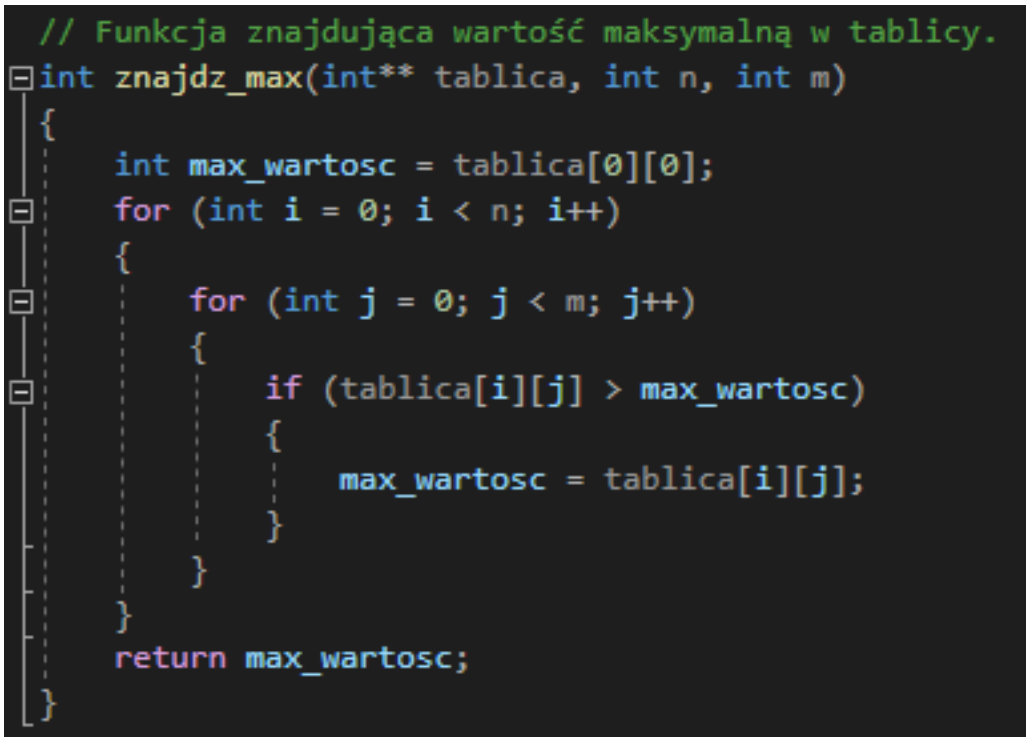
Funkcja wypelnij_tablice() wypełnia tablicę dwuwymiarową wartościami losowymi z przedziału od 0 do x. Na początku funkcja wywołuje funkcję srand() z argumentem time(NULL), aby zainicjować generator liczb pseudolosowych. Następnie w zagnieżdżonej pętli for przeglądane są wszystkie elementy tablicy, a do każdego z nich przypisywana jest losowa wartość z przedziału od 0 do x.

Funkcja wyświetl tablice

```
// Funkcja wyświetlająca zawartość tablicy.
void wyswietl_tablice(int** tablica, int n, int m)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cout << tablica[i][j] << " ";
        }
        cout << endl;
    }
}
```

Funkcja wyswietl_tablice() wyświetla zawartość tablicy dwuwymiarowej. Zagnieżdżone pętle for przechodzą przez wszystkie elementy tablicy, a do każdego z nich odwołuje się za pomocą indeksów i oraz j. Wartość każdego elementu jest wyświetlana na ekranie, oddzielona spacją, a po wyświetleniu wszystkich elementów w wierszu przechodzi do kolejnego wiersza

Funkcja znajdź maksymalną wartość w tablicy



```
// Funkcja znajdująca wartość maksymalną w tablicy.
int znajdz_max(int** tablica, int n, int m)
{
    int max_wartosc = tablica[0][0];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (tablica[i][j] > max_wartosc)
            {
                max_wartosc = tablica[i][j];
            }
        }
    }
    return max_wartosc;
}
```

Ta funkcja znajduje największą wartość w tablicy dwuwymiarowej. Iteruje po wszystkich elementach tablicy i porównuje wartość każdego elementu z aktualną maksymalną wartością. Jeśli wartość elementu jest większa niż aktualna maksymalna wartość, aktualizuje maksymalną wartość. Po przejrzeniu całej tablicy zwraca maksymalną wartość.

