

Jan Danowski

Politechnika Rzeszowska

Algorytmy i struktury danych

Sprawozdanie

Projekt 3

Opis problemu

Dokonaj implementacji struktury danych typu lista jednokierunkowa wraz w wszelkimi potrzebnymi operacjami charakterystycznymi dla tej struktury (inicjowanie struktury, dodawanie/usuwanie elementów, wyświetlanie elementów, zliczanie elementów/wyszukiwanie zadanego elementu itp.)

Opis podstaw teoretycznych

Lista jednokierunkowa jest strukturą o dynamicznie zmieniającej się wielkości. Listę można 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.

Funkcje programu

1. Inicjowanie struktury

```
int main( )
{
    slist L;      // zawiera adres początku listy
    slistEl * e;  // do wskazywania elementów listy
    int i;

    for( i = 1; i <= 7; i++ ) L.push_back ( i );
    L.printl( );

    // Przechodzimy do elementu o wartości 4 w naszym przypadku czyli e

    e = L.head;

    for( i = 1; i <= 3; i++ ) e = e->next;

    //dodawanie przed
    L.insert_before ( e, i );
    //dodawanie po
    L.insert_after ( e, i );

    L.printl( );

    // Usuamy element 4 oraz element pierwszy i ostatni

    L.remove ( e );
    L.pop_front();
    L.pop_back( );
    L.printl( );

    return 0;
}
```

Rys. 1 inicjowanie funkcji programu

2. Przechodzenie przez listę i liczenie jej elementów

K01: $c \leftarrow 0$

K02: Dopóki p ,

wykonuj kroki K03...K04

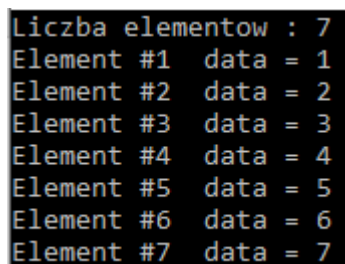
K03: $c \leftarrow c + 1$ zwiększ licznik

K04: $p \leftarrow (p \rightarrow \text{next})$

K05: Zakończ z wynikiem c

```
unsigned slist::size( )
{
    unsigned c = 0;
    slistEl * p = head;
    while( p )
    {
        c++;
        p = p->next;
    }
    return c;
}
```

Rys. 2 funkcja do przechodzenie i liczenia elementów



```
Liczba elementow : 7
Element #1 data = 1
Element #2 data = 2
Element #3 data = 3
Element #4 data = 4
Element #5 data = 5
Element #6 data = 6
Element #7 data = 7
```

Rys. 3 Wynik programu

3. Dołączanie elementu na początek listy

K01: Utwórz nowy element listy

K02: $p \leftarrow$

K03: $(p \rightarrow \text{data}) \leftarrow v$

K04: $(p \rightarrow \text{next}) \leftarrow \text{head}$

K05: $\text{head} \leftarrow p$

K06: Zakończ

```

void slist::push_front ( int v )
{
    slistEl * p;

    p = new slistEl;
    p->next = head;
    p->data = v;
    head = p;
}

```

Rys. 4 funkcja dołączania elementu

4. Dołączanie na koniec listy

K01: Utwórz nowy element

K02: $e \leftarrow$

K03: $(e \rightarrow \text{next}) \leftarrow \text{null}$

K04: $(e \rightarrow \text{data}) \leftarrow v$

K05: $p \leftarrow \text{head}$

K06: Jeśli p ,

to idź do kroku K09

K07: $\text{head} \leftarrow e$

K08: Zakończ

K09: Dopóki $(p \rightarrow \text{next}) \neq \text{null}$,

wykonuj $p \leftarrow (p \rightarrow \text{next})$

K10: $(p \rightarrow \text{next}) \leftarrow e$

K11: Zakończ

```

void slist::push_back ( int v )
{
    slistEl * p, * e;

    e = new slistEl; // tworzymy nowy element
    e->next = NULL; // inicjujemy element
    e->data = v;
    p = head;
    if( p )
    {
        while( p->next ) p = p->next;
        p->next = e;
    }
    else head = e;
}

```

Rys. 5 funkcja dołączania elementu na koniec listy

5. Usuwanie elementu z początku listy

K01: $p \leftarrow \text{head}$ zapamiętaj pierwszy element

K02 Jeśli $p = \text{null}$,

to zakończ

K03: $\text{head} \leftarrow (p \rightarrow \text{next})$

K04: Usuń z pamięci element wskazany przez p

K05: Zakończ

```

void slist::pop_front( )
{
    slistEl * p = head; // zapamiętujemy początek

    if( p )
    {
        head = p->next; // nowy początek
        delete p;       // usuń element z pamięci
    }
}

