WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

Przemysław Litwa

Zadanie 4

Rzeszów, 2022

Spis treści

1. Treść zadania: 3
2. Założenia projektu: 3
3. Wykonanie projektu: 3
4. Budowa struktur: 4
4.1. Struktura elementu listy – node: 4
4.2. Klasa obiektu lista: 4
4.2.1 Atrybuty klasy: 4
4.2.2 Metody klasy: 4
5. Schematy blokowe metod: 6
5.1. metoda wypisz() 6
5.2. metoda push_front(int licz) 7
5.3. metoda push_back(int licz) 8
5.4. metoda insert(node *el,int licz) 9
5.5. metoda removal(node *el) 10
5.6. metoda pop_front() 11
5.7. metoda pop_back() 12
5.8. metoda empty() 13
5.9. metoda find(int licz) 14
5.10. metoda find_value(int licz, node *el) 15
5.11. metoda return_size() 15
6. Pseudokody metod: 16
6.1. metoda wypisz() (schemat 1) 16
6.2. metoda push_front(int licz) (schemat 2) 16
6.3. metoda push_back(int licz) (schemat 3) 17
6.4. metoda insert(node *el,int licz) (schemat 4) 17
6.5. metoda removal(node *el) (schemat 5) 18
6.6. metoda pop_front() (schemat 6) 18
6.7. metoda pop_back() (schemat 7) 18
6.8. metoda empty() (schemat 8) 19
6.9. metoda find(int licz) (schemat 9) 19
6.10. metoda find_value(int licz, node *el) (schemat 10) 20
6.11. metoda return_size() (schemat 11) 20
7. Opis kodu: 20
Spisy obrazów: 27

1. Treść zadania:

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

1) przyjąć, że podstawowym typem danych przechowywanym w elemencie struktury będzie struktura z jednym polem typu int.

2) w funkcji main() przedstawić możliwości napisanej przez siebie biblioteki.

3) kod powinien być opatrzony stosownymi komentarzami.

2. Założenia projektu:

Utworzenie programu z implementacją struktury danych typu lista dwukierunkowa z elementem będącym strukturą z pojedynczym polem typu int oraz utworzenie aplikacji konsolowej umożliwiającej utworzenie i obsługę zaimplementowanej listy.

3. Wykonanie projektu:

W ramach projektu zostały utworzone 2 pliki. Plik nagłówkowy lista.h oraz plik źródłowy main.cpp Oba pliki należą do pliku projektowego zad4.cbp. Plik lista.h służy do utworzenia struktury będącej elementem listy oraz

utworzenia klasy obiektu lista. W pliku main.cpp definiuje się funkcje służącej tej klasie za metody oraz główny program do tworzenia listy i umożliwienia użytkownikowi interakcje z nią.

4. Budowa struktur:

4.1. Struktura elementu listy – node:

- liczba – pole typu int zawierające wartość wpisywaną do listy,
- *prew – wskaźnik wskazujący poprzedni element listy,
- *next - wskaźnik wskazujący następny element listy.

4.2. Klasa obiektu lista:

4.2.1 Atrybuty klasy:

- *begin – wskaźnik wskazujący początek listy,
- *end – wskaźnik wskazujący koniec listy,
- count – zmienna zliczająca ilość elementów listy.

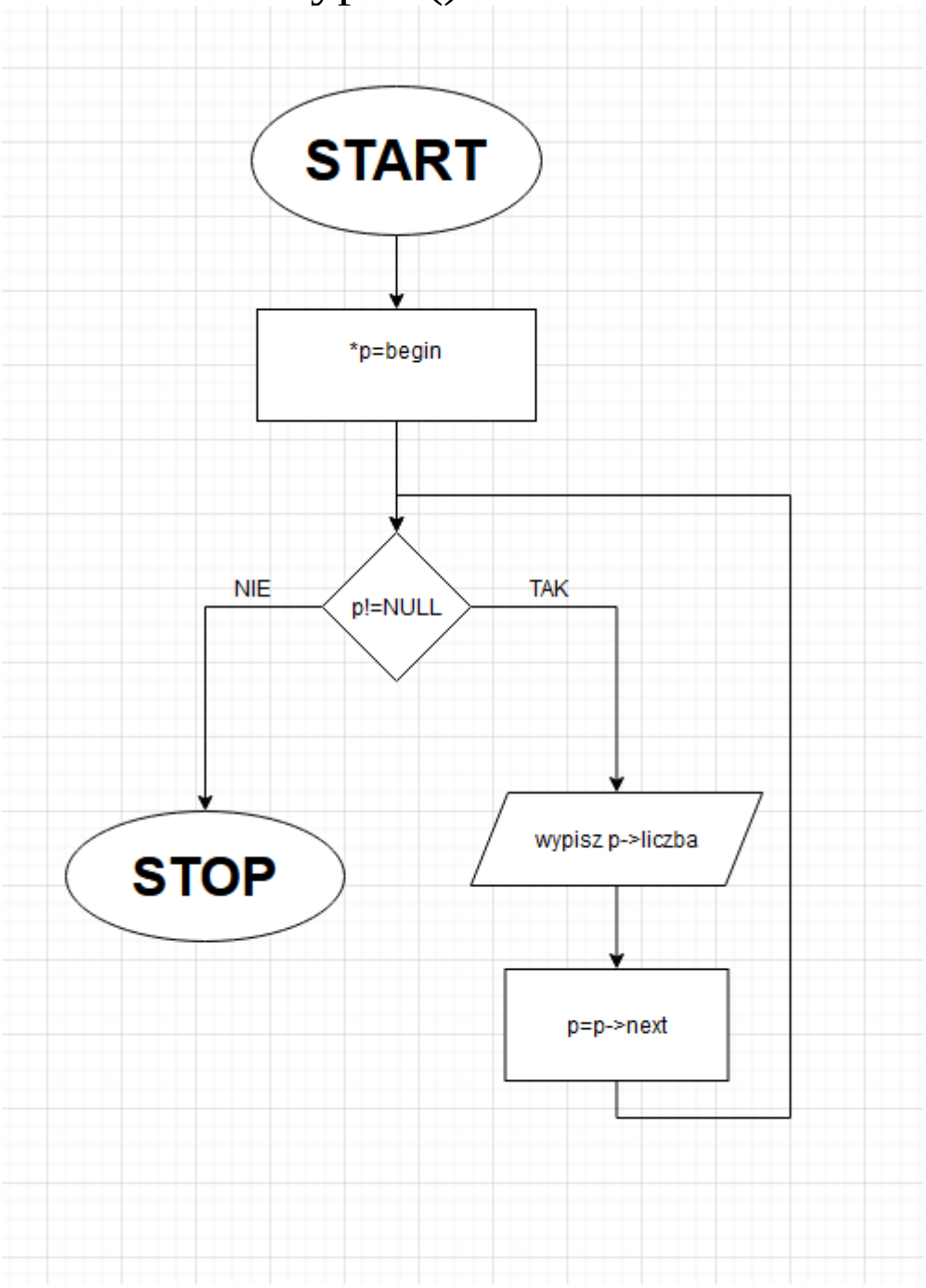
4.2.2 Metody klasy:

