

## Wykład

### **Dynamiczne struktury danych - lista jednokierunkowa**



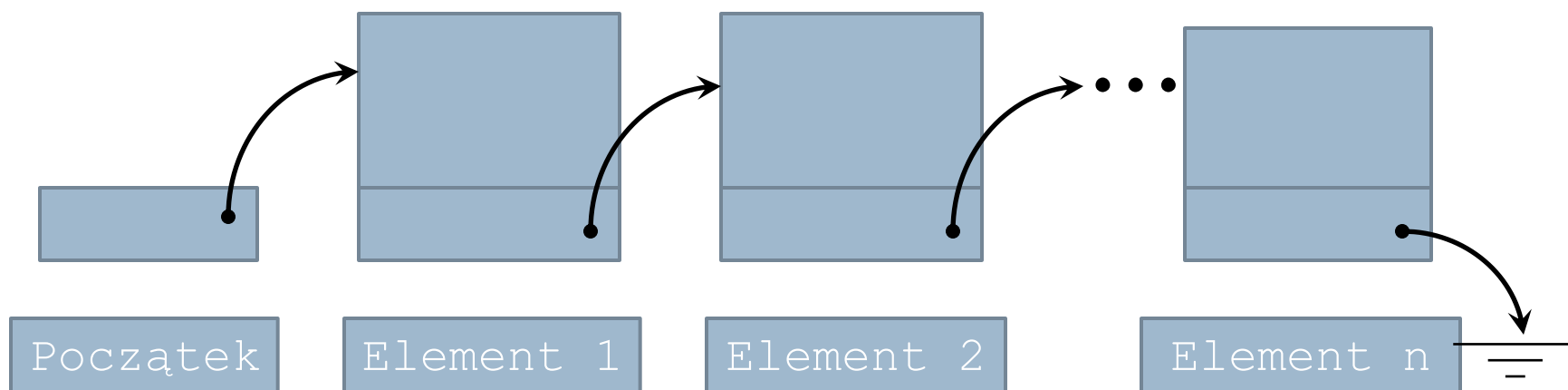
## Listy i drzewa

---

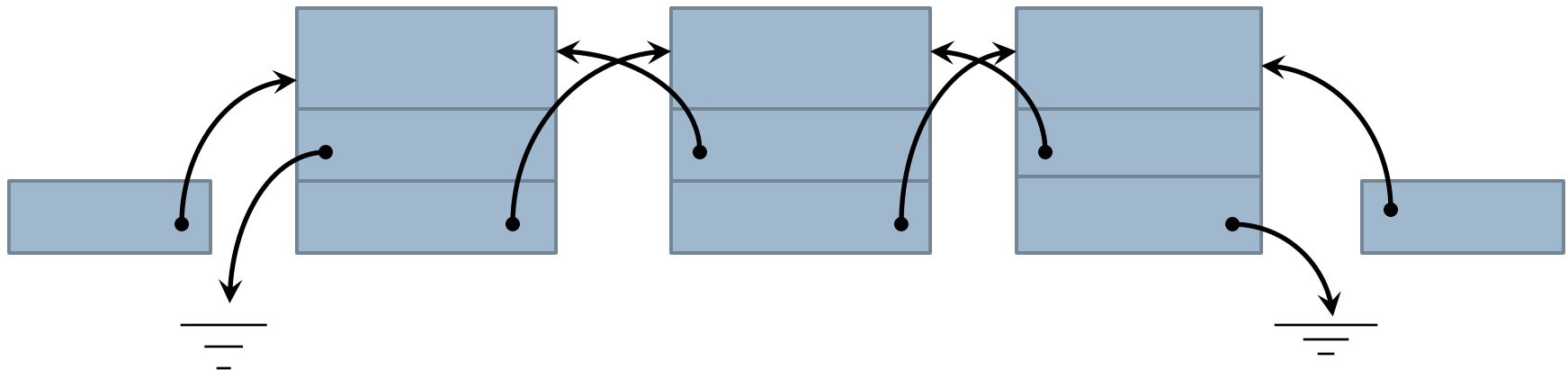
- ▶ Listy jednokierunkowe
- ▶ Listy jednokierunkowe uporządkowane
- ▶ Listy dwukierunkowe
- ▶ Listy dwukierunkowe uporządkowane
- ▶ Struktury drzewiaste

