Algorytmy i struktury danych

Dynamiczne struktury danych:
stos, kolejka, lista

Witold Marańda
maranda@dmcs.p.lodz.pl

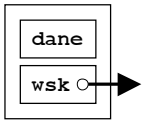
Dynamiczne struktury danych

Zmienne dynamiczne pozwalają tworzyć struktury danych o dowolnej (i zmiennej) wielkości oraz skomplikowanej budowie wewnętrznej, lepiej wykorzystując dostępną pamięć operacyjną komputera.

Szczególne znaczenie mają struktury danych o budowie liniowej, w której każdy obiekt ma swojego poprzednika i swój następnik. Struktury takie realizuje się najczęściej używając jako elementarnego składnika zmiennej dynamicznej typu rekordowego, która zawiera pola danych oraz pola będące wskaźnikami do sąsiadujących elementów struktury.

Na potrzeby dalszych przykładów, zdefiniowany zostanie następujący element składowy dynamicznej struktury danych:

```
TYPE element =   RECORD
                     dane : T0;
                     wsk : ^element;
                 END
```



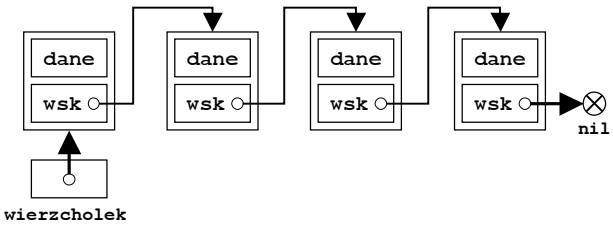
Ponadto, przyjmuje się, że wartość zmiennej wskaźnikowej, która nie wskazuje żadnego obiektu równa jest pewnej umownej stałej o nazwie '**nil**' (tzw. wskazanie puste).

Stos

Stos jest dynamiczną strukturą danych o charakterze liniowo uporządkowanym, w którym wstawianie, usuwanie i dostęp do elementów są możliwe tylko w jednym końcu, zwanym ‘wierzchołkiem stosu’.

Stos jest strukturą jednokierunkową, tj. każdy element „zna” położenie jedynie swego poprzednika. Położenie wierzchołka stosu wskazywane jest przez zmienną wskaźnikową ‘wierzchołek’ stosu. Pierwszy element stosu posiada wsazanie o wartości nil (puste).

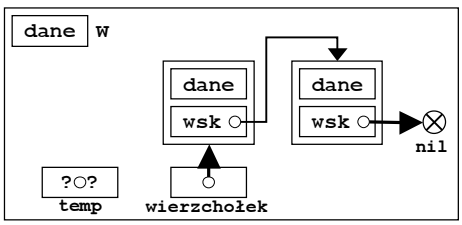
Stos jest strukturą typu **LIFO** (*last-in, first-out*).



3

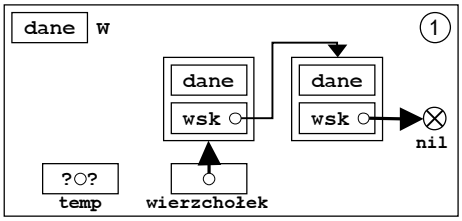
Stos – dodawanie elementów

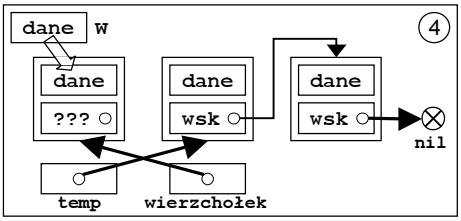
```
VAR
   wierzchołek : ^element;
   W : T0;
   temp : ^element;     {zmienna pomocnicza}
                                    (1)
BEGIN
   temp := wierzchołek;             (2)
   NEW(wierzchołek);                (3)
   wierzchołek^.dane := W;          (4)
   wierzchołek^.wsk := temp;        (5)
END
```