- lista() – konstruktor klasy lista,
- ~lista() – destruktor klasy lista,
- push_front() – dodaje element do listy na jej początku,
- push_back() – dodaje element do listy na jej końcu,
- insert() – dodaje element do listy w miejscu wskazanym przez użytkownika,
- pop_front() – usuwa pierwszy element listy,
- pop_back() – usuwa ostatni element listy,

- removal() – usuwa wskazany przez użytkownika element listy,
- return_size() – zwraca ilość elementów listy,
- empty() – sprawdza czy lista jest pusta,
- find() – znajduje element listy na miejscu podanym przez użytkownika i go zwraca,
- find_value() – wypisuje wartość elementu podanego przez użytkownika,
- wypisz() – wypisuje listę,
- lista_plik() – zapisuje listę w pliku.

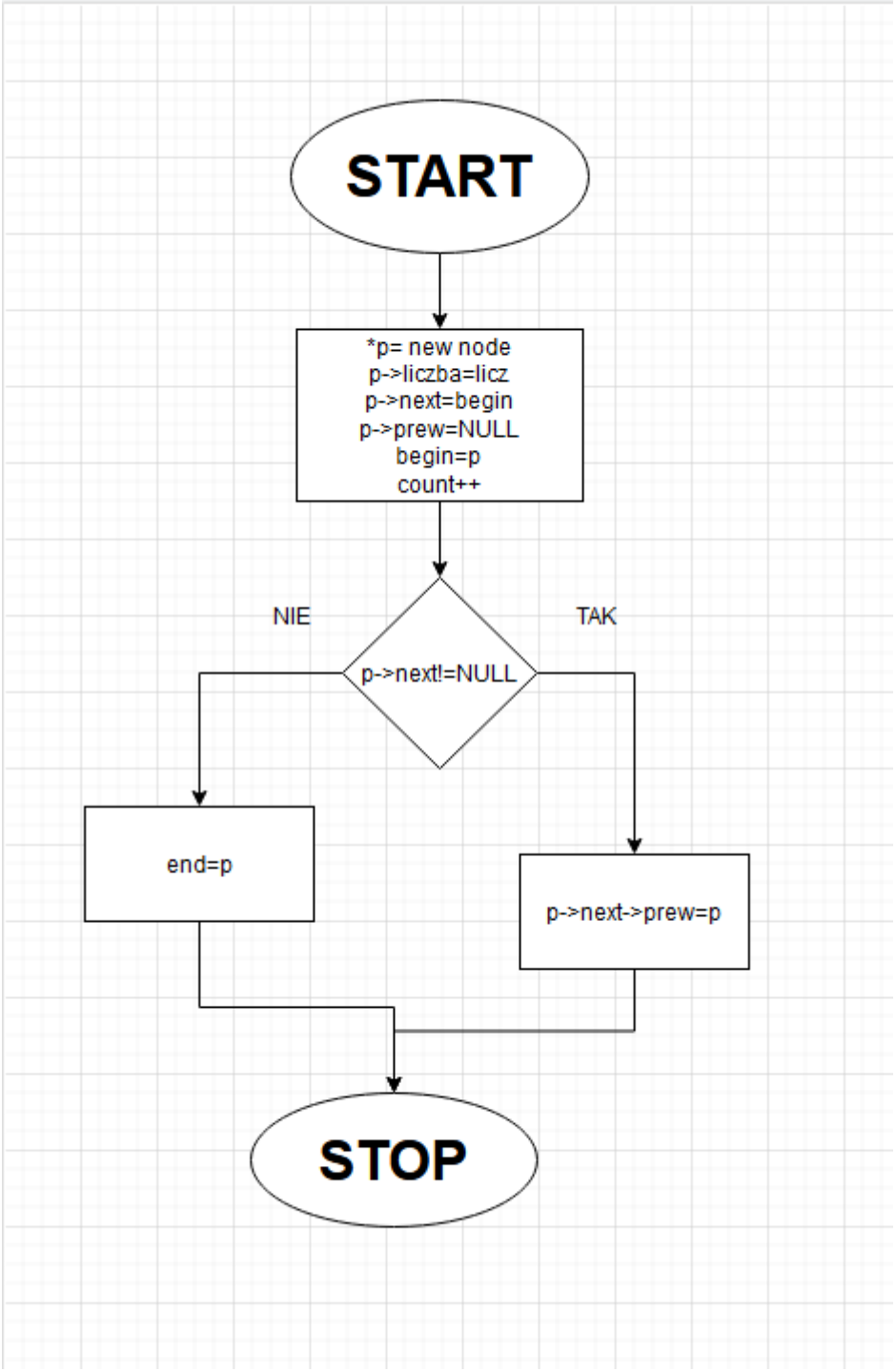
5. Schematy blokowe metod:

5.1. metoda wypisz()