## Listy z wartownikami

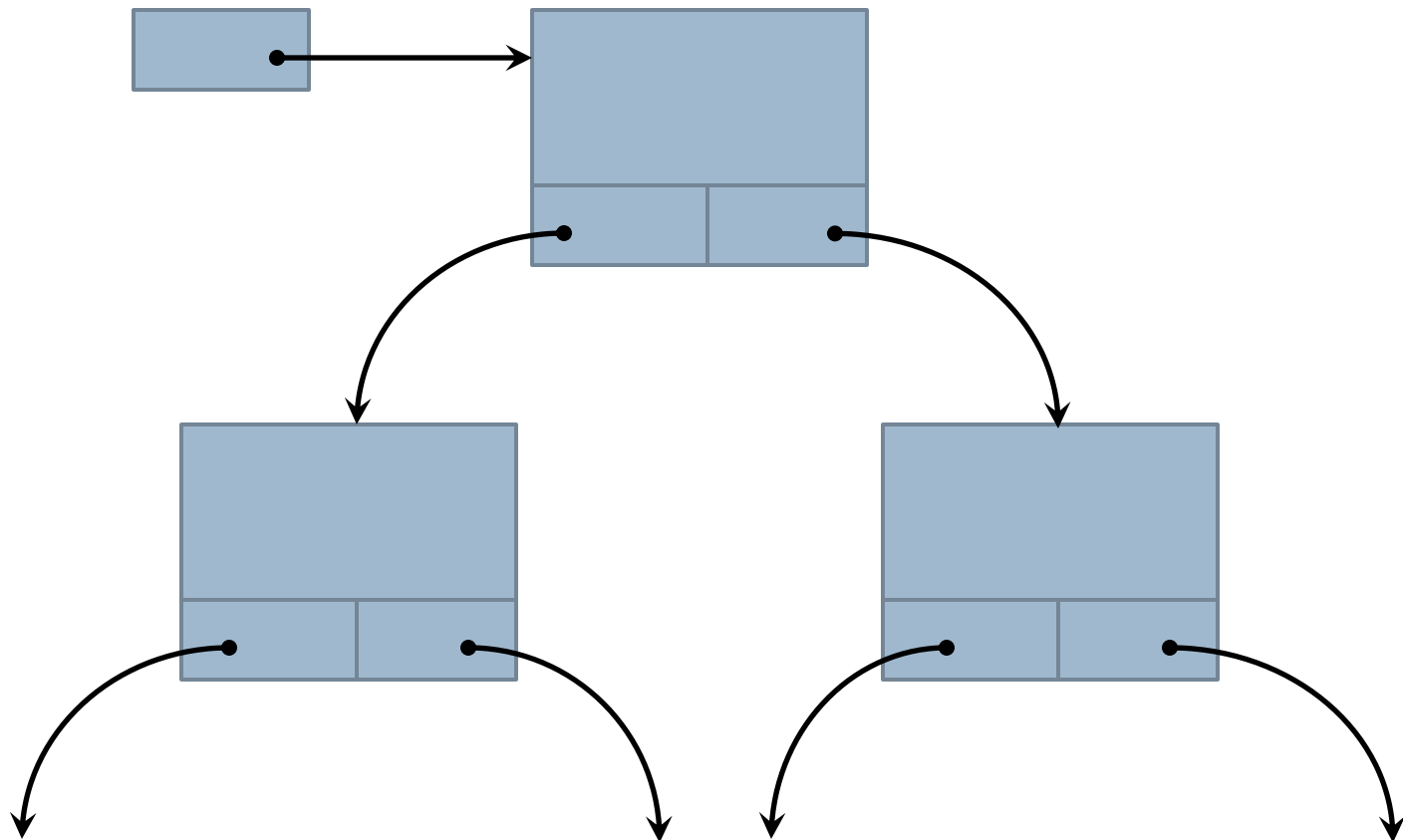
# Lista jednokierunkowa



# Lista dwukierunkowa



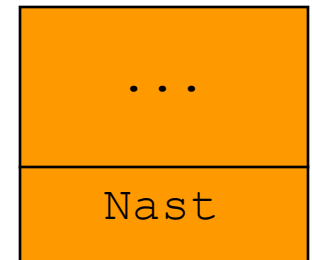
# Drzewo binarne



# Definiowanie listy

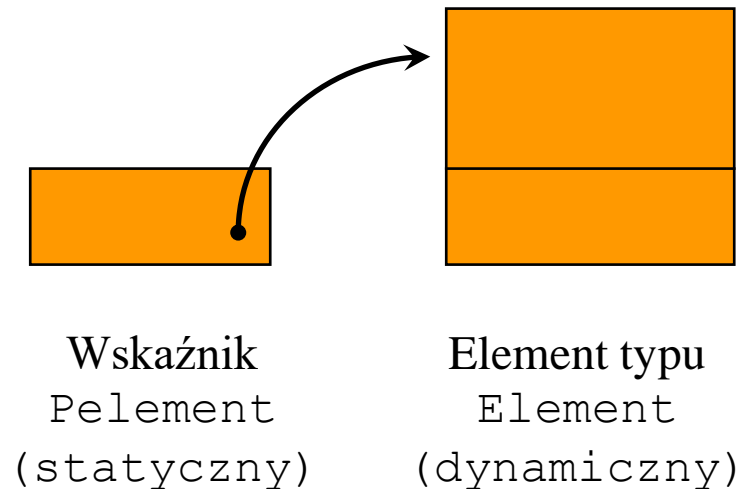
---

```
struct Telement
{
    int dana_1;
    string dana_2;
    . . . . .
    Telement * nast;
};
```

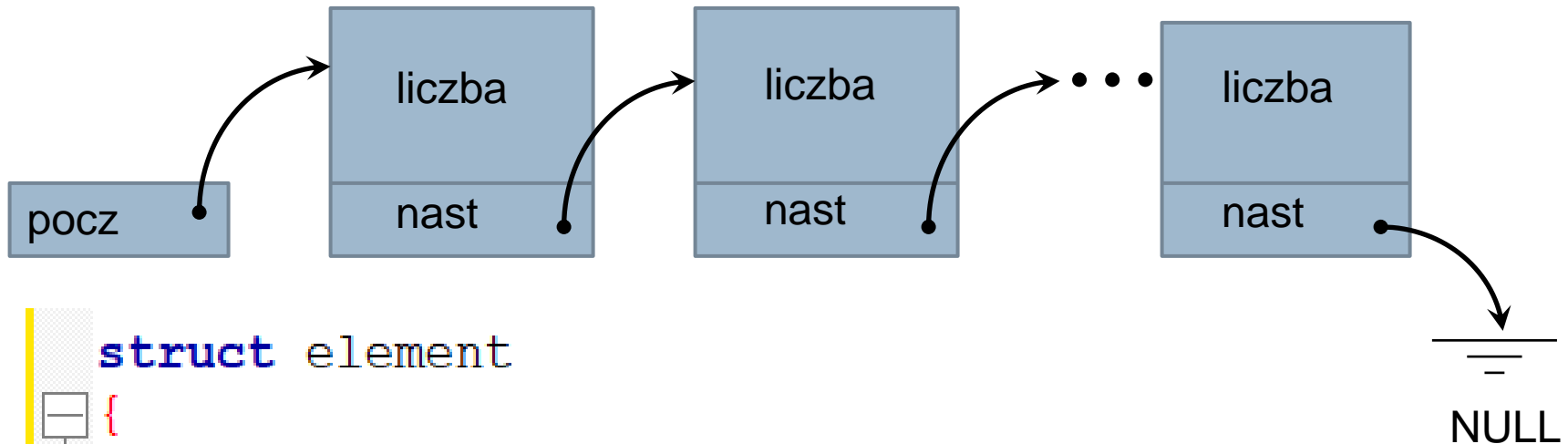


# Instrukcje **New** i **Dispose**

```
element *Pelement;  
Pelement = new element;  
.....  
.....  
delete Pelement;
```



# Lista jednokierunkowa



```
struct element  
{  
    int liczba;  
    element * nast;  
};
```

Struktura **element** zawiera dwa pola: pole **liczba**, które przechowuje dane (może być więcej pól dowolnego typu) oraz pole wskaźnikowe **nast**, które może przechować wskaźnik na taką samą strukturę typu **element**.



# Operacje na elementach listy

---

- Przejście przez listę
- Wstawianie nowego elementu na początek listy
- Wstawianie nowego elementu po danym elemencie listy
- Usunięcie pierwszego elementu listy
- Usunięcie z listy elementu znajdującego się po danym elemencie



## Przejdźcie przez listę

---

W przypadku listy jednokierunkowej możemy się po niej poruszać tylko w jednym kierunku: od początku do końca.

Z uwagi na sposób powiązania ze sobą elementów listy, do jej przechodzenia potrzebna jest zmienna typu wskaźnik, która będzie wskazywała na kolejne elementy.