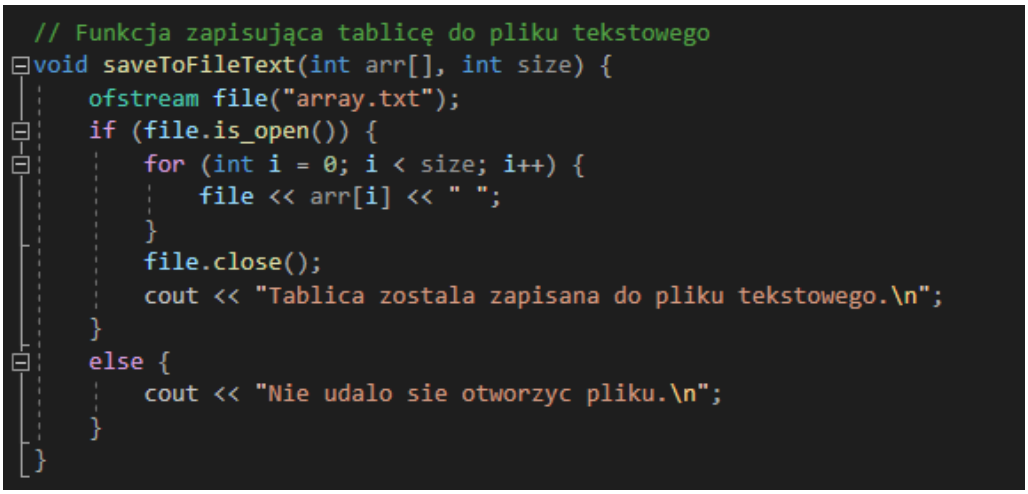
Funkcja main wraz z menu do funkcji opisanych wyżej

```cpp
int main()
{
    int n, m, x;
    int wybor;
    int** tablica;

    cout << "Podaj liczbe wierszy tablicy: ";
    cin >> n;
    cout << "Podaj liczbe kolumn tablicy: ";
    cin >> m;

    tablica = alokuj_tablice(n, m);

    do {
        cout << endl << "MENU" << endl;
        cout << "1. Wypelnij tablice losowymi wartosciami" << endl;
        cout << "2. Wyswietl zawartosc tablicy" << endl;
        cout << "3. Znajdz wartosc maksymalna" << endl;
        cout << "4. Wyjdz z programu" << endl;
        cout << "Wybierz opcje: ";
        cin >> wybor;

        switch (wybor) {
        case 1:
            cout << "Podaj maksymalna wartosc losowanych liczb: ";
            cin >> x;
            wypelnij_tablice(tablica, n, m, x);
            cout << "Tablica zostala wypelniona losowymi wartosciami." << endl;
            break;
        case 2:
            cout << "Zawartosc tablicy:" << endl;
            wyswietl_tablice(tablica, n, m);
            break;
        case 3:
            cout << "Wartosc maksymalna w tablicy wynosi: " << znajdz_max(tablica, n, m) << endl;
            break;
        case 4:
            cout << "Koniec programu." << endl;
            break;
        default:
            cout << "Niepoprawna opcja." << endl;
            break;
        }
    } while (wybor != 4);

    // Zwolnienie zaalokowanej pamięci.
    for (int i = 0; i < n; i++)
    {
        delete[] tablica[i];
    }
    delete[] tablica;

    return 0;
}
```

1.2. Zadanie 2

W tym zadaniu należało zaimplementować tablice jednowymiarową z funkcjami odczytu i wczytującą zawartość z/do pliku tekstowego i pliku binarnego

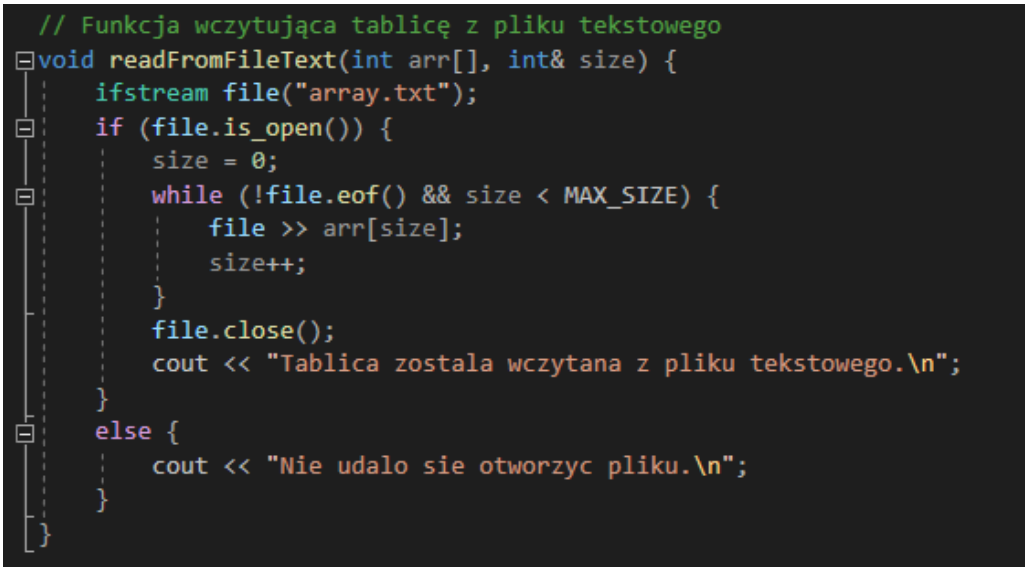
Zapisywanie do pliku tekstowego



```
// Funkcja zapisująca tablicę do pliku tekstowego
void saveToFileText(int arr[], int size) {
    ofstream file("array.txt");
    if (file.is_open()) {
        for (int i = 0; i < size; i++) {
            file << arr[i] << " ";
        }
        file.close();
        cout << "Tablica zostala zapisana do pliku tekstowego.\n";
    }
    else {
        cout << "Nie udalo sie otworzyc pliku.\n";
    }
}
```

Funkcja zapisująca tablicę liczb całkowitych do pliku tekstowego. Jako argumenty przyjmuje tablicę i jej rozmiar. Funkcja otwiera plik "array.txt" i zapisuje wartości tablicy oddzielone spacją do tego pliku. Jeśli plik nie może zostać otwarty, funkcja wypisuje odpowiedni komunikat na konsolę