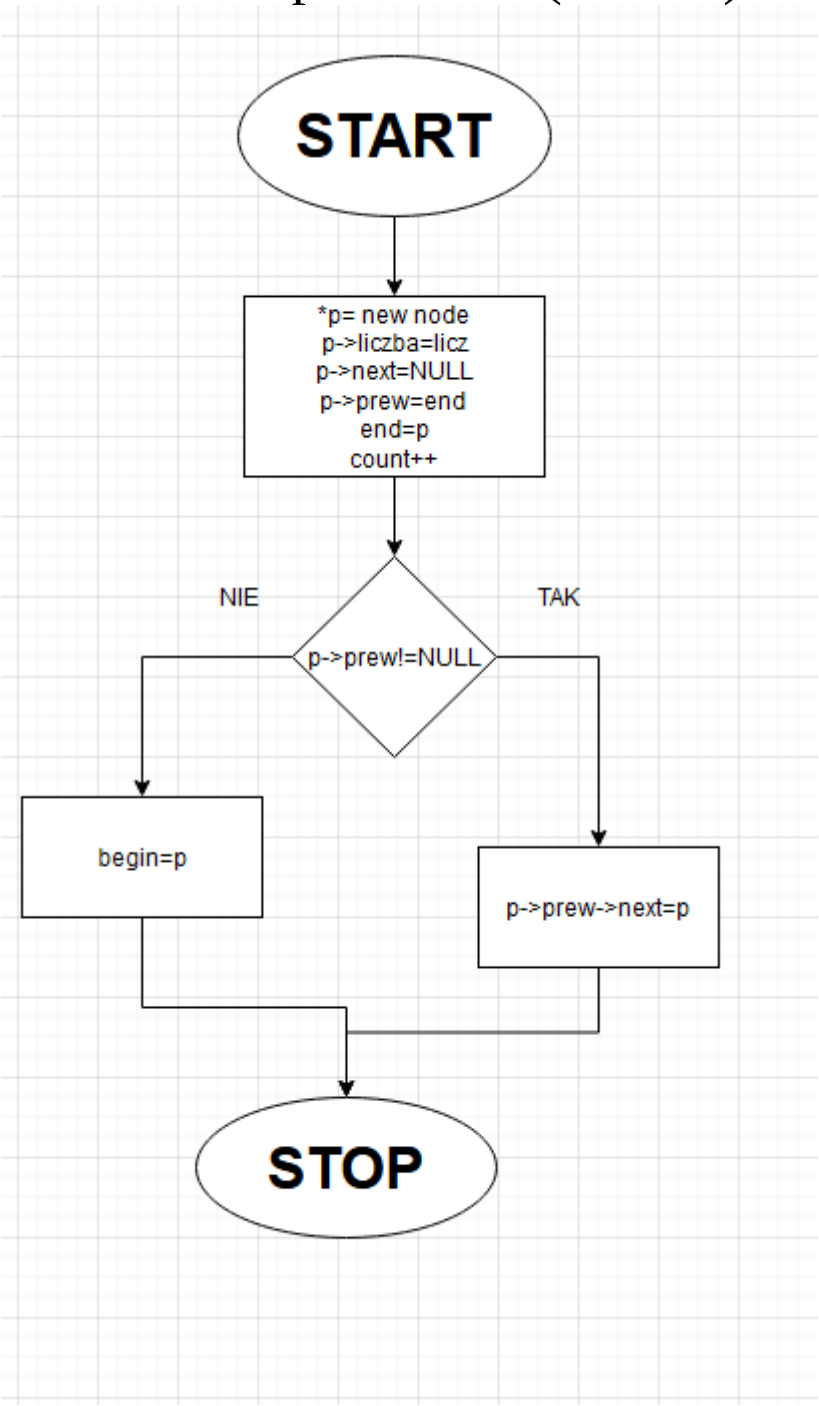
schemat 1

5.2. metoda push_front(int licz)



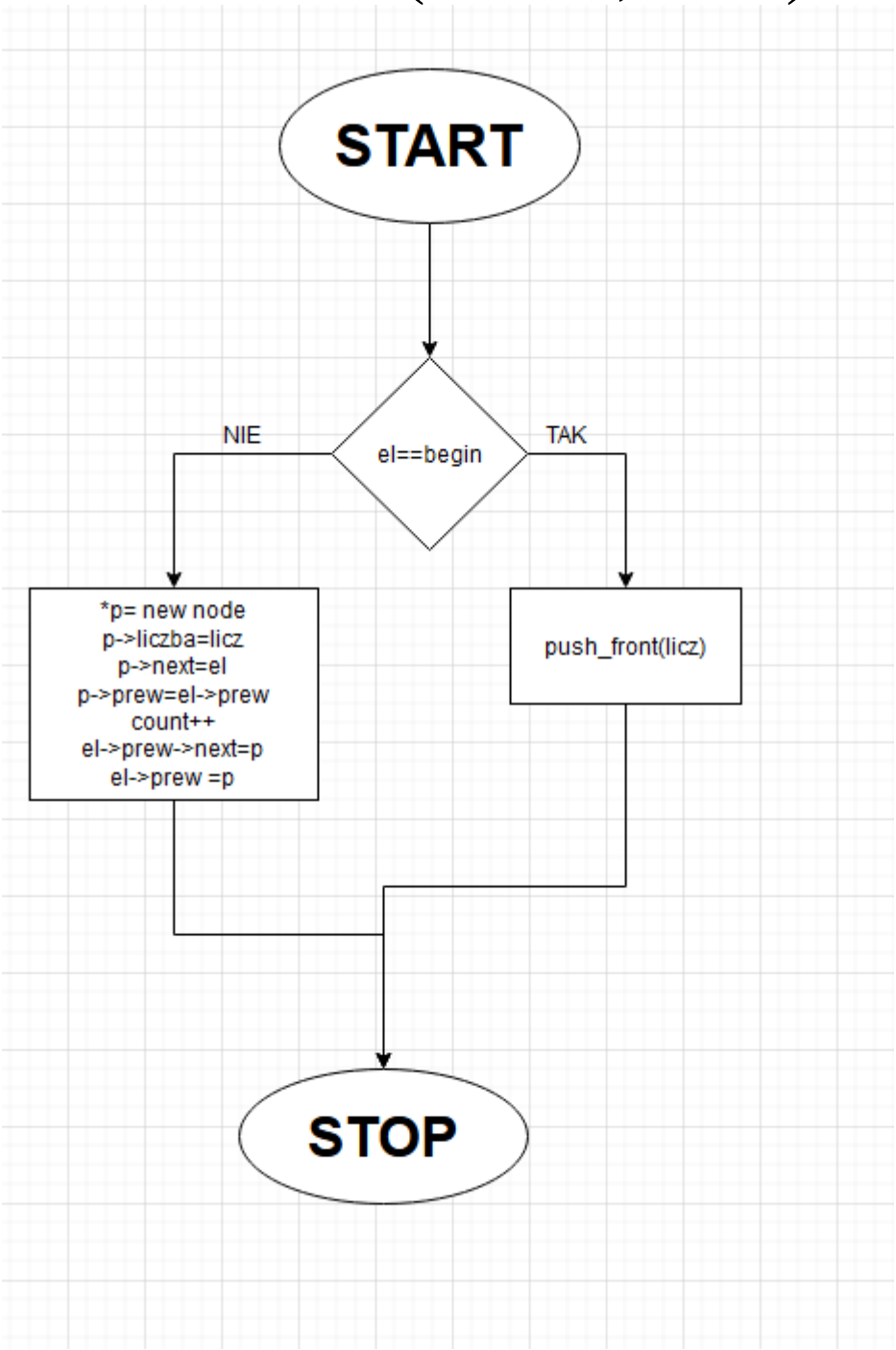
schemat 2

5.3. metoda push_back(int licz)