- Na początku algorytmu zmienna ta powinna wskazywać pierwszy element na liście.
- W pętli przetwarzamy element wskazywany przez tą zmienną, po czym za nową wartość zmiennej przyjmuje się adres następnego elementu listy.
- Adres ten jest przechowywany w polu nast elementu bieżącego.
- Listę przechodzimy do momentu, aż zmienna wskazująca przyjmie wartość NULL (zero).
- Stanie się tak po wyjściu z ostatniego elementu listy, w którego polu next przechowywany jest adres NULL.

# Przejsćie przez listę

```
element * p = pocz;
```

```
while (p)
```

```
{
```

```
...
```

```
p = p->next;
```

```
}
```

Przejsćie przez listę

Funkcja zliczająca  
elementy listy

```
unsigned suma(element * p)
```

```
{
```

```
    unsigned c |= 0; // zerujemy licznik
```

```
    while (p)
```

```
    {
```

```
        c++; // zwiększamy licznik o 1
```

```
        p = p->nast;
```

```
    }
```

```
    return c;
```

```
}
```

## Przejdźcie przez listę

---

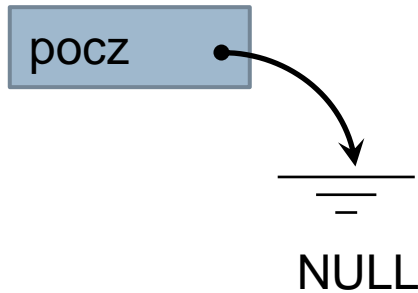
Wypisanie wszystkich elementów listy.

```
void wypisz()  
{  
    element *p = pocz;  
    while (p)  
    {  
        cout << p->liczba<< "  ";  
        p=p->nast;  
    }  
}
```

## Tworzenie listy:

Pierwszym krokiem jest stworzenie listy pustej.

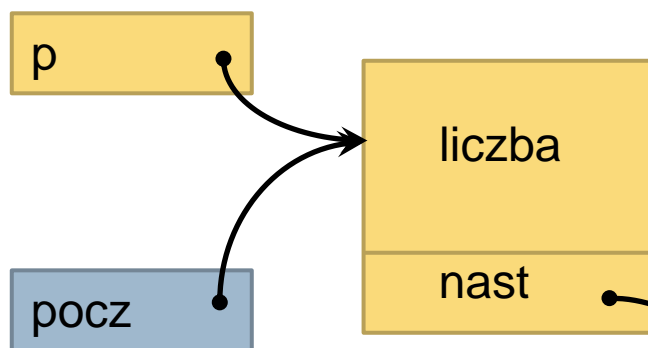
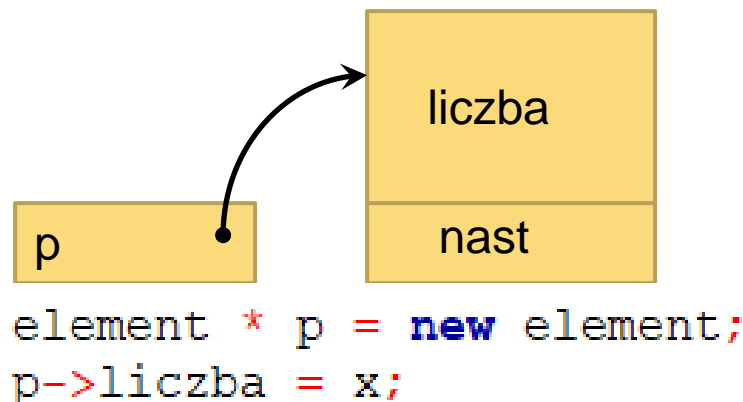
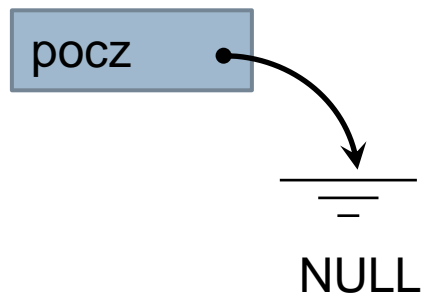
```
element * pocz = NULL;
```



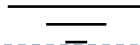
```
struct element
{
    int liczba;
    element * nast;
} | * pocz = NULL; //tworzenie listy pustej
```

# Tworzenie listy

Mając pustą listę możemy wstawiać elementy na jej początek.

