Lekcja 26 - Lista jednokierunkowa - struktura języka C++

Temat: Omówienie listy jednokierunkowej jako dynamicznej struktury danych języków programowania.

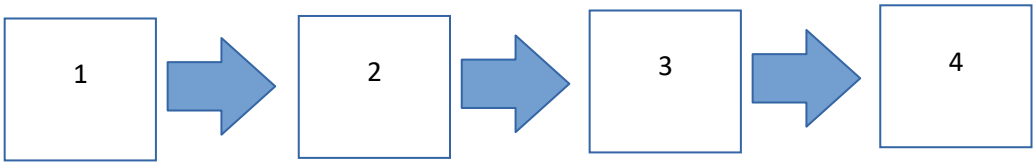
Kod programu źródłowego:

```
/*****************************************************************
lista jednokierunkowa
*****************************************************************/
#include <iostream>

using namespace std;

struct Element
{
    int numer;      //klucz listy
    Element *Enext; //wskaznik nastepnego elementu
};

void DodajElement (Element *&glowa, int num)
{               //utworzenie nowego elementu
    Element *nowy = new Element;
    nowy->numer = num;
    nowy->Enext = NULL;

        //jesli lista jest pusta tworzymy glowe
    if (glowa == NULL)
    {
        glowa = nowy;
        cout <<"glowa" << endl;
        return;
    }
    else
    {       //utworzenie tmp, zeby nie zmienic glowy
        Element *tmp = glowa;
            //nowy wskazuje na to samo co tmp - na glowe
        nowy->Enext = tmp->Enext;
            //tmp wskazuje na nowy
        tmp->Enext = nowy;
        cout <<"kolejny "<< nowy->numer <<" "<< nowy->Enext << endl;
    }
}

void wypisz( Element *glowa ) //wypisuje liste elementow
{
    while( glowa != NULL)   //dopoki nie ma konca listy
    {
        cout << glowa->numer << " ";    //wypisanie zawartosci
        glowa = glowa->Enext;   //przejscie do nastepnego elementu
    }
}

void skasuj ( Element *&glowa)
```

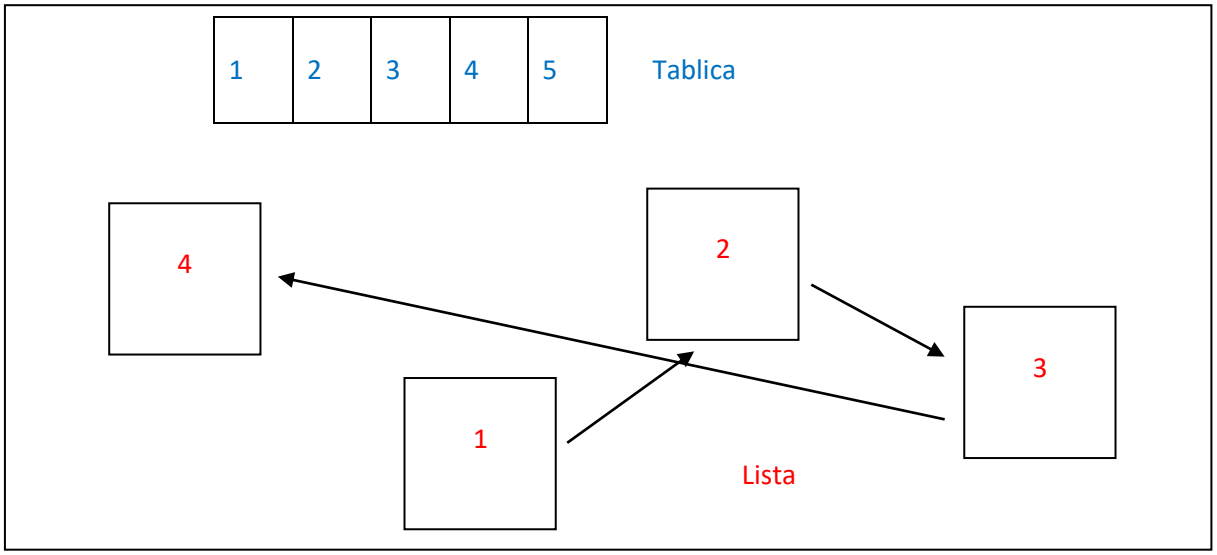
```
{
    if( glowa == NULL)
    {
        cout<<"Lista jest pusta " << endl;
        return;
    }
        //dy lista liczy tylko 1 element
     if( glowa->Enext == NULL )
    {       //glowa = NULL
        glowa = glowa->Enext;
        delete glowa;
        return;
    }

    Element *tmp = glowa;
    if (tmp->Enext != NULL)
    {       //kasowanie pierwszego elementu
        glowa = tmp->Enext;
        delete tmp;
        return;
    }
}

int main()
{
    int i, num = 99;
            //utworzenie pustego elementu glowa - to poczatek listy
    Element *glowa = NULL;

    cout<<"Lista jednokierunkowa" << endl;
    skasuj (glowa);

    DodajElement (glowa, num);
    wypisz (glowa);
    cout << endl;
    DodajElement (glowa, num);
    wypisz (glowa);
    cout << endl;
    for (i=1; i <21; i++)
    {
        DodajElement (glowa, i);
    }

    wypisz (glowa);
    cout << endl;
    for (i=0; i < 10; i++)
    {
        skasuj (glowa);
        wypisz (glowa);
        cout << endl;
    }
    return 0;
}
```