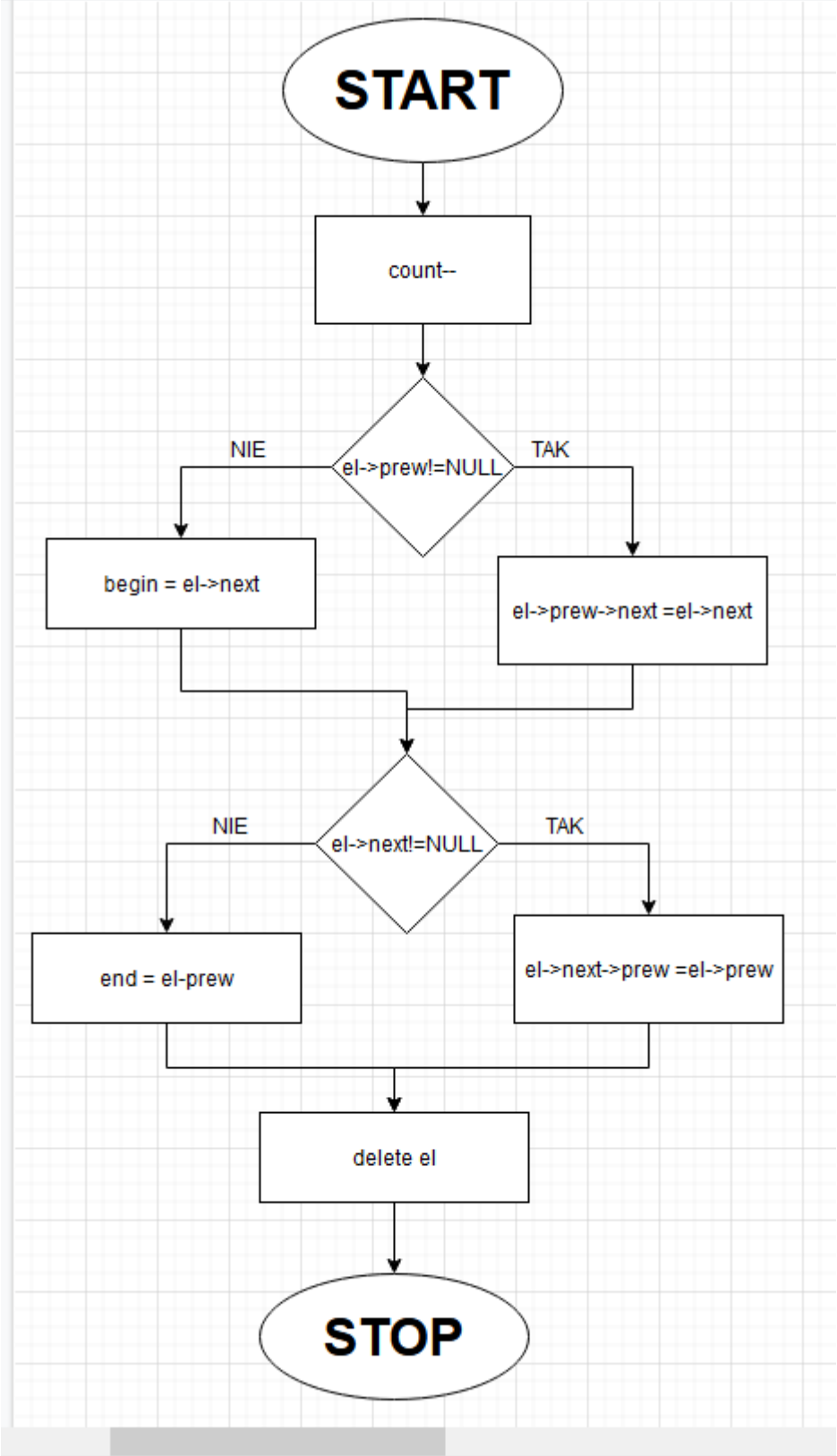
schemat 3

5.4. metoda insert(node *el,int licz)



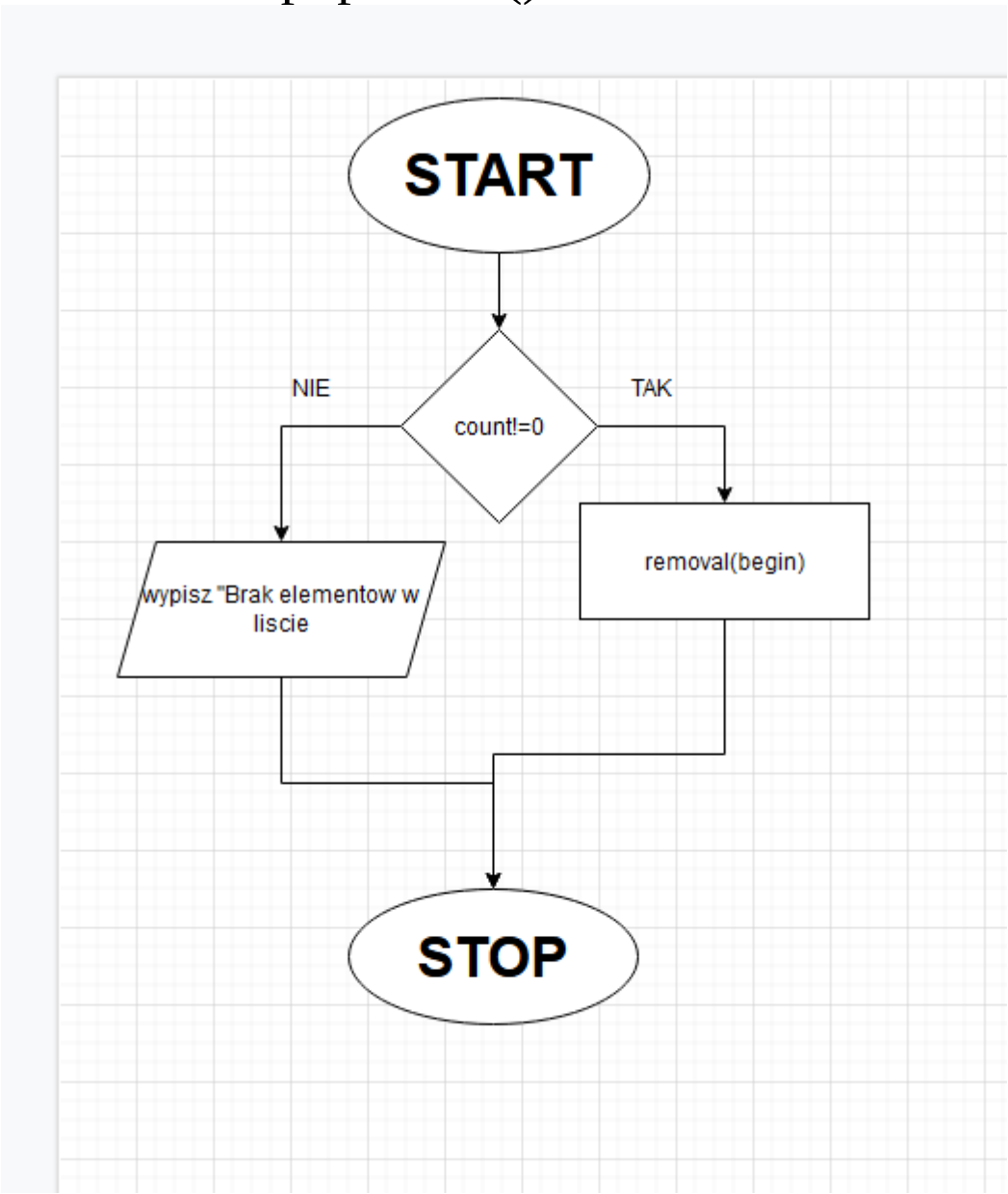
schemat 4

5.5. metoda removal(node *el)