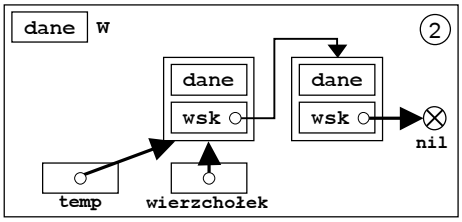


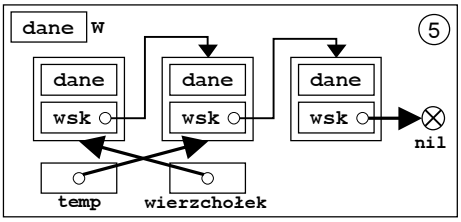
4

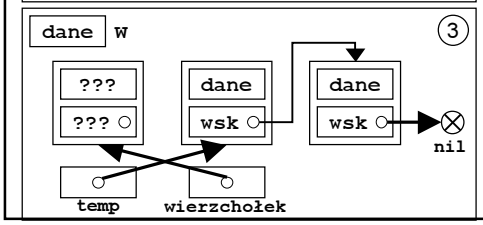
Stos – dodawanie elementów







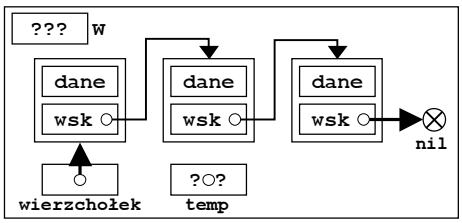





```
VAR
    wierzchołek : ^element;
    W : T0;
    temp : ^element;              (1)
BEGIN
    temp := wierzchołek;          (2)
    NEW(wierzchołek);             (3)
    wierzchołek^.dane := W;       (4)
    wierzchołek^.wsk := temp;     (5)
END
```

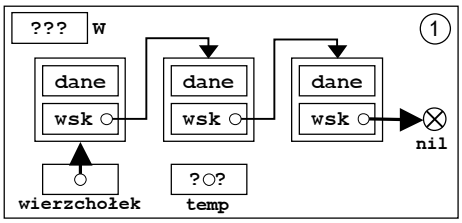
Stos – pobieranie elementów

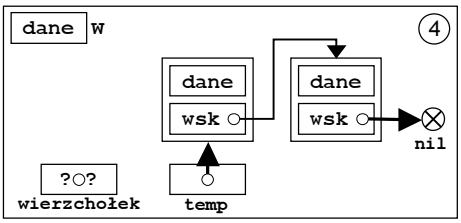
```
VAR
    wierzchołek : ^element;
    W : T0;
    temp : ^element;        {zmienna pomocnicza}
                                          (1)
BEGIN
    IF wierzchołek <> nil THEN
    BEGIN
        W := wierzchołek^.dane;           (2)
        temp := wierzchołek^.wsk;         (3)
        DISPOSE(wierzchołek);             (4)
        wierzchołek := temp;              (5)
    END
END
```

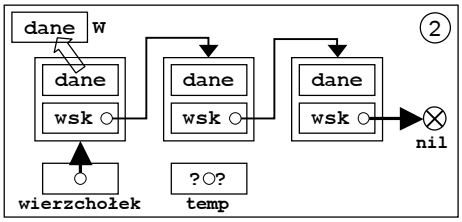


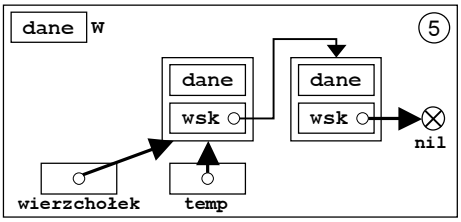
6

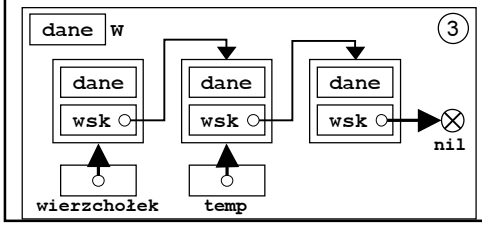
Stos – pobieranie elementów











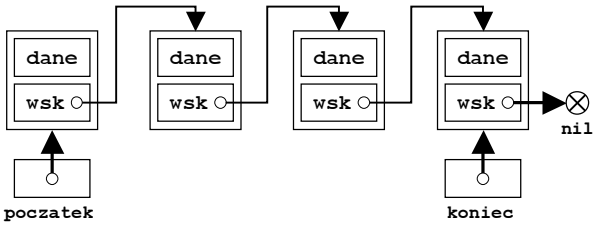

```
VAR
     wierzchołek : ^element;
     W : T0;
     temp : ^element;                    (1)
BEGIN
     IF wierzchołek <> nil THEN
     BEGIN
          W := wierzchołek^.dane;        (2)
          temp := wierzchołek^.wsk;      (3)
          DISPOSE(wierzchołek);          (4)
          wierzchołek := temp;           (5)
     END
END
```

Kolejka

Kolejka jest dynamiczną strukturą danych o charakterze liniowo uporządkowanym, w której można dołączyć składnik tylko na jednym końcu (zwanym końcem kolejki), a usunąć tylko w drugim końcu (zwanym początkiem kolejki).

