

# Algorytmy i struktury danych

**Aldona Świrad**

Projekt

Struktury danych: lista jednokierunkowa

Rzeszów, 2022



## Spis treści

<b>1.Wstęp</b>	<b>5</b>
<b>2.Opis problemu</b>	<b>6</b>
2.1.Opis podstaw teoretycznych zagadnienia	6
2.2.Opis szczegółów implementacji problemu	6
<b>3.Badanie algorytmu</b>	<b>7</b>
3.1. Dodawanie elementu na początku listy	7
3.2. Usuwanie elementu z początku listy	7
3.3. Dodawanie elementu na koniec listy	8
3.4. Usuwanie ostatniego elementu listy	8
3.5. Dodawanie nowego elementu przed wybranym elementem listy	9
3.6. Dodawanie nowego elementu za elementem wybranym	9
3.7. Usuwanie wybranego elementu listy	10
<b>4.Pseudokod</b>	<b>11</b>
<b>4.Wnioski</b>	<b>18</b>
<b>Bibliografia</b>	<b>19</b>



## **1. Wstęp**

Projekt napisany w języku programowania C++, w środowisku Code::Blocks IDE, realizowany w ramach przedmiotu „Algorytmy i struktury danych” na kierunku Inżynieria i analiza danych, semestr I, grupa 8.

## **2. Opis problemu**

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.

### **2.1. Opis podstaw teoretycznych zagadnienia**

Do rozwiązania problemu listy jednokierunkowej wykorzystano tablice statyczne i dynamiczne, pętle „for” oraz instrukcje warunkowe „if”, klasy, obiekty, struktury. W celu sprawnego testowania algorytmu, za pomocą instrukcji warunkowej „switch” stworzony został program z przejrzystym menu, gdzie użytkownik sam może zdecydować o źródle pochodzenia danych wejściowych.

### **2.2. Opis szczegółów implementacji problemu**


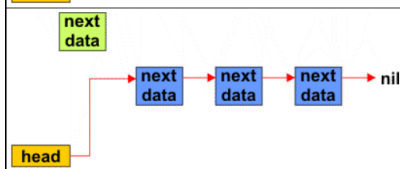
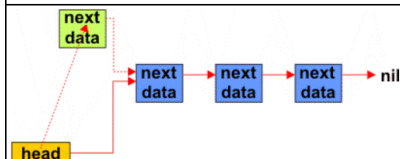

Możliwości listy jednokierunkowej, jakie zostały zaimplementowane:

1. Zliczanie liczby elementów listy
2. Dodawanie elementu na początku listy
3. Usuwanie elementu z początku listy
4. Dodawanie elementu na koniec listy
5. Usuwanie ostatniego elementu listy
6. Dodawanie nowego elementu przed wybranym elementem listy
7. Dodawanie nowego elementu za elementem wybranym
8. Usuwanie wybranego elementu listy

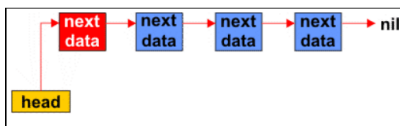


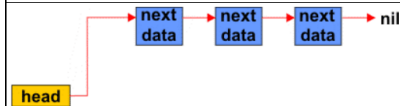
W dalszych rozważaniach będziemy używali list jako obiektów ze względu na wygodę programowania. Obiekt jest strukturą danych, która posiada pola danych oraz funkcje i procedury obsługujące te pola. Do funkcji i procedur obiektu odwołujemy się identycznie jak do pól struktury.