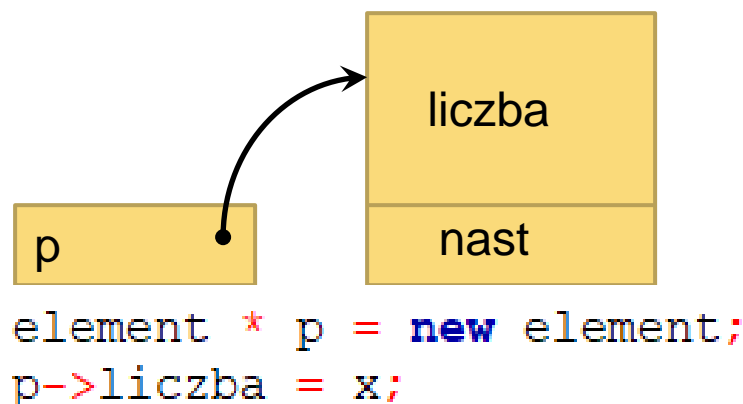
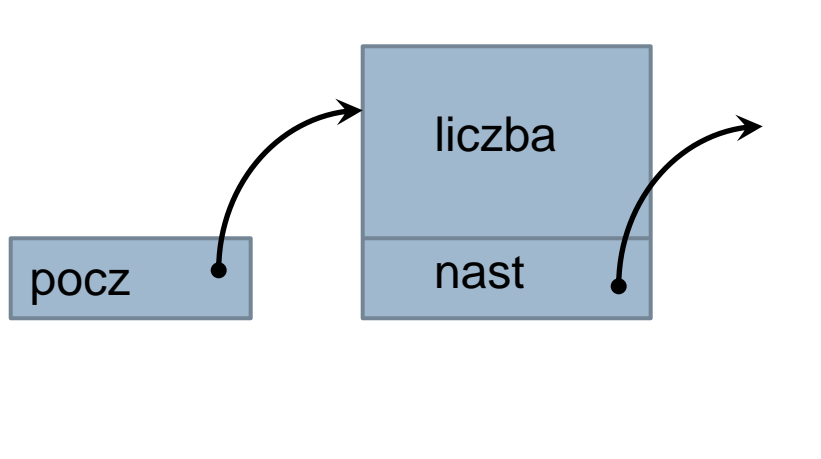


```
p->nast = pocz;
pocz = p; // nowy początek listy
```

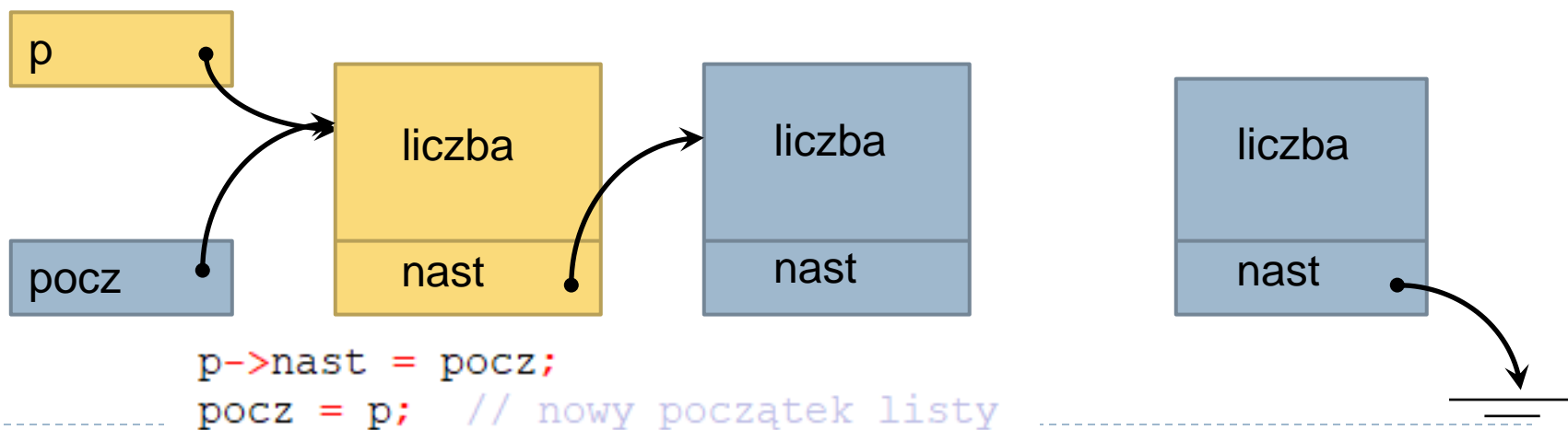


# Tworzenie listy

Jeżeli lista już istnieje postępujemy identycznie.



```
element * p = new element;  
p->liczba = x;
```

