WYDZIAŁ
INFORMATYKI
I TELEKOMUNIKACJI

listy

Veronika Tronchuk

I rok Informatyki

Nr albumu: 30019

Lista to

Lista to struktura danych, która pozwala przechowywać sekwencję elementów w pamięci komputera. W przeciwieństwie do tablicy, jej rozmiaru nie trzeba określać z góry. Lista automatycznie się powiększa w miarę dodawania nowych elementów. Jest to bardzo wygodne, gdy liczba danych jest nieznana z wyprzedzeniem lub często się zmienia.

Różnica między tablicą a listą

Tablica to blok pamięci, w którym elementy są ułożone jeden po drugim. Jej długość zazwyczaj ustala się z góry i trudno ją zmienić w trakcie działania programu. Aby dodać element do środka tablicy, trzeba przesunąć wszystkie kolejne elementy.

Lista działa inaczej. Każdy element to oddzielna część składająca się z dwóch elementów: wartości i wskaźnika na następny element. Dzięki temu nowe elementy można łatwo dodawać lub usuwać bez przesuwania pozostałych.

Lista jednokierunkowa

Lista jednokierunkowa składa się z węzłów. Każdy węzeł zawiera wartość oraz adres następnego węzła. W ten sposób wszystkie elementy są połączone ze sobą w łańcuch. Ostatni węzeł nie posiada następnego — jego wskaźnik ma wartość nullptr, co oznacza koniec listy.


```
#include <iostream>
#include <cstdlib>
#include <ctime>
```

#include <iostream> pozwala używać cout i cin (wejście/wyjście)

#include <cstdlib> pozwala używać rand() i srand() — generowanie liczb losowych

#include <ctime> pozwala pobrać aktualny czas — time(0)

using namespace std; żeby nie pisać std::cout, std::cin itd.

```
struct Node {
    int data;
    Node* next;
};
```

`int data;`

przechowuje wartość węzła

`Node* next;`

wskaźnik na następny węzeł na liście.

Węzeł przechowuje liczbę i wskaźnik na następny. To podstawa listy jednokierunkowej.

```
void pushFront(Node*& head, int value) {
    Node* newNode = new Node{ value, head };
    head = newNode;
}
```

`Node*& head,`

przekazujemy referencję do głowy listy, aby można było ją zmienić

`new Node{ value, head };`

tworzymy nowy węzeł, gdzie `value` to liczba, a `head` to poprzednia głowa

`head = newNode;`

nowy węzeł staje się nową głową listy.

Funkcja dodaje nowy element na początek listy.

```
void pushBack(Node*& head, int value) {
    Node* newNode = new Node{ value, nullptr };
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next)
        temp = temp->next;
    temp->next = newNode;
}
```

```
new Node{ value, nullptr };
```

nowy węzeł nie ma kolejnego – to będzie ostatni

```
if (!head) {
    head = newNode;
    return;
}
```

jeśli lista jest pusta – nowy węzeł staje się głową.

```
while (temp->next)
    temp = temp->next;
temp->next = newNode;
```

przechodzimy na koniec listy.

```
temp->next = newNode;
```

dodajemy węzeł na końcu.

Funkcja dodaje nowy element na koniec listy.

```
void popFront(Node*& head) {
    if (!head) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}
```

```
if (!head) return;
```

jeśli lista jest pusta — nic nie robimy.

```
Node* temp = head;
```

zapisujemy obecną głowę do zmiennej tymczasowej.

```
head = head->next;
```

przesuwamy głowę na kolejny węzeł.

```
delete temp;
```

usuwamy starą głowę, zwalniamy pamięć.

Usuwa pierwszy element listy (jeśli istnieje).

```
void popBack(Node*& head) {
    if (!head) return;
    if (!head->next) {
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while (temp->next->next)
        temp = temp->next;
    delete temp->next;
    temp->next = nullptr;
}
```

```
if (!head) return;
```

jeśli lista pusta — wyjście.

```
if (!head->next) {
    delete head;
    head = nullptr;
    return;
}
```

jeśli w liście jest tylko jeden element — usuwamy go.

```
while (temp->next->next)
    temp = temp->next;
```

przechodzimy do przedostatniego węzła.

```
delete temp->next;
temp->next = nullptr;
```

usuwamy ostatni węzeł, poprzedni staje się ostatnim.

Usuwa ostatni element listy.

```
void printList(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << "->";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}
```

```
while (temp) {
    cout << temp->data << "->";
    temp = temp->next;
}
```

dopóki nie dotrzemy do końca listy.

```
cout << temp->data << "->";
```

wypisujemy wartość węzła i strzałkę.

```
cout << "NULL" << endl;
```

pokazujemy, że lista się kończy.