### 3. Badanie algorytmu

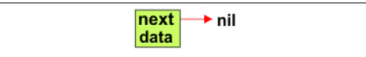
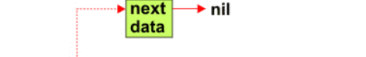
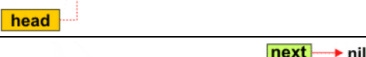


#### 3.1. Dodawanie elementu na początku listy

	<p>Założmy, iż mamy daną listę trójelementową. Naszym zadaniem jest dodanie nowego elementu na początku tej listy. Postępujemy wg następującego schematu:</p>
	<p>Tworzymy dynamicznie w pamięci nowy element listy (na rysunku w kolorze zielonym). Wprowadzamy do niego dane dla pola <i>data</i>.</p>
	<p>W polu <i>next</i> nowego elementu umieszczamy adres przechowywany przez zmienną <i>head</i>. W ten sposób następnikiem nowego elementu stanie się obecny pierwszy element listy.</p>
	<p>W zmiennej <i>head</i> umieszczamy adres nowego elementu. Po tej operacji początkiem listy staje się nowo utworzony element.</p>





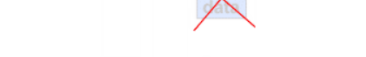

#### 3.2. Usuwanie elementu z początku listy

	<p>Jeśli lista jest pusta, to nic nie robimy.</p> <p>Założmy, że mamy daną listę, z której początku należy usunąć element zaznaczony na rysunku kolorem czerwonym.</p>
	<p>W zmiennej <i>head</i> umieszczamy zawartość pola <i>next</i> usuwanego elementu. W ten sposób początek listy rozpocznie się w następniku, a usuwany element zostanie wyłączony z listy.</p>
	<p>Odlączony element usuwamy z pamięci.</p>
	<p>Otrzymujemy listę bez pierwszego elementu.</p>

### 3.3. Dodawanie elementu na koniec listy

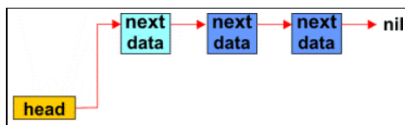


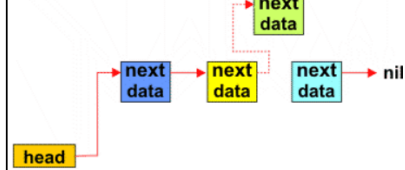
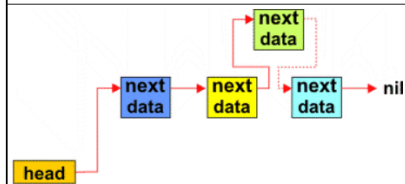
	Tworzymy nowy element. W polu <i>next</i> umieszczamy adres zero – ostatni element listy nie posiada następnika.
	Jeśli lista jest pusta, to <i>head</i> zawiera adres zero. W takim przypadku adres nowego elementu umieszczamy w zmiennej <i>head</i> . Nowy element staje się jednocześnie pierwszym i ostatnim elementem listy.
	Jeśli lista zawiera elementy, to przechodzimy do ostatniego z nich (za pomocą podanego wcześniej algorytmu przejścia).
	W polu <i>next</i> ostatniego elementu umieszczamy adres nowego elementu. W ten sposób nowy element staje się częścią listy i jest umieszczony na jej końcu.
	Lista zostaje wydłużona o nowy element.