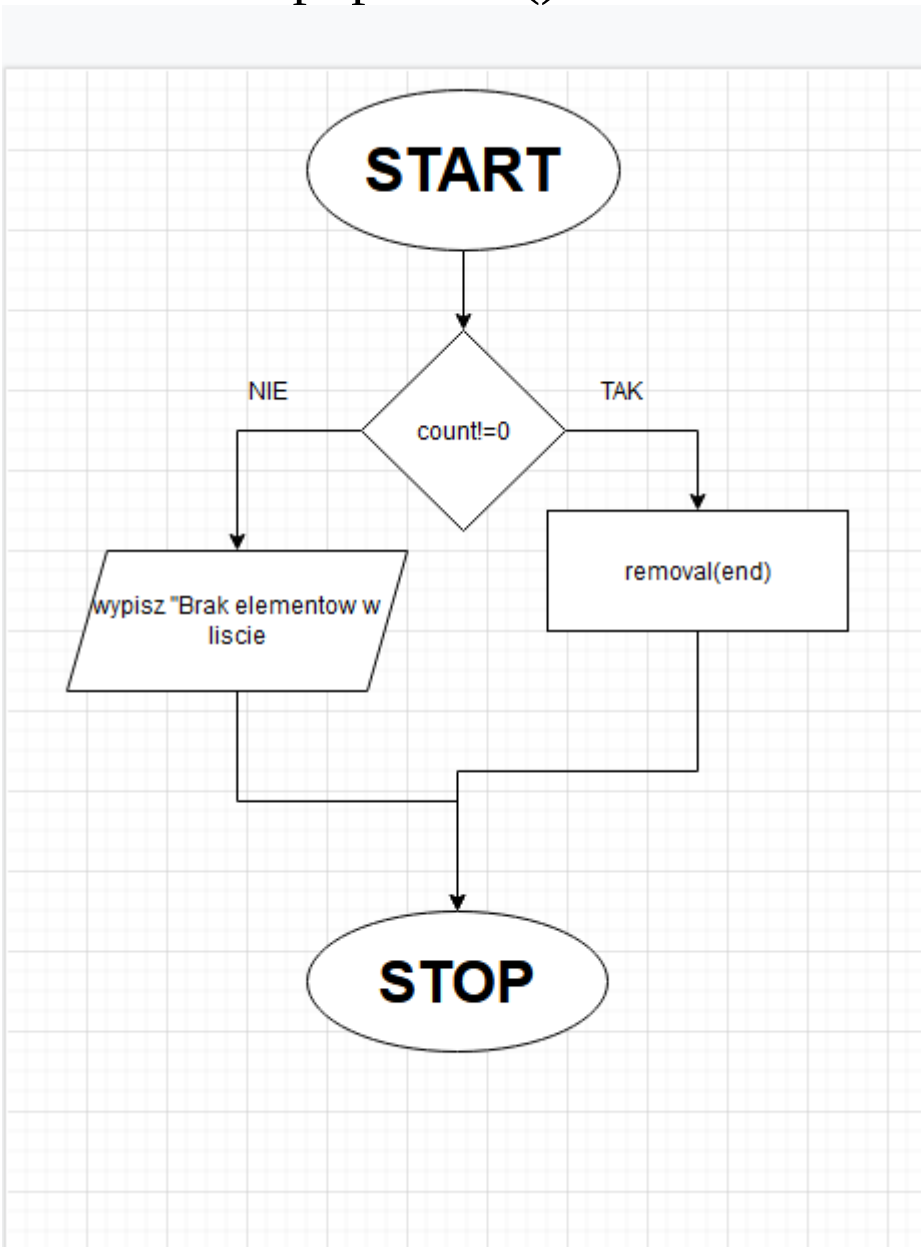
schemat 5

5.6. metoda pop_front()



