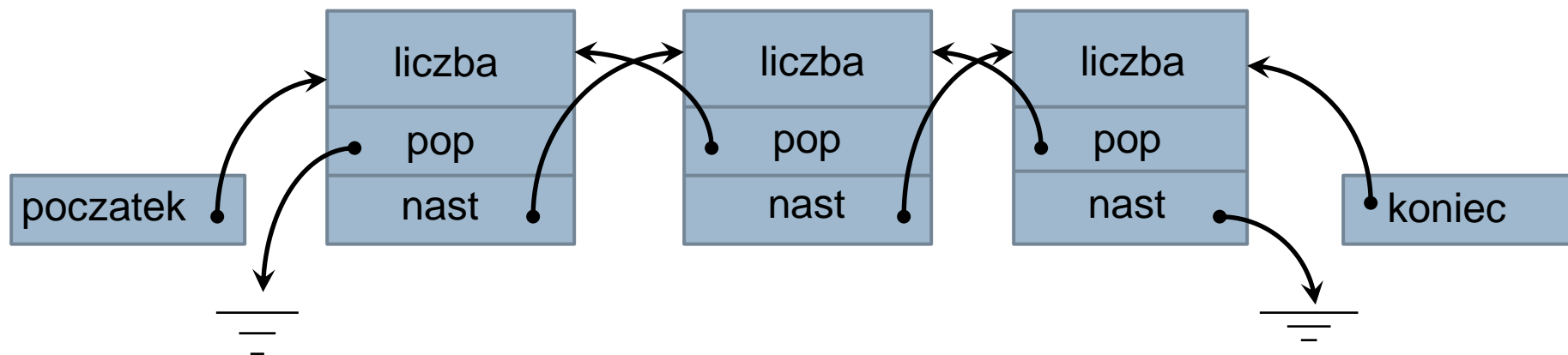


Wykład:

Dynamiczne struktury danych - lista dwukierunkowa

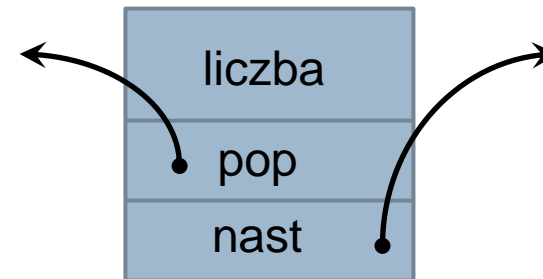
Lista dwukierunkowa



Definiowanie listy

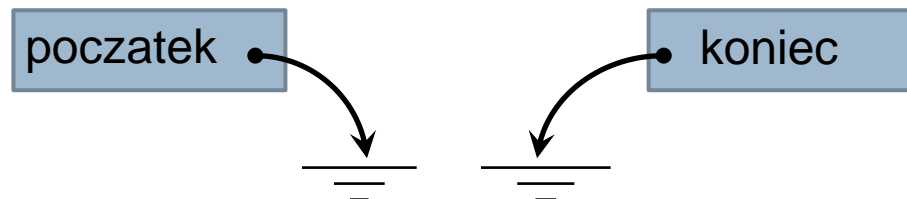
Definiowanie elementu listy

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



Tworzenie listy pustej

```
element * poczatek = NULL;  
element * koniec = NULL;
```





Operacje na liście dwukierunkowej

Na kolejnych slajdach przedstawiono podstawowe operacje na liście dwukierunkowej:

- dodanie elementu na początku listy ,
- dodanie elementu na końcu listy,
- usunięcie elementu o wskazanej wartości (kluczu)
- wypisanie listy od początku,
- wypisanie listy od początku

Dodanie elementu na początku listy

```
void wstawNaPoczątek(int x)
```

```
{  
    element * p = new element;  
    p->liczba = x;  
    p->pop = NULL;  
    if (początek == NULL)  
    {  
        p->nast = NULL;  
        początek = NULL;  
        koniec = NULL;  
    } else  
    {  
        początek->pop = p;  
        p->nast = początek;  
        początek = p;  
    }  
}
```

Dodanie elementu na końcu listy

```
void wstawNaKoniec (int x)
{
    element * p = new element;
    p->liczba = x;
    p->nast= NULL;
    if (koniec==NULL)
    {
        p->pop=NULL;
        poczatek=NULL;
        koniec=NULL;
    } else
    {
        koniec->nast=p;
        p->pop=koniec;
        koniec=p;
    }
}
```

Wypisanie listy od początku

```
void wypiszOdPoczatku()
```



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

Wypisanie listy od końca

```
void wypiszOdKonca()
```

```
{
```

```
    element *p = koniec;
```

```
    while (p)
```

```
    {
```

```
        cout << p->liczba<<" ";
```

```
        p=p->pop;
```

```
    }
```

```
}
```


Usuniecie elementu o wskazanej wartości (kluczu)

```
56 void znajdziUsun(int x)
57 {
58     element *temp=poczatek, *n, *p;
59     while (temp)
60     {
61         if (temp->liczba==x)
62         {
63             if (temp->pop==NULL)           //element jest pierwszy
64             {
65                 if (temp->nast==NULL)      //jest także ostatni
66                     // CZYLI JEDYNY NA LISCIE
67                     poczatek=NULL;
68                     koniec=NULL;
69             } else                          //pierwszy, ale nie jedyny
70             {
71                 n=temp->nast;
72                 n->pop=temp->pop;
73                 poczatek=n;
74             }
75         }
    }
```

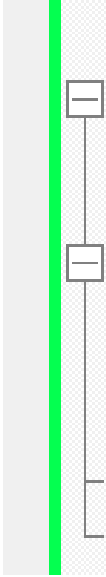
Usuniecie elementu o wskazanej wartości (kluczu)

c.d.

```
76      else                                     //nie jest pierwszy
77      {
78          if (temp->nast==NULL) //jest ostatni
79          {                                     //ale nie jedyny
80              p=temp->pop;
81              p->nast=temp->nast;
82              koniec=temp->pop;
83          } else
84          {
85              n=temp->nast;
86              p=temp->pop;
87              p->nast=temp->nast;
88              n->pop=temp->pop;
89          }
90      }
91      p=temp;
92      temp=temp->nast;
93      delete p;
94  } else temp=temp->nast;
95  }
96  }
```

Usuniecie listy

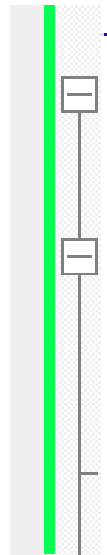
```
void usunListe()
```



```
{  
    element *p;  
    while(koniec)  
    {  
        p=koniec;  
        koniec=koniec->pop;  
        delete p;  
    }  
}
```

Wersja1:
usuwamy listę od końca

```
void usunListe2()
```



```
{  
    element *p;  
    while(poczatek)  
    {  
        p=poczatek;  
        poczatek=poczatek->nast;  
        delete p;  
    }  
}
```

Wersja2:
usuwamy listę od początku