Wypisuje listę w formacie `5 -> 10->15 -> NULL`

```
int countNodes(Node* head) {
    int count = 0;
    while (head) {
        count++;
        head = head->next;
    }
    return count;
}
```

```
int count = 0;
```

zmienna do liczenia.

```
while (head) {
    count++;
    head = head->next;
}
return count;
```

przechodzimy listę, dopóki się nie skończy.

```
count++;
```

dodajemy +1 do licznika dla każdego węzła.

Zwraca ilość elementów w liście.

```
Node* findNode(Node* head, int value) {
    while (head) {
        if (head->data == value)
            return head;
        head = head->next;
    }
    return nullptr;
}
```

```
if (head->data == value)
    return head;
```

jeśli wartość się zgadza — zwracamy węzeł.

```
    head = head->next;
```

w przeciwnym razie — idziemy dalej.

```
return nullptr;
```

jeśli nie znaleziono — zwracamy pusty wskaźnik.

Zwraca wskaźnik na węzeł o danej wartości lub `nullptr`

```
int main() {
    srand(time(0));
    Node* head = nullptr;
```

stawia ziarno (seed) dla generatora liczb losowych, żeby `rand()` dawał inne wartości za każdym razem.

Inicjalizuje generator liczb losowych.

```
Node* head = nullptr;
```

tworzymy pustą listę (wskaźnik na głowę = nic).

Lista jest pusta – jeszcze nie zawiera elementów.

```
pushFront(head, 10);
pushBack(head, 20);
pushFront(head, 5);
```

dodajemy trzy liczby do listy, testujemy dodawanie z przodu i z tyłu.

Lista po tych komendach: `5 -> 10 -> 20`

```
cout << "Po dodaniu elementow: ";
printList(head);
```

Wypisujemy listę po dodaniu.

Sprawdzamy, czy elementy zostały poprawnie dodane.

```
popFront(head);
popBack(head);
```

Usuwamy pierwszy i ostatni element:
Lista `5 -> 10 -> 20` ------> `10`

```
cout << "Po usunieciach: ";
printList(head);
```

Wypisujemy listę po usunięciach.

Sprawdzamy, czy popFront i popBack działają.

```
int n;
cout << "Podaj liczbe wezlow: ";
cin >> n;
```

Użytkownik wpisuje, ile węzłów losowych dodać.

```
for (int i = 0; i < n; i++) {
    pushBack(head, rand() % 100);
}
```

Generujemy n liczb losowych i dodajemy je na koniec listy.

Tworzymy listę z losowymi wartościami.

```
cout << "Lista z losowymi wartosciami: ";
printList(head);
```

Wypisujemy całą listę po wygenerowaniu.

```
int counted = countNodes(head);
cout << "Liczba wezlow: " << counted << endl;
```

Liczymy ile węzłów znajduje się na liście.

Sprawdzamy, czy naprawdę mamy n elementów.

```
int target;
cout << "Podaj wartosc do wyszukania: ";
cin >> target;
```

Użytkownik wpisuje liczbę do wyszukania.

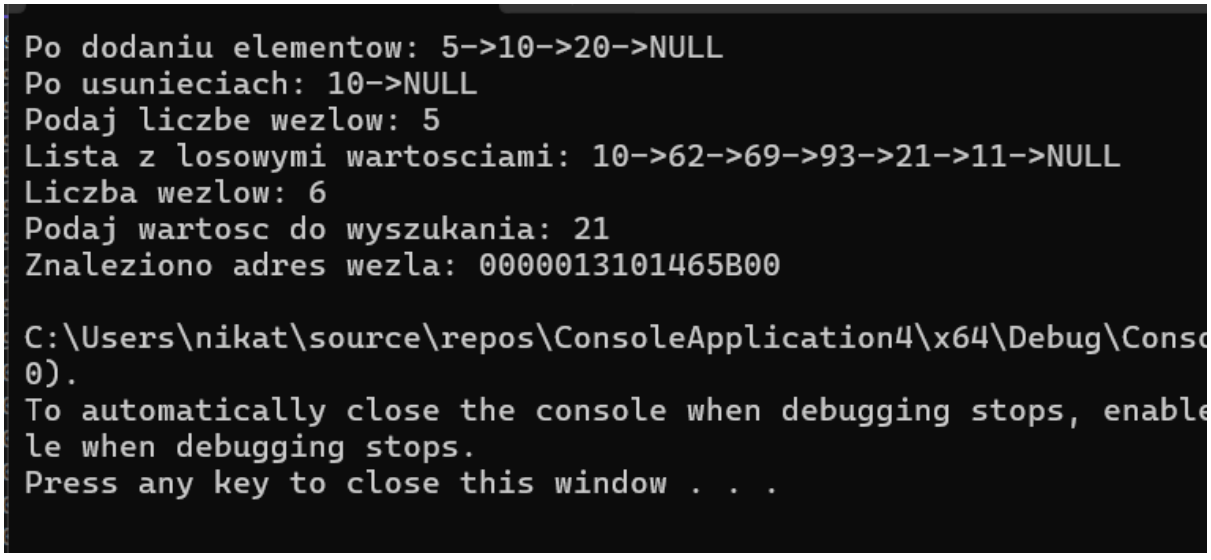
```
Node* found = findNode(head, target);
if (found)
    cout << "Znaleziono adres wezla: " << found << endl;
else
    cout << "Nie znaleziono wartosci w liscie." << endl;

return 0;
```

Jeśli znaleziono — wypisujemy adres. W przeciwnym razie — komunikat.