Opis listy: Lista to dynamiczna struktura danych, w której dane są poukładane liniowo, a której rozmiar jest zmienny.

Można ją opisać jako uszeregowany zbiór elementów. Każdy element zawiera jakieś dane oraz wskazuje na swojego następcę. Cechą listy jednokierunkowej jest to, że można przeglądać ją tylko w jedną stronę, od początku do końca.

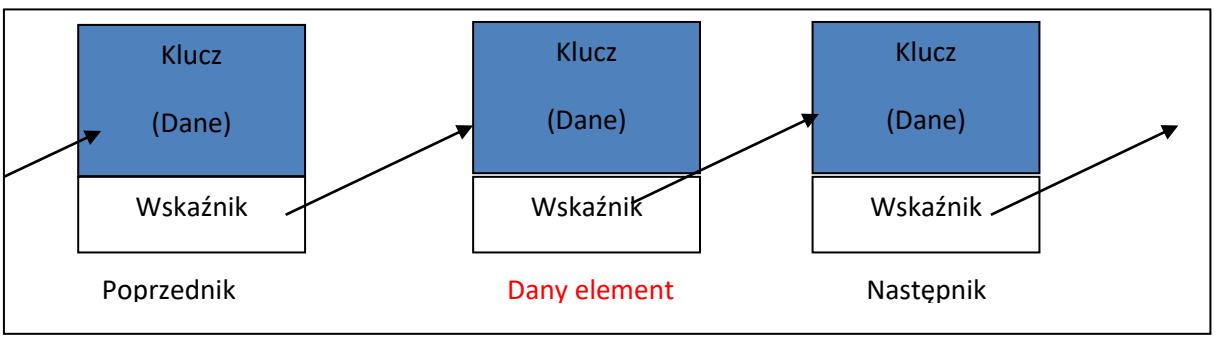


Lista jednokierunkowa jest rozrzucona po pamięci aplikacji. Jej elementy nie występują kolejno po sobie. Element poprzedni wskazuje na element następny. Dlatego tak ważne jest przypisywanie odpowiednich wskaźników do nowych elementów. To odróżnia ją od tablicy, gdzie elementy są zebrane w jednym bloku pamięci.



Struktura elementów listy:

Poszczególne elementy składają się minimum z dwóch pól: klucza i wskaźnika do kolejnego elementu listy. Klucz jest polem identyfikującym dany element listy.

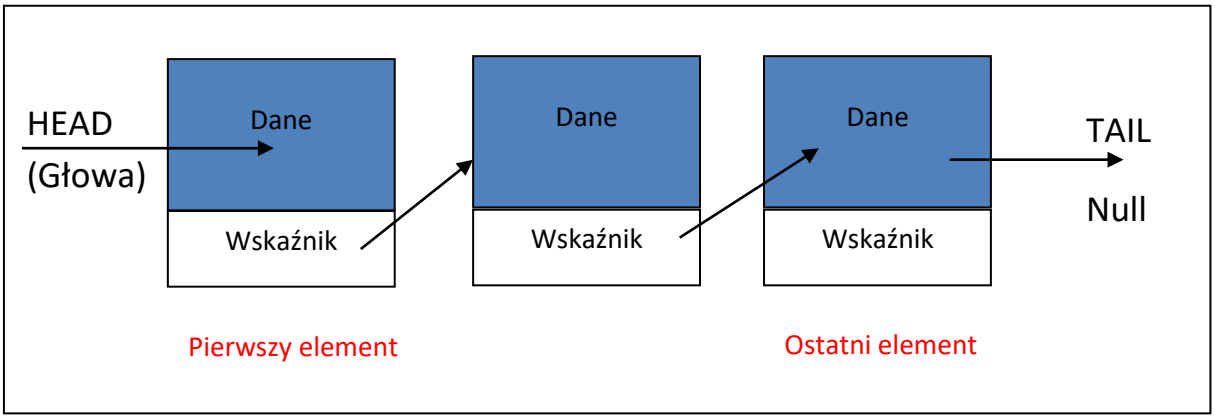


Budowa listy:

Początek listy jest wskazywany przez znacznik HEAD. Jest to wskaźnik (nie element!), który wskazuje na pierwszy element listy.

Wskaźnik pierwszego elementu wskazuje na drugi, ten na trzeci itd. Kolejne elementy są wskazywane przez wskaźniki poprzedników.

Ostatni element wskazuje na NULL. Oznacza to koniec listy. Dalej nie ma już nic.



Zastosowanie:

Listy są preferowaną strukturą danych w sytuacjach, gdy nie wiadomo, ile elementów docelowo będzie przechowywanych. Pozwalają na szybkie dodawanie i usuwanie elementów.

Lista nie rezerwuje z góry zadanej ilości miejsca w pamięci. To powoduje lepsze zarządzanie dostępną pamięcią.

Elementy listy mogą być umieszczone w różnych miejscach pamięci – nie trzeba rezerwować dla niej z góry ustalonego, ciągłego obszaru (tak jak dla tablic). W rezultacie do listy można dodawać nowe elementy.

Gdy zachodzi potrzeba odwołać się do wybranego elementu, konieczne jest przejście kolejno przez wszystkie elementy go poprzedzające. Cecha ta jest rezultatem sekwencyjnej budowy listy, a zatem jej elementy nie są oznaczone za pomocą indeksów (tak jak w przypadku tablic), lecz odwołują się do siebie kolejno (obiekt A wskazuje obiekt B, obiekt B wskazuje obiekt C i tak dalej).

Operacje na liście:

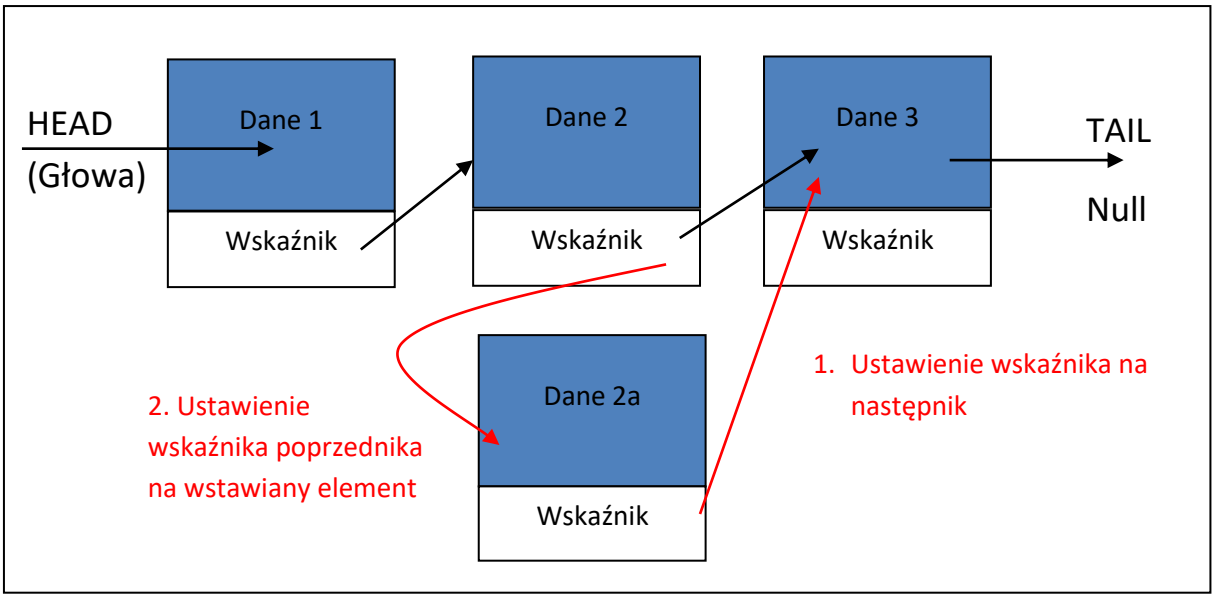
- dodanie elementu do listy;
  - dodanie w środku listy
  - dodanie na początku listy
  - dodanie na końcu listy
- Usunięcie elementu z listy;
  - Usunięcie ze środka listy