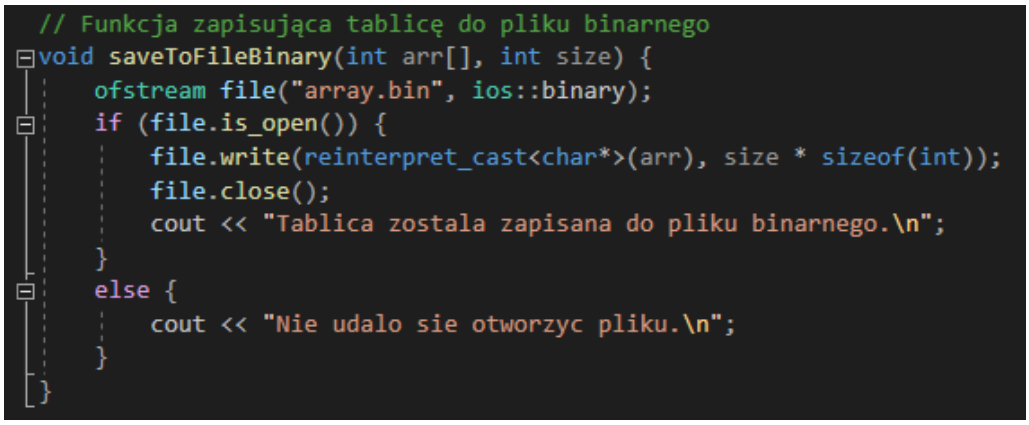
Wczytywanie z pliku tekstowego



```
// Funkcja wczytująca tablicę z pliku tekstowego
void readFromFileText(int arr[], int& size) {
    ifstream file("array.txt");
    if (file.is_open()) {
        size = 0;
        while (!file.eof() && size < MAX_SIZE) {
            file >> arr[size];
            size++;
        }
        file.close();
        cout << "Tablica zostala wczytana z pliku tekstowego.\n";
    }
    else {
        cout << "Nie udalo sie otworzyc pliku.\n";
    }
}
```

Funkcja wczytująca tablicę liczb całkowitych z pliku tekstowego. Jako argumenty przyjmuje tablicę i referencję do jej rozmiaru. Funkcja otwiera plik "array.txt" i wczytuje wartości do tablicy aż do końca pliku lub maksymalnego rozmiaru tablicy. Funkcja zwiększa rozmiar tablicy o ilość wczytanych wartości. Jeśli plik nie może zostać otwarty, funkcja wypisuje odpowiedni komunikat na konsolę.

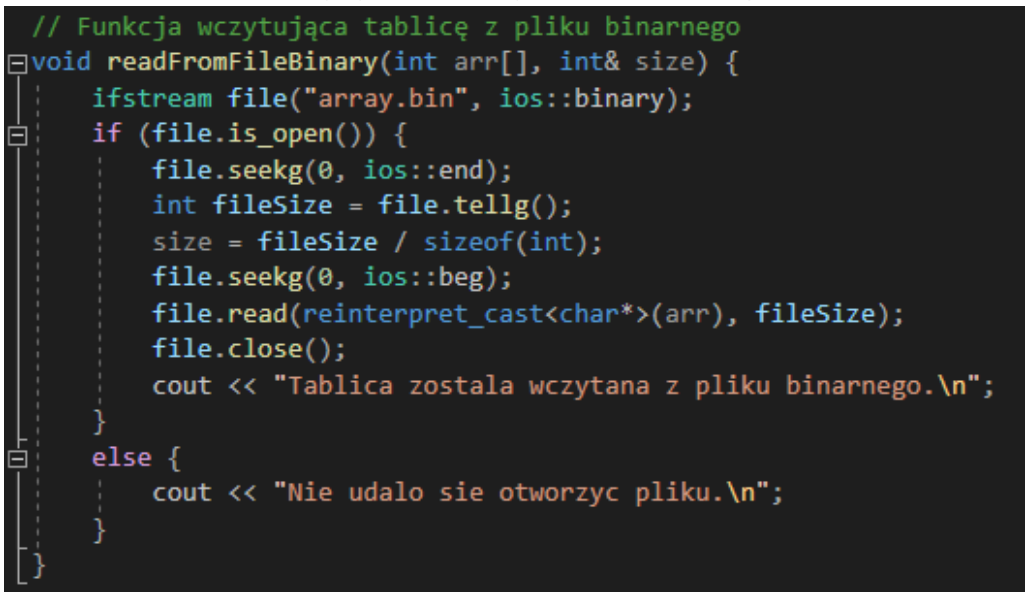
Zapisywanie do pliku binarnego



```
// Funkcja zapisująca tablicę do pliku binarnego
void saveToFileBinary(int arr[], int size) {
    ofstream file("array.bin", ios::binary);
    if (file.is_open()) {
        file.write(reinterpret_cast<char*>(arr), size * sizeof(int));
        file.close();
        cout << "Tablica zostala zapisana do pliku binarnego.\n";
    }
    else {
        cout << "Nie udalo sie otworzyc pliku.\n";
    }
}
```

Funkcja zapisująca tablicę liczb całkowitych do pliku binarnego. Jako argumenty przyjmuje tablicę i jej rozmiar. Funkcja otwiera plik "array.bin" i zapisuje wartości tablicy do tego pliku w formacie binarnym. Jeśli plik nie może zostać otwarty, funkcja wypisuje odpowiedni komunikat na konsolę.