```

Rys. 6 funkcja do usuwania elementu z końca listy

```

Liczba elementow : 6
Element #1  data = 2
Element #2  data = 3
Element #3  data = 4
Element #4  data = 4
Element #5  data = 5
Element #6  data = 6

```

Rys. 7 usunięty element

6. Usuwanie ostatniego elementu listy

K01: $p \leftarrow \text{head}$
K02: Jeśli $p = \text{null}$,
to zakończ
K03: Jeśli $(p \rightarrow \text{next}) \neq \text{null}$,
to idź do kroku K07
K04: Usuń z pamięci element wskazywany przez p
K05: $\text{head} \leftarrow \text{null}$
K06: Zakończ
K07: Dopóki $((p \rightarrow \text{next}) \rightarrow \text{next}) \neq \text{null}$,
wykonuj $p \leftarrow (p \rightarrow \text{next})$
K08: Usuń z pamięci element wskazywany przez $(p \rightarrow \text{next})$
K09: $(p \rightarrow \text{next}) \leftarrow \text{null}$
K10: Zakończ

```
void slist::pop_back( )
{
    slistEl * p = head; // zapamiętujemy początek

    if( p )
    {
        if( p->next )
        {
            while( p->next->next ) p = p->next;
            delete p->next;
            p->next = NULL;
        }
        else
        {
            delete p;
            head = NULL;
        }
    }
}
```

Rys. 8 funkcja usuwania ostatniego elementu

```
Liczba elementow : 6
Element #1  data = 2
Element #2  data = 3
Element #3  data = 4
Element #4  data = 4
Element #5  data = 5
Element #6  data = 6
```

Rys. 9 usunięty element 7

7. Usuwanie wybranego elementu z listy

K01: Jeśli $\text{head} \neq e$,

to idź do kroku K04

K02: Usuń pierwszy element listy jeśli tak, usuwamy go z listy

K03: Zakończ

K04: $p \leftarrow \text{head}$

K05: Dopóki $(p \rightarrow \text{next}) \neq e$,

wykonuj $p \leftarrow (p \rightarrow \text{next})$

K06: $(p \rightarrow \text{next}) \leftarrow (e \rightarrow \text{next})$

K07: Usuń z pamięci element wskazywany przez e

K08: Zakończ

```
// Usuwanie wybranego elementu

void slist::remove ( slistEl * e )
{
    slistEl * p;

    if( head == e ) pop_front( );
    else
    {
        p = head;
        while( p->next != e ) p = p->next;
        p->next = e->next;
        delete e;
    }
}
```

Rys. 10 usuwanie wybranego elementu

```
Liczba elementow : 7
Element #1 data = 1
Element #2 data = 2
Element #3 data = 3
Element #4 data = 4
Element #5 data = 5
Element #6 data = 6
Element #7 data = 7
```

Rys. 11 przed usunięciem

```
Liczba elementow : 6
Element #1 data = 2
Element #2 data = 3
Element #3 data = 4
Element #4 data = 4
Element #5 data = 5
Element #6 data = 6
```

Rys. 12 po usunięciu

8. Dołączanie elementu przed wybranym elementem listy

K01: $p \leftarrow \text{head}$

K02: Jeśli $p \neq e$,

to idź do kroku K05

K03: Wstaw nowy element na początku listy

K04: Zakończ

K05; Dopóki $(p \rightarrow \text{next}) \neq e$,

wykonuj $p \leftarrow (p \rightarrow \text{next})$

K06: Utwórz nowy element

K07: $(p \rightarrow \text{next}) \leftarrow \text{adres nowego elementu}$

K08: $((p \rightarrow \text{next}) \rightarrow \text{next}) \leftarrow e$

K09: $((p \rightarrow \text{next}) \rightarrow \text{data}) \leftarrow v$

K10: Zakończ


```

void slist::insert_before ( slistEl * e, int v )
{
    slistEl * p = head;

    if( p == e ) push_front ( v );
    else
    {
        while( p->next != e ) p = p->next;
        p->next = new slistEl;
        p->next->next = e;
        p->next->data = v;
    }
}

```

Rys. 13 Widok funkcji dołączającej

```

Liczba elementow : 9
Element #1 data = 1
Element #2 data = 2
Element #3 data = 3
Element #4 data = 4
Element #5 data = 4
Element #6 data = 4
Element #7 data = 5
Element #8 data = 6
Element #9 data = 7

```

Rys. 14 wynik programu przed 4 została dodana 4

9. Dołączanie elementu za wybranym elementem listy

- K01: Utwórz nowy element
- K02: $p \leftarrow$ adres nowego elementu
- K03: $(p \rightarrow \text{next}) \leftarrow (e \rightarrow \text{next})$
- K04: $(p \rightarrow \text{data}) \leftarrow v$
- K05: $(e \rightarrow \text{next}) \leftarrow p$
- K06: Zakończ

```

void slist::insert_after ( slistEl * e, int v )
{
    slistEl * p = new slistEl;

    p->next = e->next;
    p->data = v;
    e->next = p;
}