Szukanie wartości na liście.

W `main()` testujemy wszystkie funkcje: dodawanie, usuwanie, wypisywanie, generowanie, zliczanie i szukanie.



Wnioski

Program dodaje elementy na początek i koniec listy, potrafi je usuwać, pokazuje zawartość listy, zlicza liczbę węzłów i umożliwia wyszukiwanie konkretnej wartości. Po kilku testach wszystko wygląda stabilnie i logicznie.

Jest kilka rzeczy, które można by ulepszyć. Na przykład, zamiast wyświetlać adres węzła podczas wyszukiwania, lepiej byłoby pokazać jego pozycję. Przydałaby się też funkcja, która usuwa całą listę po zakończeniu programu, żeby nie zostawały niezwolnione dane w pamięci. Poza tym, program nie sprawdza, czy użytkownik wpisuje poprawne dane — np. czy naprawdę podał liczbę.

Lista dwokierunkowa

Lista dwukierunkowa to struktura, w której każdy element ma dwa wskaźniki:

- Next – na następny,
- `Prev` – na poprzedni

```cpp
struct DNode {
    int data;
    DNode* prev;
    DNode* next;

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

```cpp
int data;
```

wartość przechowywana w węźle

```cpp
DNode* prev;
```

wskaźnik na poprzedni węzeł

```cpp
DNode* next;
```

wskaźnik na następny węzeł

```cpp
DNode(int val) : data(val), prev(nullptr), next(nullptr) {}
```

konstruktor — ustawia wartość i puste wskaźniki

```cpp
void pushFront(DNode*& head, int val) {
    DNode* newNode = new DNode(val);
    if (head) head->prev = newNode;
    newNode->next = head;
    head = newNode;
}
```

```cpp
DNode* newNode = new DNode(val);
```

tworzymy nowy węzeł

```cpp
if (head) head->prev = newNode;
```

jeśli lista nie jest pusta, ustawiamy prev

```cpp
newNode->next = head;
```

nowy węzeł wskazuje na głowę

```
head = newNode;
```

głową staje się nowy węzeł

```
void popBack(DNode*& head) {
    if (!head) return;
    DNode* temp = head;
    while (temp->next) temp = temp -> next;
    if (temp->prev) temp -> prev -> next = nullptr;
    else head = nullptr;
    delete temp;
}
```

```
if (!head) return;
```

jeśli lista jest pusta — nic nie robimy

```
DNode* temp = head;
while (temp->next) temp = temp -> next;
```

idziemy do ostatniego węzła

```
if (temp->prev) temp -> prev -> next = nullptr;
```

poprzedni staje się ostatnim

```
else head = nullptr;
```

w przeciwnym razie lista jest pusta

```
delete temp;
```

usuwamy ostatni węzeł

```
void printList(DNode* head) {
    while (head) {
        cout << head->data << " <-> ";
        head = head->next;
    }
    cout << "NULL" << endl;
```

```
while (head) {
    cout << head->data << " <-> ";
```

wypisujemy wartość węzła

```
head = head->next;
```

przechodzimy do następnego

```
cout << "NULL" << endl;
```

koniec listy

```
DNode* head = nullptr;
```

tworzymy pustą listę

```
pushFront(head, 30);
pushFront(head, 20);
pushFront(head, 10);
```

dodajemy 30, 20, 10

```
cout << "Po dodaniu: ";
printList(head);
```

wynik: Po dodaniu: 10 <-> 20 <-> 30 <-> NULL

```
popBack(head);
```

usuwamy ostatni element (30)

```
cout << "Po usunieciu z konca: ";
printList(head);
```

wynik: Po usunieciu z konca: 10 <-> 20 <-> NULL