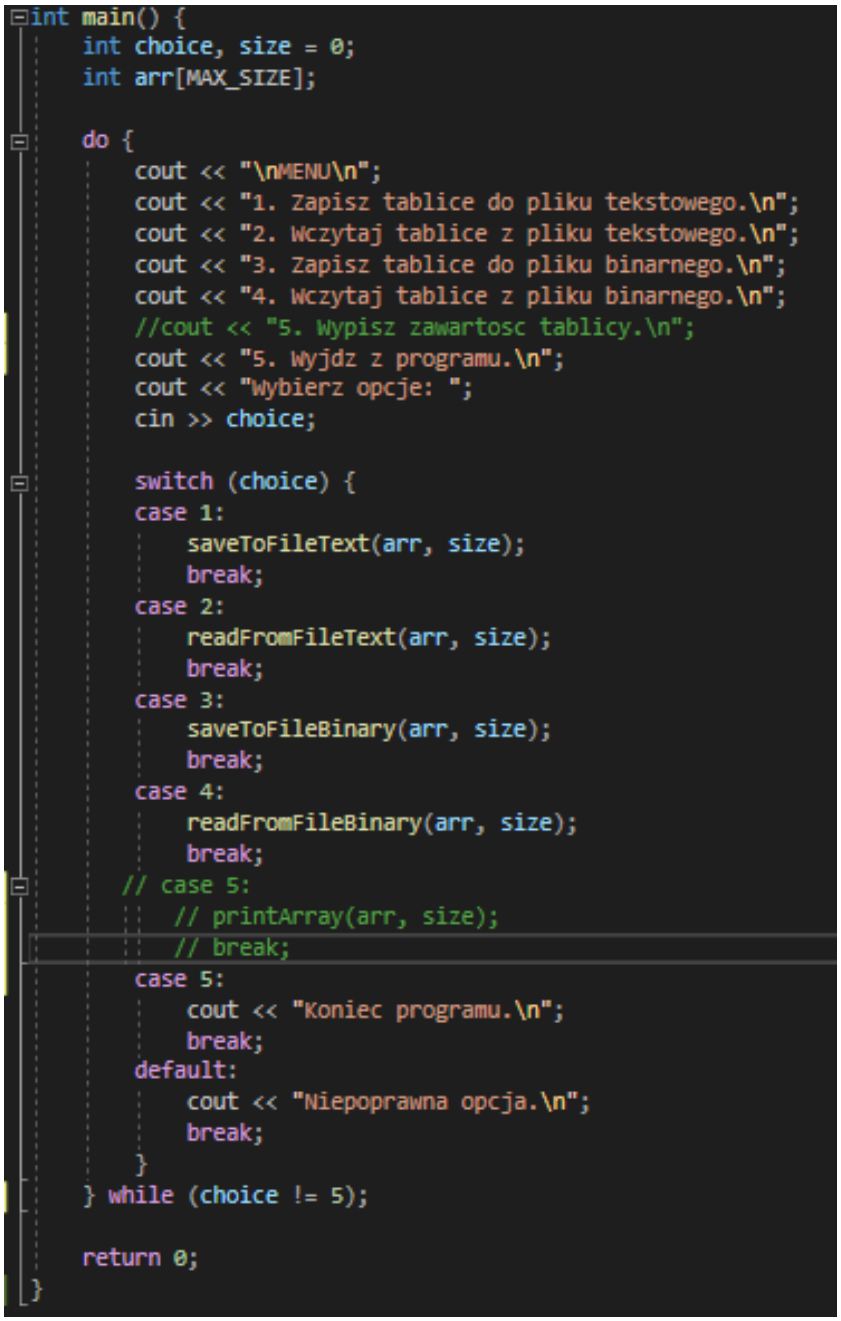
Wczytywanie z pliku binarnego



```
// Funkcja wczytująca tablicę z pliku binarnego
void readFromFileBinary(int arr[], int& size) {
    ifstream file("array.bin", ios::binary);
    if (file.is_open()) {
        file.seekg(0, ios::end);
        int fileSize = file.tellg();
        size = fileSize / sizeof(int);
        file.seekg(0, ios::beg);
        file.read(reinterpret_cast<char*>(arr), fileSize);
        file.close();
        cout << "Tablica zostala wczytana z pliku binarnego.\n";
    }
    else {
        cout << "Nie udalo sie otworzyc pliku.\n";
    }
}
```

Funkcja wczytująca tablicę liczb całkowitych z pliku binarnego. Jako argumenty przyjmuje tablicę i referencję do jej rozmiaru. Funkcja otwiera plik "array.bin" i wczytuje wartości do tablicy w formacie binarnym. Funkcja zwiększa rozmiar tablicy o ilość wczytanych wartości. Jeśli plik nie może zostać otwarty, funkcja wypisuje odpowiedni komunikat na konsolę.

Funkcja main wraz z menu



```
int main() {
    int choice, size = 0;
    int arr[MAX_SIZE];

    do {
        cout << "\nMENU\n";
        cout << "1. Zapisz tablice do pliku tekstowego.\n";
        cout << "2. Wczytaj tablice z pliku tekstowego.\n";
        cout << "3. Zapisz tablice do pliku binarnego.\n";
        cout << "4. Wczytaj tablice z pliku binarnego.\n";
        //cout << "5. Wypisz zawartosc tablicy.\n";
        cout << "5. Wyjdz z programu.\n";
        cout << "Wybierz opcje: ";
        cin >> choice;

        switch (choice) {
        case 1:
            saveToFileText(arr, size);
            break;
        case 2:
            readFromFileText(arr, size);
            break;
        case 3:
            saveToFileBinary(arr, size);
            break;
        case 4:
            readFromFileBinary(arr, size);
            break;
        // case 5:
            // printArray(arr, size);
            // break;
        case 5:
            cout << "Koniec programu.\n";
            break;
        default:
            cout << "Niepoprawna opcja.\n";
            break;
        }
    } while (choice != 5);

    return 0;
}
```

1.3. Zadanie 3

Tutaj należało napisać dwie funkcje rekurencyjne tzn. funkcje obliczającą potęgę oraz silnie

Funkcja rekurencyjna licząca potęgę

```
int Potega(int x, int p) {
    if (p == 0) {
        return 1;
    }
    else if (p == 1) {
        return x;
    }
    else if (p % 2 == 0) {
        int temp = Potega(x, p / 2);
        return temp * temp;
    }
    else {
        int temp = Potega(x, (p - 1) / 2);
        return temp * temp * x;
    }
}
```

Funkcja Potęga to implementacja rekurencyjnego algorytmu podnoszenia liczby całkowitej x do potęgi całkowitej p. Algorytm działa w czasie logarytmicznym, tzn. wymaga log2(p) kroków.