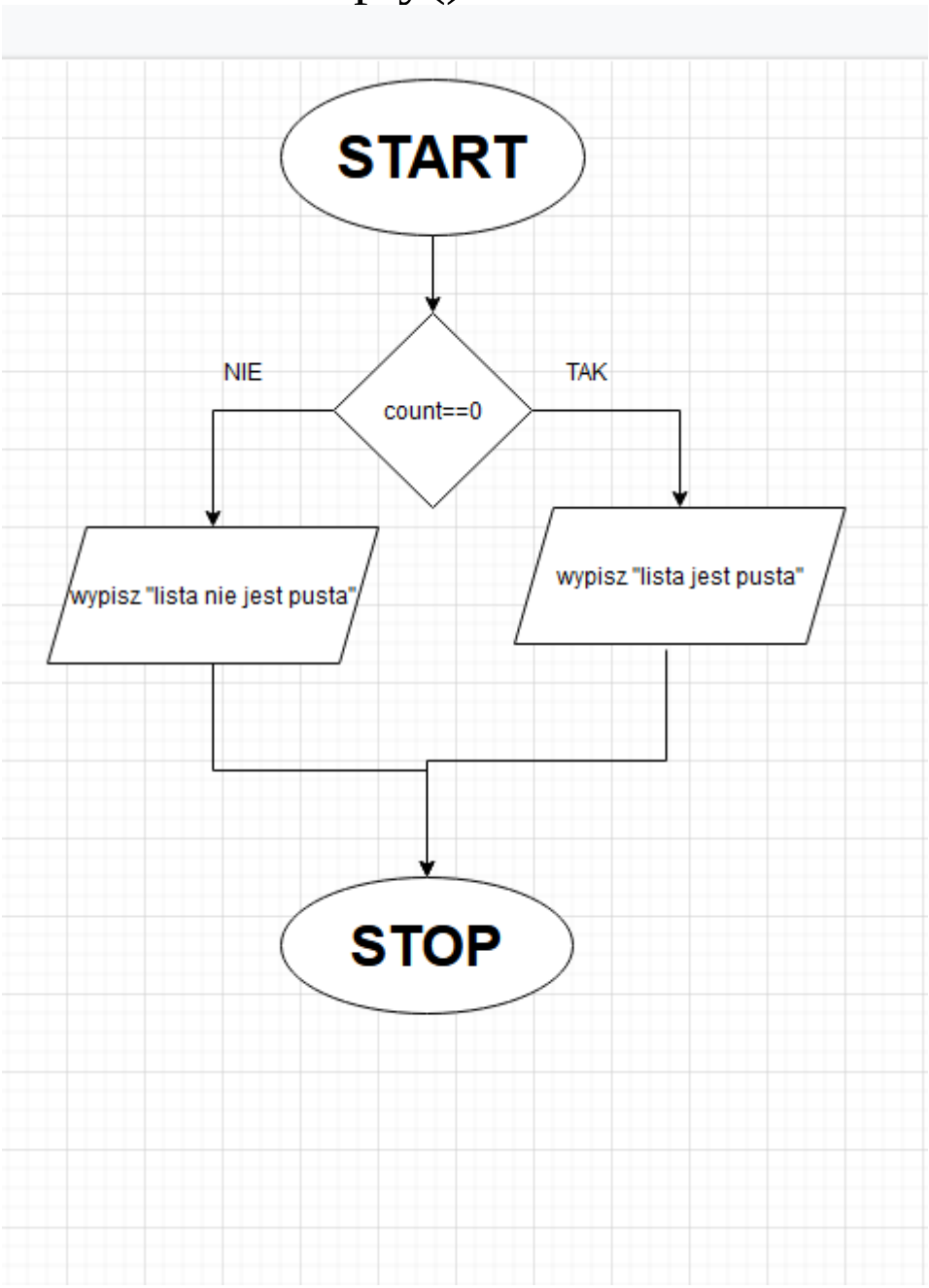
schemat 6

5.7. metoda pop_back()



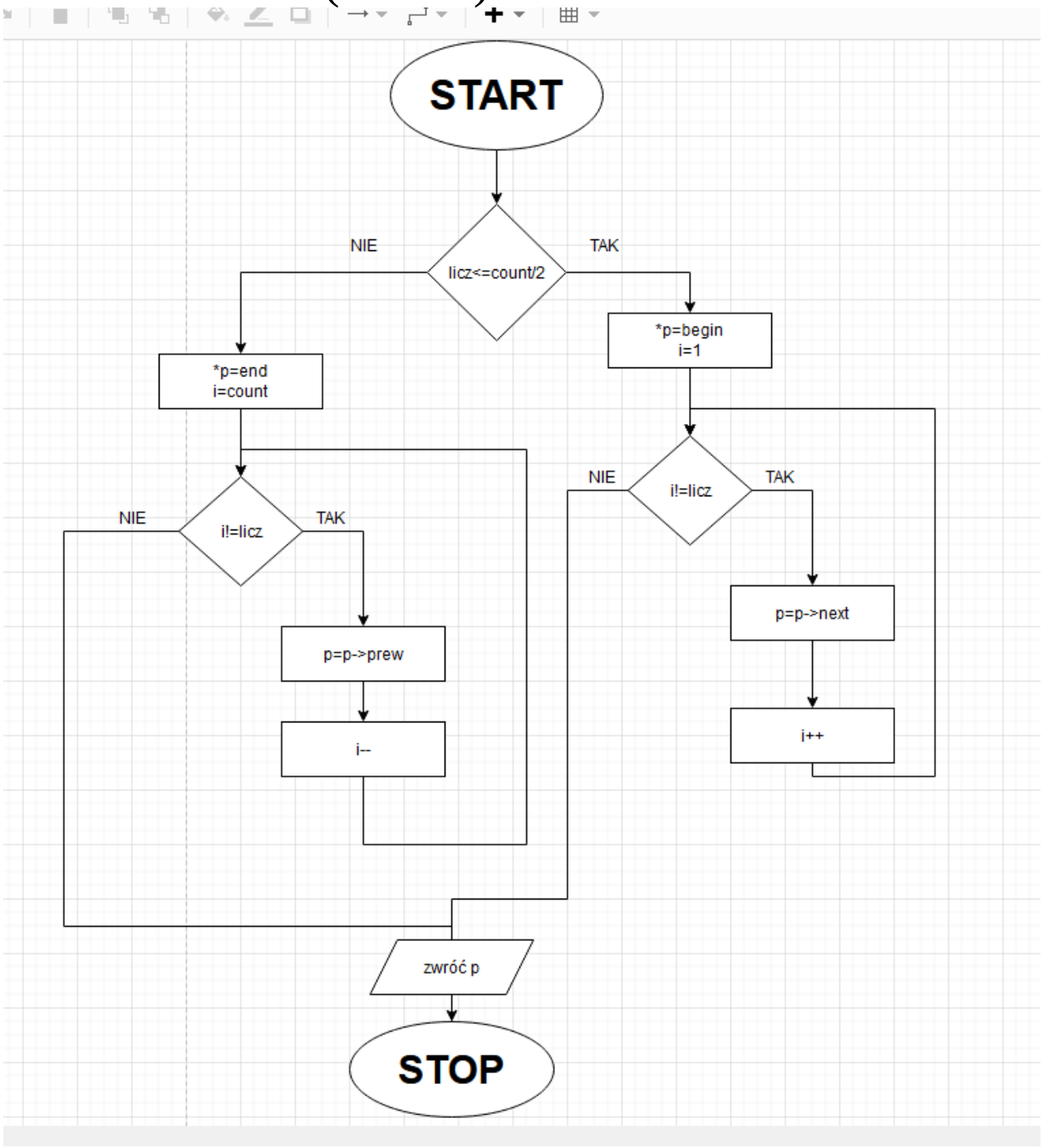
schemat 7

5.8. metoda empty()



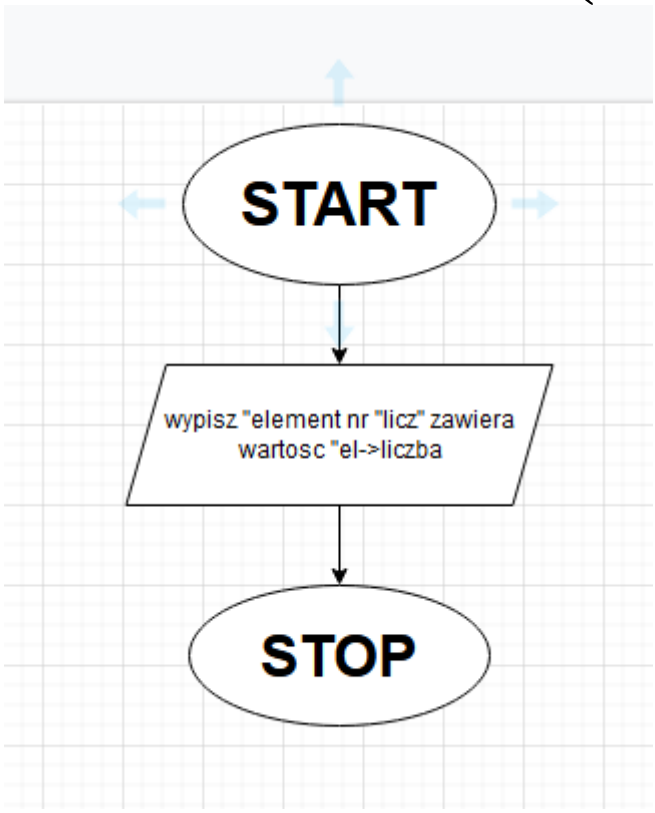
schemat 8

5.9. metoda find(int licz)