### 3.4. Usuwanie ostatniego elementu listy

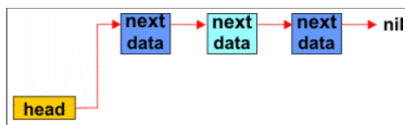
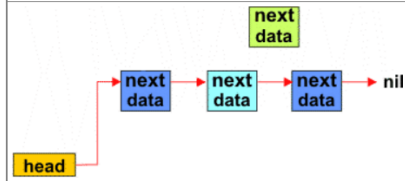
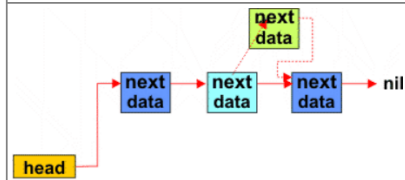
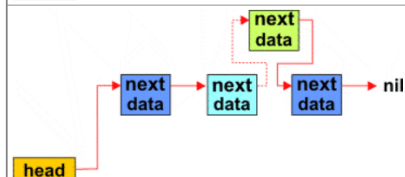
	Jeśli lista jest pusta, nic nie robimy. W przeciwnym razie sprawdzamy, czy lista zawiera tylko jeden element. Rozpoznamy to po tym, iż pole <i>next</i> pierwszego elementu zawiera adres zero.
	Jeśli tak, to w zmiennej <i>head</i> umieszczamy adres zero. Spowoduje to odłączenie elementu od listy (oczywiście adres tego elementu musimy wcześniej zapamiętać, gdyż inaczej stracimy do niego dostęp! )..
	Odłączony element usuwamy z listy. Lista pozostaje pusta. Kończymy.
	Jeśli lista nie jest pusta, to naszym zadaniem staje się usunięcie ostatniego jej elementu, który zaznaczyliśmy na rysunku kolorem czerwonym. Przechodzimy na liście do przedostatniego elementu – przedostatni element rozpoznamy po tym, iż pole <i>next</i> jego następnika zawiera adres zero.
	Usuwanie z pamięci element wskazywany przez pole <i>next</i> przedostatniego elementu listy.
	W polu <i>next</i> elementu przedostatniego (który teraz staje się elementem ostatnim listy) umieszczamy adres zero. W ten sposób lista staje się krótsza o jeden element.




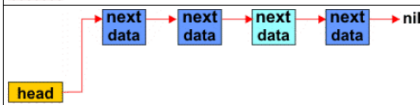
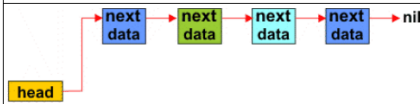
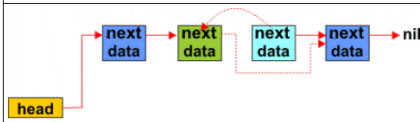
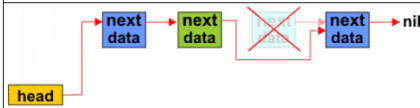
### 3.5. Dodawanie nowego elementu przed wybranym elementem listy

	<p>Jeśli wybranym elementem jest pierwszy element listy, to operacja sprowadza się do opisanej wcześniej operacji dodawania elementu na początku listy.</p>
	<p>Założmy, że wybranym elementem nie jest pierwszy element na liście, lecz dowolny z następnych. Na rysunku zaznaczono go kolorem jasnoniebieskim.</p>
	<p>Znajdujemy poprzednik elementu wybranego – listę przechodzimy dotąd, aż pole <i>next</i> przetwarzanego elementu będzie zawierało adres elementu wybranego. Poprzednik elementu wybranego zaznaczyliśmy kolorem żółtym.</p>
	<p>Tworzymy dynamicznie nowy element (na rysunku zaznaczony kolorem zielonym) i jego adres umieszczamy w polu <i>next</i> poprzednika elementu wybranego. W ten sposób nowy element stanie się elementem następnym poprzednika. Lista chwilowo zostanie rozdzielona.</p>
	<p>W polu <i>next</i> nowego elementu umieszczamy adres elementu wybranego. W ten sposób nowy element zostanie umieszczony na liście przed elementem wybranym.</p>

### 3.6. Dodawanie nowego elementu za elementem wybranym

	<p>Zadanie mamy znacznie uproszczone w porównaniu z poprzednim. Założmy, że wybrany element jest dowolnym elementem listy, nieważne którym.</p>
	<p>Tworzymy dynamicznie w pamięci nowy element.</p>
	<p>W polu <i>next</i> nowego elementu umieszczamy zawartość pola <i>next</i> elementu wybranego. W ten sposób następnikiem nowego elementu będzie następnik elementu wybranego.</p>
	<p>W polu <i>next</i> elementu wybranego umieszczamy adres nowego elementu. Nowy element zostaje umieszczony na liście za elementem wybranym.</p>