Funkcja rekurencyjna licząca silnię

```
int Silnia(int x) {
    if (x == 0) {
        return 1;
    }
    else {
        return x * Silnia(x - 1);
    }
}
```

Funkcja Silnia oblicza silnię liczby całkowitej x, czyli iloczyn wszystkich liczb naturalnych od 1 do x. Algorytm działa rekurencyjnie, aż do osiągnięcia wartości 0, dla której zwracana jest wartość 1. Funkcja działa w czasie liniowym, czyli wymaga x kroków.

1.4. Zadanie 4

Mieliśmy do napisania funkcje typu bool i sprawdzenie czy dane słowo jest palindromem

```
bool jestPal(string testStr, int start, int end) {
    // Baza rekurencji: łańcuch jest palindromem, gdy pozostała do sprawdzenia jego długość wynosi 0 lub 1
    if (start >= end) {
        return true;
    }
    // Zamiana znaków na małe litery przed porównaniem
    char firstChar = tolower(testStr[start]);
    char lastChar = tolower(testStr[end]);
    // Sprawdzenie, czy pierwszy i ostatni znak są takie same
    if (firstChar != lastChar) {
        return false;
    }
    // Rekurencyjne wywołanie dla łańcucha bez pierwszego i ostatniego znaku
    return jestPal(testStr, start + 1, end - 1);
}
```

Funkcja jestPal sprawdza, czy dany łańcuch znaków testStr jest palindromem, tzn. czy czyta się tak samo od lewej do prawej strony, jak od prawej do lewej. W tym celu funkcja używa podejścia rekurencyjnego. Jeśli łańcuch ma długość 0 lub 1, to jest to palindrom, w przeciwnym wypadku funkcja sprawdza, czy pierwszy i ostatni znak są takie same, a następnie wywołuje się dla łańcucha bez pierwszego i ostatniego znaku.

1.5. Zadanie 5

Na podstawie zadania 4, mieliśmy wykorzystać funkcje do sprawdzania czy jest palindromem. Następnie napisać funkcje do stworzenia permutacji tak aby sprawdzić wszystkie możliwości ułożenia znaków i sprawdzenia czy również są palindromem. Jeśli tak ciąg tych znaków jest palindromem zapisz go do tablicy. A następnie funkcja usunDup, miała działać tak aby usunąć duplikaty ciągów znaków z tablicy.

Globalne zdefiniowanie tablicy palList

```
#include <iostream>
#include <string>
#include <cctype>
#include <vector>
#include <algorithm>

using namespace std;

vector<string> palList; // globalna tablica zawierająca znalezione palindromy
```

Funkcja do permutacji

```
void generujPermutacje(string str, int left, int right) {
    // Baza rekurencji: wygenerowanie jednej permutacji
    if (left == right) {
        if (jestPal(str, 0, str.length() - 1)) {
            palList.push_back(str); // dodanie permutacji do tablicy palindromów
        }
    }
    else {
        // Generowanie permutacji przez zamianę każdego znaku z pozostałymi
        for (int i = left; i <= right; i++) {
            swap(str[left], str[i]);
            generujPermutacje(str, left + 1, right);
            swap(str[left], str[i]);
        }
    }
}
```

Funkcja generujPermutacje generuje wszystkie permutacje znaków w podanym łańcuchu str za pomocą algorytmu rekurencyjnego typu DFS (Depth-First Search). Dla każdej wygenerowanej permutacji funkcja wywołuje funkcję jestPal, która sprawdza, czy permutacja jest palindromem. Jeśli tak, to permutacja zostaje dodana do globalnej tablicy palList.

Funkcja usunDup

```
void usunDup() {
    sort(palList.begin(), palList.end()); // posortowanie tablicy
    palList.erase(unique(palList.begin(), palList.end()), palList.end()); // usunięcie duplikatów
}
```

Funkcja usunDup sortuje tablicę palList za pomocą standardowej funkcji sortującej i usuwa duplikaty za pomocą funkcji erase i unique.

erase() jest funkcją, która usuwa elementy z kontenera na podstawie podanego zakresu indeksów. Może być wykorzystywana w celu usunięcia duplikatów lub elementów, które nie spełniają określonych warunków. Funkcja erase() przyjmuje dwa argumenty: iterator początkowy i iterator końcowy zakresu elementów do usunięcia.

unique() jest funkcją, która usuwa duplikaty z kontenera. Funkcja ta sortuje kontener, aby duplikaty znajdowały się obok siebie, a następnie usuwa wszystkie poza pierwszym wystąpieniem każdego elementu. Funkcja unique() przyjmuje trzy argumenty: iterator początkowy i końcowy zakresu elementów oraz iterator do pierwszego elementu, który ma być usunięty.

sort() to funkcja, która służy do sortowania elementów w kontenerach, takich jak np. wektory, listy czy tablice. Funkcja przyjmuje jako argumenty dwa iteratory, które określają przedział, na którym sortowanie ma być wykonane. Sortowanie odbywa się w miejscu, czyli bez tworzenia nowej tablicy czy wektora.