- Usunięcie z początku listy
- Usunięcie z końcu listy

- Przejście przez listę

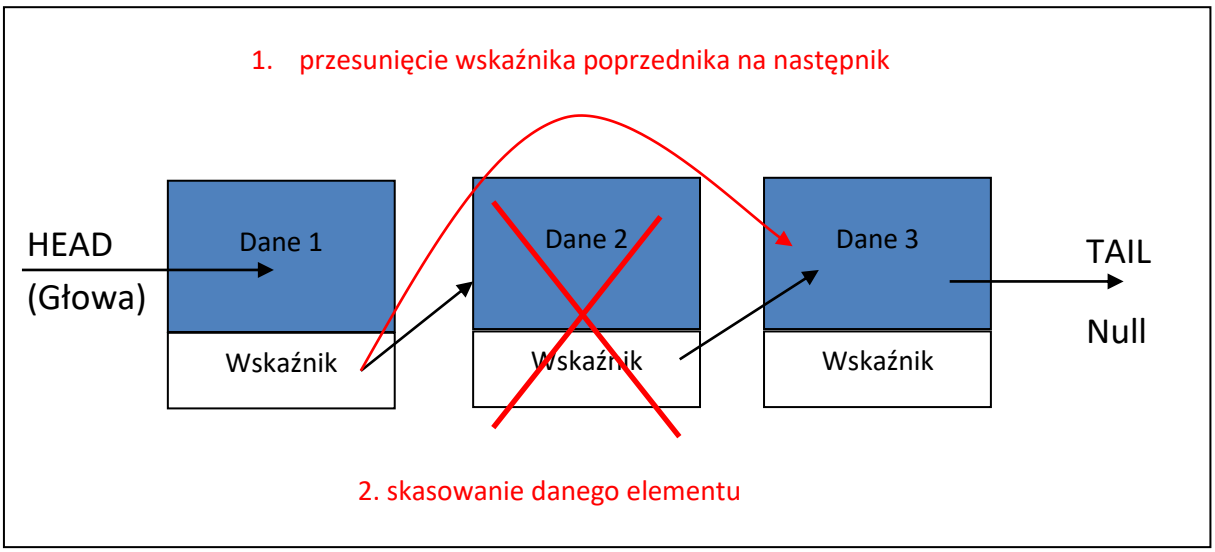
Dodawanie elementu:

Chcąc dodać element w środku listy (pomiędzy poprzednik i następnik), należy w pierwszym kroku ustawić wskaźnik dodawanego obiektu na następnik. Następnie wskaźnik poprzednika ustawiamy na element wstawiany.



Usuwanie elementu:

Chcąc usunąć dany element ze środka listy, należy w pierwszym kroku zmienić wskaźnik poprzednika z danego elementu na następnik. Następnie dany element można usunąć.



Ćwiczenie:

1. Napisz program tworzący listę i wpisujący do niej kolejne potęgi dwójki.
2. Napisz program tworzący listę i wpisujący do niej kolejne liczby ciągu Fibonacciego.
3. Napisz funkcję zliczającą liczbę elementów w liście.
4. Napisz program wyszukujący dany element w liście lub informujący, że go brak.

Zadanie 26

1. Napisz program, który umożliwia dodawanie elementów do listy według wyboru:
   a) Z początku listy
   b) Na końcu listy
   c) W środku listy według zadanego klucza.
2. Napisz program, który umożliwia kasowanie elementów z listy według wyboru:
   d) Z początku listy
   e) Na końcu listy
   f) W środku listy według zadanego klucza.
3. Napisz program, który sprawdza, czy w liście nie występują elementy zdublowane. Jeśli takie znajdzie wyświetli je i skasuje nadmiarowe wystąpienia (pozostawi pierwsze z nich).
4. Napisz program, który wczytuje liczby **n** i **m**. następnie wyświetli elementy listy o kluczach zawartych między **n** i **m**.
5. Utwórz program, który tworzy listę zawierającą katalog książek w bibliotece. Każda pozycja zawiera: tytuł książki, autora, liczbę stron, cenę, rok wydania. Dodaj funkcje dodawania książek, usuwania, przeszukiwania według autora, tytułu, daty wydania.