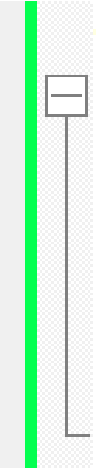


```
p->nast = pocz;  
pocz = p; // nowy początek listy
```

## Tworzenie listy

---

Przykładowa funkcja wstawiająca na początek listy element z wartością przekazaną w parametrze.




```
void WstawNaPocz(int x)
{
    element * p = new element;
    p->liczba = x;
    p->nast = pocz;
    pocz = p; // nowy początek listy
}
```



## Usuwanie elementów listy

Najprościej usunąć element z początku listy.



```
void usunPierwszy()
{
    element * p = pocz;
    if (p != NULL) // jeśli lista nie jest pusta
    {
        pocz = p->nast; // nowy początek
        delete p;       // usuń element z pamięci
    }
}
```

1. Ustawiamy pomocniczy wskaźnik p na początek listy (pierwszy element)
2. Jeśli lista nie jest pusta:
  - a) przesuwamy wskaźnik początku na kolejny element (wskazywany przez pole nast. Pierwszego)
  - b) Usuwamy pierwszy element (wskazywany przez wskaźnik pomocniczy p)

## Usuwanie całej listy

Usunięcie całej listy jest odbywa się podobnie jak jej wypisanie

```
void usunListe()
```

```
{  
    element *p;  
    while(p)  
    {  
        p=pocz; //zapamiętujemy wskaźnik do pierwszego elementu  
        pocz=pocz->nast; //przesuwamy początek na kolejny element  
        delete p; //usuwany zapamiętany w zmiennej pomocniczej p  
    }  
}
```

1. Zapamiętujemy wskaźnik do pierwszego elementu w zmiennej p,
2. Przesuwamy początek na kolejny element.
3. Usuwany zapamiętany w zmiennej pomocniczej p

```

1  #include <iostream>
2
3  using namespace std;
4
5  struct element
6  {
7      int liczba;
8      element * nast;
9  } * pocz = NULL; //tworzenie listy pustej
10
11  unsigned suma();
12  void WstawNaPocz(int x);
13  void wypisz()
14
15  int main()
16  {
17      for (int i=0; i<20; i++)
18          WstawNaPocz(i);
19      wypisz();
20      return 0;
21  }
22
23  unsigned suma()
24  {
25      element *p = pocz; //ustawiamy wskaznik
26                          //pomocniczy na poczatek listy
27      unsigned c = 0; // zerujemy licznik
28
29      while (p!=NULL)
30      {
31          c++; // zwiększamy licznik o 1
32          p = p->nast;
33      }
34      return c;
35  }

```

```
37 void wypisz()
38 {
39     element *p = pocz;
40     while(p)
41     {
42         cout << p->liczba<< " ";
43         p=p->nast;
44     }
45 }
46 void WstawNaPocz(int x)
47 {
48     element * p = new element;
49     p->liczba = x;
50     p->nast = pocz;
51     pocz = p; // nowy początek listy
52 }
53 void usunPierwszy()
54 {
55     element * p = pocz;
56     if(p!= NULL) // jeśli lista nie jest pusta
57     {
58         pocz = p->nast; // nowy początek
59         delete p; // usuń element z pamięci
60     }
61 }
62 void usunListe()
63 {
64     element *p;
65     while(p)
66     {
67         p=pocz; //zapamiętujemy wskaźnik do pierwszego elementu
68         pocz=pocz->nast; //przesuwamy początek na kolejny element
69         delete p; //usuwany zapamiętany w zmiennej pomocniczej p
70     }
71 }
```

## Dołączanie nowego elementu na końcu listy

```
void dodajNaKoniec(int x)
{
    element * p, * n;
    n = new element; // tworzymy nowy element
    n->nast = NULL;   // będzie on ostatnim elem. listy
    n->dana = x;       // wstawimy podaną wartość
    p = pocz;
    if(p!=NULL)        //jeśli lista nie jest pusta
    {
        while(p->nast) p = p->nast; //szukamy końca listy
        p->nast = n;               //wstawiamy utworzony element
                                   //na końcowy zamiast NULL
    }
    else pocz = n;               //lista była pusta
}
```

- jeśli lista nie jest pusta
  - szukamy końca listy ( $p \rightarrow \text{nast} \neq \text{NULL}$  – nie możemy wyjść za listę)
  - wstawiamy utworzony element na końcowy listy (po elemencie wskazywanym przez wskaźnik p)
- Jeśli lista była pusta – wstawiamy nowy element jako pierwszy.