Kolejka jest strukturą jednokierunkową, tj. każdy element zna położenie jedynie swego poprzednika. Położenie początku kolejki wskazywane jest przez zmienną wskaźnikową ‘początek’, a koniec kolejki przez zmienną wskaźnikową ‘koniec’. Element na końcu kolejki nie wskazuje na żaden obiekt poprzedzający (nil).

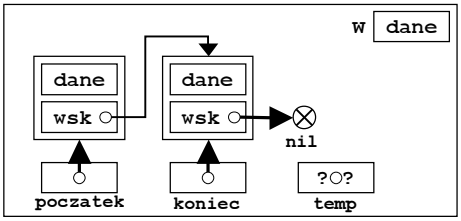
Kolejka jest strukturą typu **FIFO** (*first-in, first-out*).



8

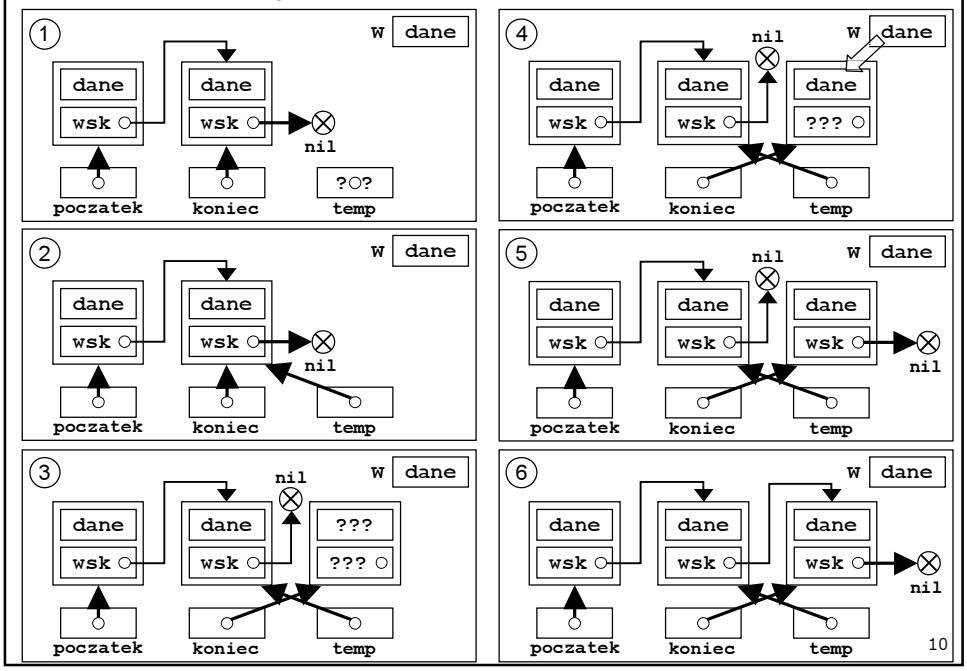
Kolejka – dodawanie elementów

```
VAR
   koniec : ^element;
   W : TO;
   temp : ^element      {zmienna pomocnicza}
                                  (1)
BEGIN
   temp := koniec;                (2)
   NEW(koniec);                   (3)
   koniec^.dane := W;             (4)
   koniec^.wsk := nil;            (5)
   IF temp <> nil THEN
        temp^.wsk := koniec;      (6)
END
```