Funkcja main

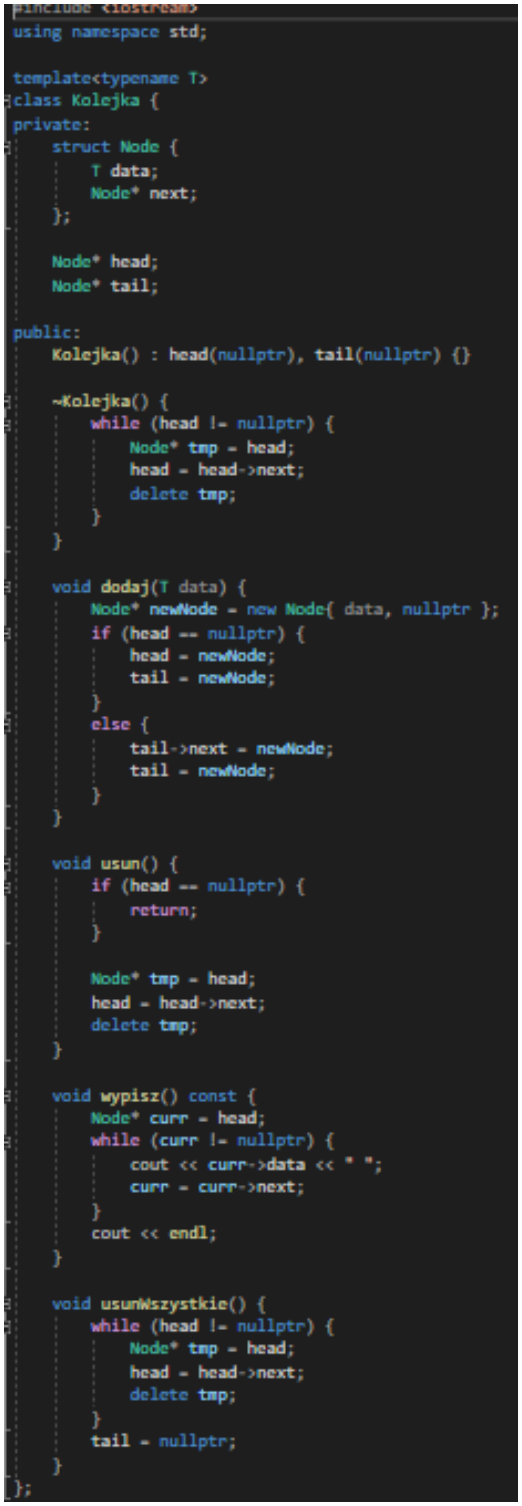
```
int main() {
    int x, p;
    string testStr;
    /*
    // Test funkcji Potega
    cout << "Podaj podstawe potegi: ";
    cin >> x;
    cout << "Podaj wykladnik potegi: ";
    cin >> p;
    cout << x << "^" << p << " = " << Potega(x, p) << endl;

    // Test funkcji Silnia
    cout << "Podaj liczbe dla ktorej chcesz policzyc silnie: ";
    cin >> x;
    cout << x << "! = " << Silnia(x) << endl;
    */
    // Test funkcji Palindrom
    cout << "Podaj lancuch znakow: ";
    getline(cin, testStr);

   /*if (jestPal(testStr, 0, testStr.length() - 1)) {
        cout << testStr << " jest palindromem." << endl;
    }
    else {
        cout << testStr << " nie jest palindromem." << endl;
    }*/

    generujPermutacje(testStr, 0, testStr.length() - 1);

    cout << "Palindromy znalezione w lancuchu " << testStr << ":" << endl;
    for (auto pal : palList) {
        cout << pal << endl;
    }

    usunDup();

    cout << "Palindromy bez duplikatow:" << endl;
    for (auto pal : palList) {
        cout << pal << endl;
    }
    return 0;
}
```

1.6. Zadanie 8

Zadanie to polegało na napisaniu obiektowo dwóch klas: Lista i Kolejka. Lista do zaimplementowania to lista jednokierunkowa, miała spełniać funkcje takie jak dodawanie elementu, usunięcie jednego lub wszystkich elementów czy wyświetlanie całej zwartości.

Kolejka miała spełniać te same funkcje co lista czyli dodawanie, usunięcie jednego lub wszystkich elementów kolejki oraz wyświetlenie zawartości kolejki.