### 3.6. Usuwanie wybranego elementu listy

	<p>Jeśli wybranym do usunięcia elementem jest pierwszy na liście, to wykorzystujemy podany wcześniej algorytm usuwania pierwszego elementu listy.</p>
	<p>Założmy, że usuwany element nie jest pierwszym elementem listy.</p>
	<p>Skoro tak, to posiada on poprzedniki. Znajdujemy poprzednik wybranego elementu – na rysunku zaznaczyliśmy go na zielono.</p>
	<p>W polu <i>next</i> poprzednika umieszczamy zawartość pola <i>next</i> usuwanego elementu. W ten sposób poprzednik będzie wskazywał na następnik wybranego elementu, a sam element wybrany znajdzie się poza listą.</p>
	<p>Wybrany element można usunąć z pamięci.</p>

## 4. Pseudokod

### Liczba elementów listy

K01:  $c \leftarrow 0$  *zerujemy licznik*

K02: **Dopóki**  $p \neq \text{nil}$ , *w pętli przechodzimy przez kolejne elementy listy*  
    **wykonuj** kroki K03...K04

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

K04:  $p \leftarrow (p \rightarrow \text{next})$  *w p umieść zawartość pola next elementu wskazywanego przez p*

K05: **Zakończ** z wynikiem  $c$  *koniec, wynik w c*

## **Dodawanie elementu na początku listy**

K01: Utwórz nowy element listy

K02:  $p \leftarrow$  adres nowego elementu

K03:  $(p \rightarrow data) \leftarrow v$  *umieszczamy dane w elemencie*

K04:  $(p \rightarrow next) \leftarrow head$  *następnikiem będzie bieżący pierwszy element listy*

K05:  $head \leftarrow p$  *ustawiamy początek listy na nowy element*

K06: **Zakończ**

## Usuwanie elementu z początku listy

K01:  $p \leftarrow head$

*zapamiętaj pierwszy element*

K02: **Jeśli**  $p = nil$ ,  
**to zakończ**

*zakończ, jeśli lista jest pusta*

K03:  $head \leftarrow (p \rightarrow next)$

*początkiem listy będzie element następny*

K04: Usuń z pamięci element wskazany przez  $p$

K05: **Zakończ**

## Dodawanie elementu na koniec listy

K01: Utwórz nowy element

K02:  $e \leftarrow$  adres nowego elementu

K03:  $(e \rightarrow next) \leftarrow nil$

*inicjujemy pola nowego elementu*

K04:  $(e \rightarrow data) \leftarrow v$

K05:  $p \leftarrow head$

*w  $p$  ustawiamy początek listy*

K06: **Jeśli**  $p \neq nil$ ,  
**to idź do** kroku K09

*czy lista jest pusta?*

K07:  $head \leftarrow e$

*jeśli tak, to nowy element będzie pierwszym elementem listy*

K08: **Zakończ**

K09: **Dopóki**  $(p \rightarrow next) \neq nil$ ,  
**wykonuj**  $p \leftarrow (p \rightarrow next)$

*inaczej przechodzimy do ostatniego elementu listy*

K10:  $(p \rightarrow next) \leftarrow e$

*dołączamy nowy element za ostatnim na liście*

K11: **Zakończ**

### Usuwanie ostatniego elementu listy

K01:  $p \leftarrow head$

*pobieramy do p adres początku listy*

K02: **Jeśli**  $p = nil$ ,  
**to zakończ**

*jeśli lista jest pusta, kończymy*

K03: **Jeśli**  $(p \rightarrow next) \neq nil$ ,  
**to idź do** kroku K07

*sprawdzamy, czy lista jest jednoelementowa*

K04: Usuń z pamięci element wskazywany przez  $p$

K05:  $head \leftarrow nil$

*lista jednoelementowa staje się listą pustą*

K06: **Zakończ**

K07: **Dopóki**  $((p \rightarrow next) \rightarrow next) \neq nil$ ,  
wykonuj  $p \leftarrow (p \rightarrow next)$