```

Rys. 15 Funkcja do dołączania za wybranym elementem

```
Liczba elementow : 9  
Element #1 data = 1  
Element #2 data = 2  
Element #3 data = 3  
Element #4 data = 4  
Element #5 data = 4  
Element #6 data = 4  
Element #7 data = 5  
Element #8 data = 6  
Element #9 data = 7
```

Rys. 16 Za elementem 4 została dodana 4

Kod źródłowy

```
#include <iostream>

using namespace std;

// Typ elementów listy

struct slistEl
{
    slistEl * next;
    int data;
};

// Definicja typu obiektowego slist

class slist
{
public:
    slistEl * head;

    slist( ); // konstruktor
    ~slist( ); // destruktor
    unsigned size( );
    void printl( );
    void push_front ( int v );
    void push_back ( int v );
    void insert_before ( slistEl * e, int v );
    void insert_after ( slistEl * e, int v );
    void pop_front( );
    void pop_back( );
    void remove ( slistEl * e );
```

```
};
```

```
// Konstruktor listy
```

```
slist::slist()
```

```
{  
    head = NULL;  
}
```

```
// Destruktor listy
```

```
slist::~~slist()
```

```
{  
    while( head ) pop_front();  
}
```

```
// Funkcja oblicza liczbę elementow listy
```

```
unsigned slist::size()
```

```
{  
    unsigned c = 0;  
    slistEl * p = head;  
    while( p )  
    {  
        c++;  
        p = p->next;  
    }  
    return c;  
}
```

```
// Procedura wyświetla zawartosc elementów listy
```

```

void slist::printl( )
{
    unsigned i;
    slistEl * p = head;

    cout << "Liczba elementow : " << size( ) << endl;
    for( i = 1; p; p = p->next ) cout << "Element #" << i++ << " data = " << p->data << endl;
    cout << endl;
}

```

// Dolaczanie na początek listy

```

void slist::push_front ( int v )
{
    slistEl * p;

    p = new slistEl;
    p->next = head;
    p->data = v;
    head = p;
}

```

// Dolaczanie na koniec listy

```

void slist::push_back ( int v )
{
    slistEl * p, * e;

    e = new slistEl; // tworzymy nowy element
    e->next = NULL; // inicjujemy element

```

```

e->data = v;
p = head;
if( p )
{
    while( p->next ) p = p->next;
    p->next = e;
}
else head = e;
}

// Procedura dolaczania przed elementem e

```

```

void slist::insert_before ( slistEl * e, int v )
{
    slistEl * p = head;

    if( p == e ) push_front ( v );
    else
    {
        while( p->next != e ) p = p->next;
        p->next = new slistEl;
        p->next->next = e;
        p->next->data = v;
    }
}

```

```

// Procedura dolaczania za elementem e

void slist::insert_after ( slistEl * e, int v )
{
    slistEl * p = new slistEl;

```

```

p->next = e->next;
p->data = v;
e->next = p;
}

```

// Usuwanie pierwszego elementu

```

void slist::pop_front( )
{
    slistEl * p = head; // zapamiętujemy początek

    if( p )
    {
        head = p->next; // nowy początek
        delete p;      // usuń element z pamięci
    }
}

```

// Usuwanie ostatniego elementu

```

void slist::pop_back( )
{
    slistEl * p = head; // zapamiętujemy początek

    if( p )
    {
        if( p->next )
        {
            while( p->next->next ) p = p->next;
            delete p->next;
        }
    }
}

```

```

    p->next = NULL;
}
else
{
    delete p;
    head = NULL;
}
}
}

```

// Usuwanie wybranego elementu

```

void slist::remove ( slistEl * e )
{
    slistEl * p;

    if( head == e ) pop_front();
    else
    {
        p = head;
        while( p->next != e ) p = p->next;
        p->next = e->next;
        delete e;
    }
}

```

```

int main( )
{
    slist L;    // zawiera adres poczatku listy
    slistEl * e; // do wskazywania elementow listy
    int i;

```



```

for( i = 1; i <= 7; i++ ) L.push_back ( i );
L.printl( );

// Przechodzimy do elementu o wartosci 4 w naszym przypadku czyli e

e = L.head;

for( i = 1; i <= 3; i++ ) e = e->next;

//dodawanie przed
L.insert_before ( e, i );
//dodawanie po
L.insert_after ( e, i );

L.printl( );

// Usuwamy element 4 oraz element pierwszy i ostatni

L.remove ( e );
L.pop_front();
L.pop_back( );
L.printl( );

return 0;
}

```

Link do githuba

<https://github.com/Dzbanowsky/Projekt3>