```
Po dodaniu: 10 <-> 20 <-> 30 <-> NULL
Po usunieciu z konca: 10 <-> 20 <-> NULL
```

Wnioski

Po zakończeniu działania programu lista dwukierunkowa nie jest całkowicie zwalniana z pamięci. Nie została zaimplementowana obsługa błędów podczas wprowadzania danych, co powoduje, że podanie nieprawidłowej wartości, np. litery, prowadzi do błędu wykonania. Wypisywanie zawartości listy dwukierunkowej działa tylko w jednym kierunku, bez możliwości przeglądania jej od końca. Dostępne są wyłącznie operacje

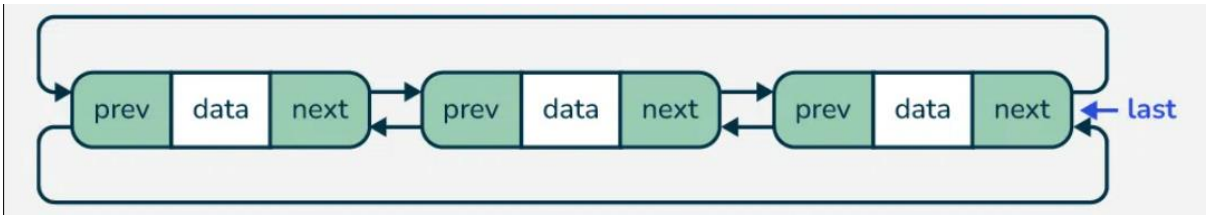
dodawania i usuwania elementów na początku i na końcu listy, bez opcji edycji elementów w jej środku. Cała logika działania listy została zaimplementowana przy użyciu funkcji globalnych, bez wykorzystania programowania obiektowego ani klasy zarządzającej strukturą listy.

Lista cykliczna

Lista cykliczna to taka struktura, w której ostatni element wskazuje z powrotem na pierwszy, a nie na nullptr

Lista cykliczna jednokierunkowa:

Lista cykliczna, dwokierunkowa:



```
struct Node {
    int data;
    Node* next;
};
```

Struktura jednego elementu listy:

```
 int data;
```

przechowywana wartość

```
 Node* next;
```

wskaźnik na następny element

```
void pushBack(Node*& head, int value) {
    Node* newNode = new Node{ value, nullptr };
```

```
(Node*& head,
```

referencję do głowy listy

```
int value)
```

wartość do dodania

```
  Node* newNode = new Node{ value, nullptr };
```

Tworzy nowy węzeł w pamięci dynamicznej z daną wartością.
Na razie next ustawiony na nullptr

```
 if (!head) {
     head = newNode;
     newNode->next = head;
     return;
 }
```

Jeśli lista jest pusta:

Head wskazuje na nowy węzeł, domykamy cykl:

```
newNode->next = head;
return;
```

```
 Node* temp = head;
```

Pomocniczy wskaźnik do przejścia do końca listy

```
while (temp->next != head)
    temp = temp->next;
```

Pętla szukająca ostatniego elementu, który wskazuje na head

```
temp->next = newNode;
newNode->next = head;
```

Dodajemy nowy węzeł i domykamy cykl ponownie

```
void popFront(Node*& head)
```

Usuwa pierwszy element z listy cyklicznej

```
if (!head) return;
```

Jeśli lista jest pusta – kończymy funkcję

```
if (head->next == head) {
    delete head;
    head = nullptr;
    return;
}
```

Jeśli lista zawiera tylko jeden element to usuwamy go i ustawiamy

```
Node* temp = head;
Node* tail = head;
```

Temp – do usunięcia

Tail– by znaleźć ostatni węzeł

```
while (tail->next != head)
    tail = tail->next;
```

Szukamy ostatniego elementu listy

```
head = head->next;
tail->next = head;
delete temp;
```

Zmieniamy head domykamy cykl i usuwamy poprzedni pierwszy element

```
void printCyclicList(Node* head)
```

Wypisuje wszystkie wartości z listy cyklicznej

```
do {
    cout << temp->data << " -> ";
    temp = temp->next;
} while (temp != head);
```

Wypisuje każdy element, aż wróci do head

```
Node* head = nullptr;
```

Tworzymy pustą listę

```
pushBack(head, 10);
pushBack(head, 20);
pushBack(head, 30);
```

Dodajemy liczby do listy: 10 20 30

```
cout << "Po dodaniu elementow: ";
printCyclicList(head);
cout << "\n";
```

Wypisujemy listę po dodaniu elementów

```
popFront(head);
```

Usuwamy pierwszy węzeł

```
cout << "Po usunieciu pierwszego: ";
printCyclicList(head);
```

wypisujemy listę jeszcze raz

```
Po dodaniu elementow: 10 -> 20 -> 30 ->
Po usunieciu pierwszego: 20 -> 30 ->
```

Wnioski

Dane wprowadzane przez użytkownika nie są sprawdzane, więc błędne wartości mogą powodować błędy działania. Usunięcie pierwszego elementu wymaga przejścia przez całą listę, co obniża wydajność przy dużej liczbie węzłów. Zaimplementowano tylko podstawowe operacje na początku i końcu listy. Cała logika oparta jest na funkcjach globalnych bez wykorzystania podejścia obiektowego.