Pliki zawierające implementacje Listy i Kolejki



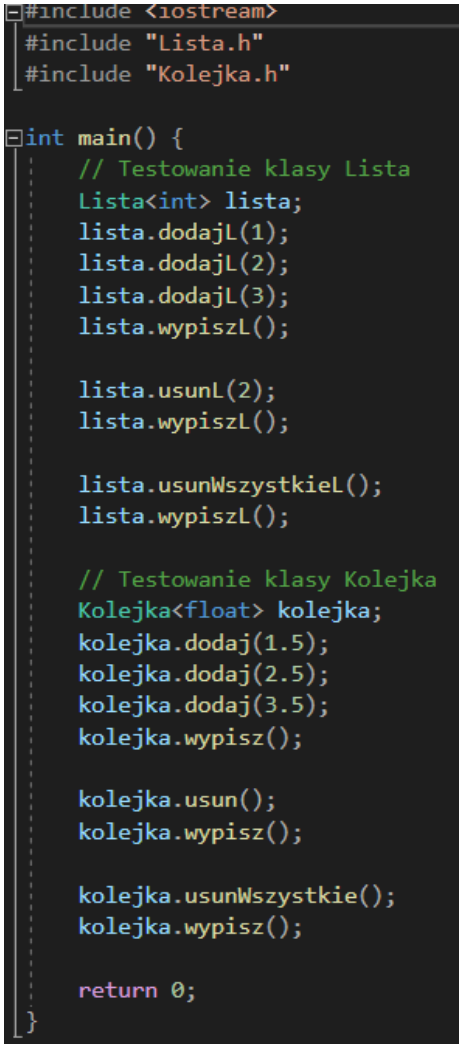
```cpp
#include <iostream>
using namespace std;

template<typename T>
class Kolejka {
private:
    struct Node {
        T data;
        Node* next;
    };

    Node* head;
    Node* tail;

public:
    Kolejka() : head(nullptr), tail(nullptr) {}

    ~Kolejka() {
        while (head != nullptr) {
            Node* tmp = head;
            head = head->next;
            delete tmp;
        }
    }

    void dodaj(T data) {
        Node* newNode = new Node{ data, nullptr };
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        }
        else {
            tail->next = newNode;
            tail = newNode;
        }
    }

    void usun() {
        if (head == nullptr) {
            return;
        }

        Node* tmp = head;
        head = head->next;
        delete tmp;
    }

    void wypisz() const {
        Node* curr = head;
        while (curr != nullptr) {
            cout << curr->data << " ";
            curr = curr->next;
        }
        cout << endl;
    }

    void usunWszystkie() {
        while (head != nullptr) {
            Node* tmp = head;
            head = head->next;
            delete tmp;
        }
        tail = nullptr;
    }
};
```

```cpp
#include <iostream>
using namespace std;

template <typename T>
class Lista {
private:
    struct Node {
        T data;
        Node* next;
    };
    Node* head;
    Node* tail;

public:
    Lista() {
        head = nullptr;
        tail = nullptr;
    }

    void dodajL(T value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        }
        else {
            tail->next = newNode;
            tail = newNode;
        }
    }

    void usunL(T value) {
        Node* currNode = head;
        Node* prevNode = nullptr;

        while (currNode != nullptr) {
            if (currNode->data == value) {
                if (prevNode == nullptr) {
                    head = currNode->next;
                }
                else {
                    prevNode->next = currNode->next;
                }
                delete currNode;
                return;
            }
            prevNode = currNode;
            currNode = currNode->next;
        }
    }

    void wypiszL() {
        Node* currNode = head;
        while (currNode != nullptr) {
            cout << currNode->data << " ";
            currNode = currNode->next;
        }
        cout << endl;
    }

    void usunWszystkieL() {
        Node* currNode = head;
        while (currNode != nullptr) {
            Node* nextNode = currNode->next;
            delete currNode;
            currNode = nextNode;
        }
        head = nullptr;
        tail = nullptr;
    }
};
```

Przedstawiona wyżej implementacja klasy szablonowej Lista, która pozwala na dodawanie, usuwanie i wypisywanie elementów listy jednokierunkowej przechowującej wartości typu jaki zadamy. Klasa ta zawiera prywatną strukturę węzła Node przechowującego dane oraz wskaźnik na kolejny węzeł. Głowica head wskazuje na początek listy, a ogon tail wskazuje na jej koniec.
Metoda dodajL dodaje nowy węzeł z wartością value na końcu listy, a metoda usunL usuwa węzeł z wartością value z listy. Metoda wypiszL wypisuje wartości wszystkich węzłów w liście, a metoda usunWszystkieL usuwa wszystkie węzły z listy.

Przedstawiona wyżej implementacja klasy szablonowej Kolejka, która pozwala na dodawanie, usuwanie i wypisywanie elementów listy jednokierunkowej przechowującej wartości typu jaki zadamy. Klasa ta zawiera prywatną strukturę węzła Node przechowującego dane oraz wskaźnik na kolejny węzeł. Głowica head wskazuje na początek kolejki, a ogon tail wskazuje na jej koniec.
Metoda dodaj dodaje nowy węzeł z wartością data na końcu kolejki, a metoda usun usuwa pierwszy węzeł z kolejki. Metoda wypisz wypisuje wartości wszystkich węzłów w kolejce, a metoda usunWszystkie usuwa wszystkie węzły z kolejki.

Funkcja Main



```cpp
#include <iostream>
#include "Lista.h"
#include "Kolejka.h"

int main() {
    // Testowanie klasy Lista
    Lista<int> lista;
    lista.dodajL(1);
    lista.dodajL(2);
    lista.dodajL(3);
    lista.wypiszL();

    lista.usunL(2);
    lista.wypiszL();

    lista.usunWszystkieL();
    lista.wypiszL();

    // Testowanie klasy Kolejka
    Kolejka<float> kolejka;
    kolejka.dodaj(1.5);
    kolejka.dodaj(2.5);
    kolejka.dodaj(3.5);
    kolejka.wypisz();

    kolejka.usun();
    kolejka.wypisz();

    kolejka.usunWszystkie();
    kolejka.wypisz();

    return 0;
}
```

2. Złożoność obliczeniowa i pamięciowa

2.1. Zadanie 1

Złożoność obliczeniowa i pamięciowa tych funkcji zależy od rozmiaru tablicy (n x m).

Funkcja alokuj_tablice() ma złożoność obliczeniową O(n * m), ponieważ tworzy tablicę o rozmiarze n x m.

Funkcja wypelnij_tablice() ma złożoność obliczeniową O(n * m), ponieważ wypełnia każdy element tablicy.