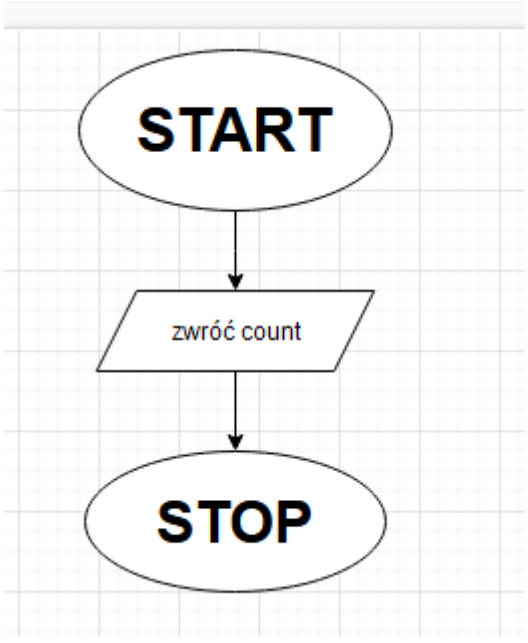
schemat 9

5.10. metoda find_value(int licz, node *el)



schemat 10

5.11. metoda return_size()



schemat 11

6. Pseudokody metod:

6.1. metoda wypisz() (schemat 1)

Wypisz "elementy w liscie to: "

p = begin

dla (p != NULL) wykonuj{

    wypisz p->liczba " "

    p=p->next

}

6.2. metoda push_front(int licz) (schemat 2)

p = nowy node

p->liczba=licz

p->prew=NULL

p->next=begin

begin=p

count++

jeżeli (p->next!=NULL)

    p->next->prew = p

w przeciwnym wypadku

    end=p

6.3. metoda push_back(int licz) (schemat 3)

p = nowy node

p->liczba=licz

p->next=NULL

p->prew=end

end=p

count++

jeżeli (p->prew!=NULL)

    p->prew->next = p

w przeciwnym wypadku

    begin=p

6.4. metoda insert(node *el,int licz) (schemat 4)

jeżeli (el==begin)

    push_front(licz)

w przeciwnym wypadku

    p = nowy node

    p->liczba=licz

    p->next=el

    p->prew=el->prew

    count++

    el->prew->next = p

el->prew = p

6.5. metoda removal(node *el) (schemat 5)

count--

jeżeli (el->prew!=NULL)

el->prew->next = el->next

w przeciwnym wypadku

begin = el->next

jeżeli (el->next!=NULL)

el->next->prew = el->prew

w przeciwnym wypadku

end = el->prew

usuń el

6.6. metoda pop_front() (schemat 6)

jeżeli (count!=0)

removal(begin)

w przeciwnym wypadku

wypisz "Brak elementow w liscie"

6.7. metoda pop_back() (schemat 7)

jeżeli (count!=0)

removal(end)

w przeciwnym wypadku

wypisz "Brak elementow w liscie"

6.8. metoda empty() (schemat 8)

```
jeżeli (count==0)
    wypisz "lista jest pusta"
w przeciwnym wypadku
    wypisz "lista nie jest pusta"
```

6.9. metoda find(int licz) (schemat 9)

```
jeżeli (licz<=count/2)
{
    p=begin
    i=1
    dla (i!=licz) wykonuj
    {
        p=p->next
        i++
    }
}
w przeciwnym wypadku
{
    p=end
    i=count
    dla (i!=licz) wykonuj
```

{

p=p->prew

i--

}

}

zwróć p

6.10. metoda find_value(int licz, node *el) (schemat 10)

wypisz "element nr " licz " zawiera wartosc " el->liczba

6.11. metoda return_size() (schemat 11)

zwróć count

7. Opis kodu:

Program składa się z 2 plików: Pliku nagłówkowego lista.h oraz pliku źródłowego main.cpp. Plik lista.h zawiera budowę struktury node oraz klasę listę. Budowa tych struktur jest przedstawione w rozdziale - 4. Budowa struktur:.

```
struct node // struktura elementu listy
{
    int liczba;
    node *prew;
    node *next;
};

class lista // klasa lista
{
private:
    // atrybuty klasy lista
    node *begin;
    node *end;
    unsigned int count;
public:


    lista(); //konstruktor klasy lista
    ~lista(); //destruktor klasy lista

    // metody klasy lista
    int return_size();
    void push_front( int licz);
    void push_back( int licz);
    void pop_front();
    void pop_back();
    void insert(node *el,int licz);
    void removal (node *el);
    node *find(int licz);
    void find_value(int licz,node *el);
    void wypisz();
    void lista_plik();
    void empty();

};
```

Obraz 1 lista.h

Drugim plikiem jest plik main.cpp który zawiera ciała metod klasy lista funkcję main zawierającą interfejs użytkownika oraz funkcję podaj liczbę.

```
int podaj_liczbe() // fu
{
    int liczba;
    while(true){
       cin>>liczba;
        if(!cin)
    {
       cin.clear();
       cin.sync();
    }
    else break;
    }
    return liczba;
}
```

Obraz 2 podaj_liczbe

Funkcja ta pozwala na wpisanie liczby z klawiatury. Sprawdza też czy to co wpisał użytkownik faktycznie jest liczbą. Jeżeli zostało wpisane coś innego niż liczba program nie przejdzie dalej dopóki nie zostanie wpisana liczba.

```
void lista::wypisz() //funkcja wypisujaca liste
{
    node *p;

    cout<<"elementy w liscie to: ";
    p= begin;
    while(p)
    {
        cout<<p->liczba<<" ";
        p=p->next;
    }
}
void lista::lista_plik() //funkcja do zapisu listy w pliku tekstowym
{
    node *p;
    ofstream plik("lista.txt");
    int i=1;

    p= begin;
    while(p)
    {
        plik<<"el["<<i<<"]="<<p->liczba<<"  ";
        p=p->next;
        i++;
    }
    plik.close();
}
```

Obraz 3 wypisz i lista plik()