*idziemy do przedostatniego elementu*

K08: Usuń z pamięci element  
wskazywany przez  $(p \rightarrow next)$

*usuwamy następny element, czyli ostatni*

K09:  $(p \rightarrow next) \leftarrow nil$

*przedostatni element nie ma już następnika*

K10: **Zakończ**

### **Dodawanie nowego elementu przed wybranym elementem listy**

K01:  $p \leftarrow head$

*pobieramy do p adres początku listy*

K02: **Jeśli**  $p \neq e$ ,  
**to idź do** kroku K05

*sprawdzamy, czy e nie jest pierwszym elementem listy*

K03: Wstaw nowy element na początku listy

K04: **Zakończ**

K05: **Dopóki**  $(p \rightarrow next) \neq e$ ,  
wykonuj  $p \leftarrow (p \rightarrow next)$

*przechodzimy do elementu poprzedzającego e*

K06: Utwórz nowy element

K07:  $(p \rightarrow next) \leftarrow$  adres nowego elementu *nowy element wstawiamy za element poprzedzający*

K08:  $((p \rightarrow next) \rightarrow next) \leftarrow e$  *inicjujemy nowy element*

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

K10: **Zakończ**

## **Dodawanie nowego elementu za elementem wybranym**

K01: Utwórz nowy element

K02:  $p \leftarrow$  adres nowego elementu

K03:  $(p \rightarrow next) \leftarrow (e \rightarrow next)$  *następnym elementem za nowym będzie następny element za e*

K04:  $(p \rightarrow data) \leftarrow v$

K05:  $(e \rightarrow next) \leftarrow p$  *nowy element wstawiamy za e*

K06: **Zakończ**



## Usuwanie wybranego elementu listy

K01: **Jeśli**  $head \neq e$ ,  
to idź do kroku K04

*sprawdzamy, czy usuwany element jest  
pierwszym na liście*

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

K03: **Zakończ**

K04:  $p \leftarrow head$

*w p ustawiamy początek listy*

K05: **Dopóki**  $(p \rightarrow next) \neq e$ ,  
wykonuj  $p \leftarrow (p \rightarrow next)$

*w p ustawiamy adres elementu  
poprzedzającego e*

K06:  $(p \rightarrow next) \leftarrow (e \rightarrow next)$

*odłączamy e od listy*

K07: Usuń z pamięci element  
wskazywany przez e

K08: **Zakończ**

## 5. Wnioski

Lista jednokierunkowa to lista "struktur danych" przy czym każda zawiera wskaźnik do następnego elementu. W liście jednokierunkowej poruszamy się od początku do końca listy. Każdy element listy składa się z co najmniej dwóch pól: klucza oraz pola wskazującego na następny element listy.

Lista jest strukturą danych, która wykorzystujemy, gdy mamy do czynienia z góry nieznaną ilością danych. Oczekuje się, że dane będą tego samego typu, ale wykorzystując unię można ograniczenie to pominąć, dzięki czemu ta struktura danych jest bardzo często wykorzystywana.

## **Bibliografia:**

[https://eduinf.waw.pl/inf/alg/001\\_search/0086.php](https://eduinf.waw.pl/inf/alg/001_search/0086.php)

[https://home.agh.edu.pl/~pkleczek/dokuwiki/doku.php?id=dydaktyka:aisd:2016:data-structures\\_stack-queue-lists](https://home.agh.edu.pl/~pkleczek/dokuwiki/doku.php?id=dydaktyka:aisd:2016:data-structures_stack-queue-lists)

<https://www.samouczekprogramisty.pl/struktury-danych-lista-wiazana/>

[https://achilles.tu.kielce.pl/portal/Members/84df831b59534bdc88bef09b15e73c99/archive/semestr-ii-2019-2020/pdf/pp2/lecture/pp2\\_lecture\\_6.pdf](https://achilles.tu.kielce.pl/portal/Members/84df831b59534bdc88bef09b15e73c99/archive/semestr-ii-2019-2020/pdf/pp2/lecture/pp2_lecture_6.pdf)