Funkcja wyswietl_tablice() ma złożoność obliczeniową O(n * m), ponieważ wyświetla każdy element tablicy.

Funkcja znajdz_max() ma złożoność obliczeniową O(n * m), ponieważ iteruje po każdym elemencie tablicy w celu znalezienia wartości maksymalnej.

Funkcja alokuj_tablice() alokuje pamięć dla tablicy o rozmiarze n x m, więc jej złożoność pamięciowa wynosi O(n * m).

Funkcja wypelnij_tablice() i znajdz_max() nie tworzą nowych zmiennych, więc ich złożoność pamięciowa jest stała O(1).

Funkcja wyswietl_tablice() wykorzystuje jedynie zmienne tymczasowe do przechowywania wartości elementów tablicy, więc jej złożoność pamięciowa również jest O(n * m).

2.2. Zadanie 2

Funkcja saveToFileText - złożoność obliczeniowa to O(n), gdzie n to liczba elementów w tablicy. Złożoność pamięciowa to O(1), ponieważ tylko jeden element jest przetwarzany w każdym kroku.

Funkcja readFromFileText - złożoność obliczeniowa to O(n), gdzie n to liczba elementów w tablicy. Złożoność pamięciowa to O(1), ponieważ tylko jeden element jest przetwarzany w każdym kroku.

Funkcja saveToFileBinary - złożoność obliczeniowa to O(n), gdzie n to liczba elementów w tablicy. Złożoność pamięciowa to O(1), ponieważ tylko jeden element jest przetwarzany w każdym kroku.

Funkcja readFromFileBinary - złożoność obliczeniowa to O(n), gdzie n to liczba elementów w tablicy. Złożoność pamięciowa to O(n), ponieważ cała tablica musi być załadowana do pamięci.

2.3. Zadanie 3

Funkcja Potega implementuje algorytm szybkiego potęgowania, który ma złożoność obliczeniową O(log n) i pamięciową O(log n) ze względu na stosowanie rekurencji.

Funkcja Silnia implementuje algorytm iteracyjny do obliczania silni, który ma złożoność obliczeniową O(n) i pamięciową O(1), ponieważ tylko jedna zmienna jest używana do przechowywania wyniku pośredniego.

2.4. Zadanie 4

Złożoność obliczeniowa tej funkcji wynosi O(n), gdzie n to długość łańcucha testStr, ponieważ funkcja wykonuje pojedyncze porównania dla każdego z n znaków łańcucha. Złożoność pamięciowa wynosi O(n), ponieważ każde wywołanie rekurencyjne funkcji dodaje nowy klatkę stosu, a w najgorszym przypadku (gdy łańcuch jest palindromem) może to prowadzić do utworzenia n klatek stosu.

2.5. Zadanie 5

Złożoność obliczeniowa funkcji generujPermutacje wynosi O(n * n!), gdzie n jest długością łańcucha str. Wynika to z faktu, że jest to algorytm typu DFS, który generuje n! permutacji, a każde wywołanie rekurencyjne wymaga przeprowadzenia operacji swap dla pozostałych n-1 znaków, co daje łącznie n*n! operacji.

Złożoność pamięciowa funkcji generujPermutacje wynosi O(n), ponieważ używane są tylko zmienne przechowujące informacje o pozycji kolejnych znaków w ciągu.

Funkcja usunDup wykonuje operacje sortowania i usuwania duplikatów w wektorze palList. Złożoność obliczeniowa sortowania to O(n log n), gdzie n to liczba elementów w wektorze. Usuwanie duplikatów za pomocą funkcji unique ma złożoność obliczeniową O(n), gdzie n to liczba elementów w wektorze. W sumie złożoność obliczeniowa funkcji usunDup wynosi O(n log n), ponieważ sortowanie jest dominującą operacją.
Złożoność pamięciowa funkcji usunDup zależy od implementacji sortowania i usuwania duplikatów. Jednak w ogólności można przyjąć, że wymaga ona dodatkowej pamięci na przechowanie posortowanego wektora, co daje złożoność pamięciową O(n).

2.6. Zadanie 8

Złożoność obliczeniowa:
dodajL: O(1)
usunL: w najgorszym przypadku O(n), gdzie n to liczba elementów w liście
wypiszL: O(n), gdzie n to liczba elementów w liście
usunWszystkieL: O(n), gdzie n to liczba elementów w liście

Dodawanie elementu do kolejki: O(1)
Usuwanie elementu z kolejki: O(1)
Wypisanie zawartości kolejki: O(n), gdzie n to liczba elementów w kolejce
Usuwanie wszystkich elementów z kolejki: O(n), gdzie n to liczba elementów w kolejce

Złożoność pamięciowa:
dodajL: O(1)
usunL: O(1)
wypiszL: O(1)
usunWszystkieL: O(1)

Pamięć potrzebna na przechowywanie elementów kolejki: O(n), gdzie n to liczba elementów w kolejce, ponieważ każdy element zajmuje pewną ilość pamięci.

3. Wnioski

Złożoność obliczeniowa i pamięciowa są ważnymi wskaźnikami wydajności algorytmów i programów. Algorytmy o niższej złożoności obliczeniowej i pamięciowej są zazwyczaj bardziej wydajne dla większych zbiorów danych, ale wymagają dużej liczby obliczeń i pamięci dla mniejszych zbiorów danych. W przypadku algorytmów o wysokiej złożoności obliczeniowej, efektywne wykorzystanie pamięci może pomóc zminimalizować koszty obliczeniowe.