## Usunięcie ostatniego elementu z listy

```
void usunOstatni()
{
    element * p = pocz;
    if(p!=NULL)
    {
        if(p->nast) //usuwanie jeśli jest więcej elementów niż tylko startowy
        {
            while(p->nast->nast) p = p->nast; // szukamy przedostatniego z listy
            delete p->nast; //usuwamy ostatniego
            p->nast = NULL; //oznaczamy nowy koniec listy
        }
        else //był na liście tylko startowy
        {
            delete p;
            pocz = NULL; // lista staje się listą pustą
        }
    }
}
```

- Jeśli jest więcej elementów niż tylko startowy:
  - szukamy przedostatniego elementu z listy ( $p \rightarrow \text{nast} \rightarrow \text{nast}$ )
  - usuwamy ostatni element
  - oznaczamy nowy koniec listy
- Jeśli na liście był tylko element startowy
  - lista staje się listą pustą

## Dodanie nowego elementu przed wskazanym

Adres elementu przed który należy wstawić nowy element podany jest w parametrze funkcji

```
void ododajPrzedWskazanym(element * t, int x)
{
    element * p = pocz;

    if(t == pocz) dodaj(x);    //zwykle dodawanie na początku
    else
    {
        while(p->nast != t) p = p->nast; //szukamy t
        p->nast = new element;
        p->nast->nast = t;
        p->nast->dana = x;
    }
}
```

- Jeśli wskazany element to początek listy:
  - dodajemy element na początek (odwołanie do funkcji opisanej na slajdzie 14)
- Jeśli wskazany element nie jest początkiem listy:
  - przeszukujemy listę od początku, żeby znaleźć element poprzedzający wskazany,
  - dodajemy nowy element po odnalezionym (czyli przed wskazanym)

## Dodanie nowego elementu przed wskazanym

Adres elementu przed który należy wstawić nowy element podany jest w parametrze funkcji

```
void ododajPrzedWskazanym(element * t, int x)
{
    element * p = pocz;

    if(t == pocz) dodaj(x);    //zwykle dodawanie na początku
    else
    {
        while(p->nast != t) p = p->nast; //szukamy t
        p->nast = new element;
        p->nast->nast = t;
        p->nast->dana = x;
    }
}
```

- Jeśli wskazany element to początek listy:
  - dodajemy element na początek (odwołanie do funkcji opisanej na slajdzie 16)
- Jeśli wskazany element nie jest początkiem listy:
  - przeszukujemy listę od początku, żeby znaleźć element poprzedzający wskazany,
  - dodajemy nowy element po odnalezionym (czyli przed wskazanym)



## Tworzenie listy posortowanej

---

Tworzenie listy posortowanej polega na wstawianiu elementów w odpowiedniej kolejności.

W odróżnieniu od tablic niej jest konieczne przesuwanie danych, a tylko „dowiązywanie” elementów na właściwe miejsce.

- Po liście przesuwamy parę wskaźników – jeden wskazuje na pierwszy element , który nie spełnia warunku, drugi na poprzedni.
- Nowy element wstawiamy pomiędzy te wskaźniki (wskazywane przez nie elementy)
- Jeśli lista jest pusta (rozpoznamy po tym, że pierwszy wskaźnik nadal wskazuje na początek listy, a drugi pozostał na NULL):
  - dodajemy element na początek
- Jeśli wstawić mamy element na początek listy (rozpoznamy po tym, że drugi wskaźnik wskazuje na początek listy):
  - Wstawiamy element na początek listy
- Jeżeli wstawimy element w środek lub na koniec listy (pozostałe przypadki)
  - Wstawimy element pomiędzy dwa wskaźniki pomocnicze.