Funkcje wypisz i lista_plik służą do wypisania listy odpowiednio na ekran i do pliku lista.txt. Obie funkcje działają praktycznie tak samo tworzony jest wskaźnik p który następnie przyjmuje wartość wskaźnika listy begin i przechodzi po kolejnych elementach tak długo aż przejdzie po całej liście wypisując każdy element.

```
void lista::push_front(int licz) //funkja dodajaca element do listy na po
{
    node *p;

    p = new node;
    p->liczba=licz;
    p->prew=NULL;
    p->next=begin;
    begin=p;
    count++;
    if (p->next) {p->next->prew = p;}
    else end=p;
}
void lista::push_back(int licz) //funkja dodajaca element do listy na poc
{
    node *p;

    p = new node;
    p->liczba=licz;
    p->next=NULL;
    p->prew=end;
    end=p;
    count++;
    if (p->prew) p->prew->next =p;
    else begin=p;
}
```

Obraz 4 push_front i push_back

```
void lista::insert(node *el,int licz) //funkja
{
    node *p;

    if( el==begin) push_front(licz);
    else
    {
        p = new node;
        p->liczba =licz;
        p->next=el;
        p->prew = el->prew;
        count++;
        el->prew->next = p;
        el->prew =p;
    }
}
```

Obraz 5 insert

Funkcje push_front, push_back oraz insert służą do wstawienia do listy nowego elementu na odpowiednie miejsce. Na początek oraz koniec oraz miejsce wybrane przez użytkownika odpowiednio. Działanie funkcji polega na utworzeniu nowego elementu oraz ustawieniu wskaźników na elementy z którymi będzie sąsiadował oraz wskaźników elementów sąsiednich tak

aby wskazywały na nowy element. Zwiększają one też wartość argumentu count o 1 aby oddawał ilość elementów w liście.

```
void  lista::removal (node *el) //funkcja usuwa podany elem
{
    count--;
    if (el->prew) el->prew->next = el->next;
    else begin = el->next;
    if (el->next) el->next->prew = el->prew;
    else end= el->prew;
    delete el;
}
```

Obraz 6 removal

Metoda removal usuwa podany element z listy i ustawia wskaźniki elementów sąsiednich tak aby wskazywały na siebie a nie na usuwany element. Zmniejsza też wartość argumentu count o 1.

```
void lista::pop_front() //funkcja usuwa pierwszy element listy
{
    if(count!=0) removal(begin);
    else{ cout<<"Brak elementow w liscie"<<endl;
    getch();}
}
void lista::pop_back() //funkcja usuwa ostatni element listy
{
    if(count!=0) removal(end);
    else{ cout<<"Brak elementow w liscie"<<endl;
    getch();}
}
```

Obraz 7 pop_front i pop_back

Metody pop_front i pop_back usuwają odpowiednio pierwszy i ostatni element listy korzystając z metody removal. W przypadku braku elementów wyświetlany jest komunikat „Brak elementów w liście”.

```
node *lista::find(int licz) //funkcja znajduje podany element w liście
{
    int i;
    node *p;
    if(licz<=(count/2))
    {
        p=begin;
        for(i=1;i!=licz;i++)
         p=p->next;
    }
    else
    {
        p=end;
        for(i=count;i!=licz;i--)
         p=p->prew;
    }
    return p;
}

void lista::find_value(int licz, node *el) //funkcja wyświetla wartośc
{
     cout<<"element nr "<<licz<<" zawiera wartosc "<<el->liczba<<endl;
}
```

Obraz 8 find i find_value

Metoda find znajduje element o pozycji podanej przez użytkownika w liście. Funkcja przeszukuje listę w poszukiwaniu elementu o wybranej pozycji. W przypadku elementów o pozycji do połowy wszystkich elementów listy funkcja find przeszukuje od pierwszego elementu. W pozostałych przypadkach przeszukiwanie odbywa się od końca listy. Element jest następnie zwracany.

Metoda ta jest używana w tandemie z innymi metodami tej klasy (insert, removal, find_value) w celu podania tym funcją konkretnego elementy listy.

Metoda find_value wypisuje wartość konkretnego elementu.

```
void lista::empty() //funkcja sprawdzajaca czy lista jest pusta
{
     if(count==0) cout<<"lista jest pusta"<<endl;
     else cout<<"lista nie jest pusta"<<endl;
}
```

Obraz 9 empty

```
int lista::return_size()
{
     return count; // zwraca wartość atrybutu count
}
```

Obraz 10 return_size

Metoda empty wyświetla informację czy tablica jest pusta czy nie. Funkcja return_size zwraca wartość argumentu count.

```
int main()
{
    int opcja=0;
    char plik;
    lista lit;
    do{
            system("CLS");
            lit.wypisz(); // wyświetlenie listy
    cout<<endl<<endl;
    cout<<"Obsluga listy"<<endl; // wyswietlenie menu obsługi listy
    cout<<"1 - push_front"<<endl;
    cout<<"2 - push_back"<<endl;
    cout<<"3 - insert_on_position"<<endl;
    cout<<"4 - pop_front"<<endl;
    cout<<"5 - pop_back"<<endl;
    cout<<"6 - remove_from_position"<<endl;
    cout<<"7 - is_empty"<<endl;
    cout<<"8 - size"<<endl;
    cout<<"9 - find_position"<<endl;
    cout<<"10 - exit_program"<<endl;
    cout<<endl;
    cout<<"wybierz opcje: ";
    cin>>opcja; // podanie opcji

    switch(opcja) // obsługa listy po wyborze opcji
    {
        case 1:{
```

Obraz 11 main

```
elementy w liscie to: 55 30 40

Obsluga listy
1 - push_front
2 - push_back
3 - insert_on_position
4 - pop_front
5 - pop_back
6 - remove_from_position
7 - is_empty
8 - size
9 - find_position
10 - exit_program

wybierz opcje:
```

Obraz 12 program

Funkcja main inicjalizuje listę, wypisuje ją, wyświetla menu oraz umożliwia wykonywanie operacji na liście. Dostępne opcje są wyświetlane i użytkownik wpisując daną liczbę wybiera jaką

operację program będzie wykonywał po wyborze opcji. W przypadku gdy dana opcja wymaga wpisania jakiś danych z klawiatury pojawia się do tego odpowiednie okno. Po wykonaniu operacji ekran się wyczyści i wszystko zostanie ponownie wyświetlone z zaktualizowaną zawartością listy.

Spisy obrazów:

Spis ilustracji:

Obraz 1 lista.h 21
Obraz 2 podaj_liczbe 21
Obraz 3 wypisz i lista plik() 22
Obraz 4 push_front i push_back 23
Obraz 5 insert 23
Obraz 6 removal 24
Obraz 7 pop_front i pop_back 24
Obraz 8 find i find_value 25
Obraz 9 empty 25
Obraz 10 return_size 25
Obraz 11 main 26
Obraz 12 program 26

Spis schematów:

schemat 1 6
schemat 2 7
schemat 3 8
schemat 4 9
schemat 5 10
schemat 6 11
schemat 7 12
schemat 8 13
schemat 9 14
schemat 10 15
schemat 11 15