# Tworzenie listy posortowanej

```
void wstawRosnaco(int x)
{
    element * p = pocz, *pop=NULL;
    //przygotowujemy dwa wskazniki - na elementy pomiedzy ktore wstawimy nowy
    while (p && p->dana<x) // przesuwamy pare wskaznikow az znajdzemy
        //pierwszy wiekszy lub rowny element
    {
        pop=p;
        p=p->nast;
    }
    element *temp= new element;
    temp->dana=x;
    if (p==pocz) //jeżeli lista jest pusta
    {
        temp->nast=pocz;
        pocz=temp;
    } else
    if (pop==pocz) //jeżeli wstawiamy na poczatek listy
    {
        temp->nast=p;
        pocz=temp;
    } else //jesli wstawimy w srodek lub na koniec listy
    {
        temp->nast=p;
        pop->nast=temp;
    }
}
```

```

1  #include <iostream>
2  #include <cstring>
3
4  using namespace std;
5
6  struct element
7  {
8      int dana;
9      element * nast;
10 } * pocz = NULL;
11
12 void dodaj(int x);
13 void wypisz();
14 int usun();
15 element * wyszukaj(int x);
16 void dodajNaKoniec(int x);
17 void usunOstatni();
18 void wstawRoznaco(int x);
19
20
21 int main()
22 {
23     wstawRoznaco(5);
24     wstawRoznaco(11);
25     wstawRoznaco(3);
26     wstawRoznaco(50);
27     wstawRoznaco(200);
28     wstawRoznaco(50);
29     wypisz();
30
31     cout<<endl;
32     while (pocz!=NULL)
33     {
34         cout <<"usunieto: "<<usun()<<endl;;
35     }
36     return 0;
37 }

```

```

39
40 void dodaj(int x)
41 {
42     element *p = new element;
43     p->dana = x;
44     p->nast = pocz;
45     pocz = p;
46 }
47 void wypisz()
48 {
49     element *p =pocz;
50     while (p!=NULL)
51     {
52         cout <<p->dana<<" ";
53         p=p->nast;
54     }
55 }
56 int usun()
57 {
58     int x;
59     element *p=pocz;
60     if (p!=NULL)
61     {
62         pocz=pocz->nast;
63         x= p->dana;
64         delete p;
65         return x;
66     }
67     return 0;
68 }
69
70 element * wyszukaj(int x)
71 {
72     element *p = pocz;
73     while (p!=NULL && p->dana!=x)
74     {
75         p=p->nast;
76     }
77     return p;
78 }

```

```

80
81 void dodajNaKoniec(int x)
82 {
83     element * p, * n;
84     n = new element; // tworzymy nowy element
85     n->nast = NULL;   // będzie on ostatnim elem. listy
86     n->dana = x;      // wstawimy podaną wartość
87     p = pocz;
88     if(p!=NULL)       //jeśli lista nie jest pusta
89     {
90         while(p->nast) p = p->nast; //szukamy końca listy
91         p->nast = n;             //wstawiamy utworzony element
92                                 //na końcowy zamiast NULL
93     }
94     else pocz = n;             //lista była pusta
95 }
96
97 void usunOstatni()
98 {
99     element * p = pocz;
100     if(p!=NULL)
101     {
102         if(p->nast) //usuwanie jeśli jest więcej elementów niż tylko startowy
103         {
104             while(p->nast->nast) p = p->nast; // szukamy przedostatniego z listy
105             delete p->nast; //usuwamy ostatniego
106             p->nast = NULL; //oznaczamy nowy koniec listy
107         }
108         else //był na liście tylko startowy
109         {
110             delete p;
111             pocz = NULL; // lista staje się listą pustą
112         }
113     }
114 }

```

```

117 void ododajPrzedWskazany (element * t, int x)
118 {
119     element * p = pocz;
120
121     if (t == pocz) dodaj(x); //zwykle dodawanie na początku
122     else
123     {
124         while (p->nast != t) p = p->nast; //szukamy t
125         p->nast = new element;
126         p->nast->nast = t;
127         p->nast->dana = x;
128     }
129 }
130 void wstawRosnaco (int x)
131 {
132     element * p = pocz, *pop=NULL;
133     //przygotowujemy dwa wskazniki - na elementy pomiedzy ktore wstawimy nowy
134     while (p && p->dana<x) // przesuwamy pare wskaznikow az znajdziemy
135         //pierwszy wiekszy lub rowny element
136     {
137         pop=p;
138         p=p->nast;
139     }
140     element *temp= new element;
141     temp->dana=x;
142     if (p==pocz) //jezeli lista jest pusta
143     {
144         temp->nast=pocz;
145         pocz=temp;
146     } else
147     if (pop==pocz) //jezeli wstawiamy na poczatek listy
148     {
149         temp->nast=p;
150         pocz=temp;
151     } else //jesli wstawimy w srodek lub na koniec listy
152     {
153         temp->nast=p;
154         pop->nast=temp;
155     }
156 }

```