9

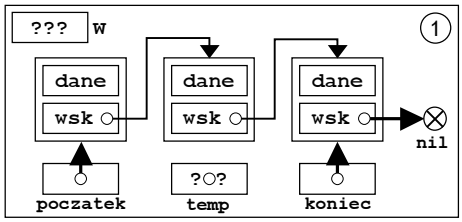
Kolejka – dodawanie elementów



10

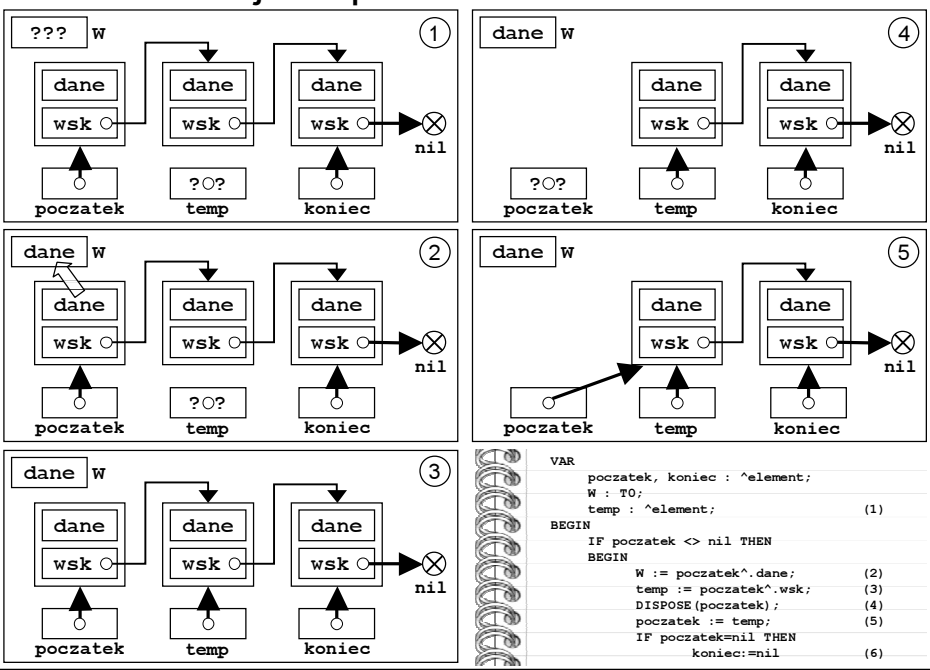
Kolejka – pobieranie elementów

```
VAR
    poczatek, koniec : ^element;
    W : T0;
    temp : ^element;        {zmienna pomocnicza}
                                        (1)
BEGIN
    IF poczatek <> nil THEN
    BEGIN
        W := poczatek^.dane;            (2)
        temp := poczatek^.wsk;          (3)
        DISPOSE(poczatek);              (4)
        poczatek := temp;               (5)
        IF poczatek=nil THEN
                    koniec:=nil         (6)
    END
END
```



11

Kolejka – pobieranie elementów




```
VAR
    poczatek, koniec : ^element;
    W : T0;
    temp : ^element;                (1)
BEGIN
    IF poczatek <> nil THEN
    BEGIN
        W := poczatek^.dane;        (2)
        temp := poczatek^.wsk;      (3)
        DISPOSE(poczatek);          (4)
        poczatek := temp;           (5)
        IF poczatek=nil THEN
                koniec:=nil         (6)
```

Lista

Listą nazywamy liniowo uporządkowany zbiór składników, z którego w dowolnym miejscu można usunąć lub dołączyć składnik.

Położenie początku listy wskazywane jest przez zmienną wskaźnikową ‘początek’, a koniec listy przez zmienną wskaźnikową ‘koniec’. Ponadto zmienna wskaźnikowa ‘bieżący’ wskazuje na miejsce listy którego dotyczy aktualna operacja (usuniecie lub dopisanie elementu).

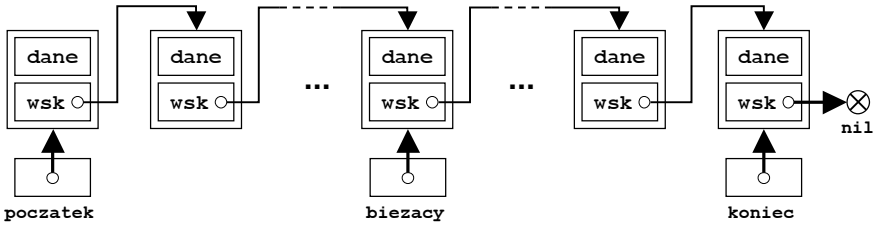
W zależności od powiązań pomiędzy składnikami wyróżnia się:

1. listy jednokierunkowe – dla każdego składnika (z wyjątkiem ostatniego) określony jest jego poprzednik, podobnie jak dla kolejki;
2. listy dwukierunkowe – dla każdego składnika określony jest jego poprzednik (z wyjątkiem ostatniego) i następnik (z wyjątkiem pierwszego);
3. listy cykliczne (jedno lub dwukierunkowe) – nie występuje składnik pierwszy ani ostatni, a dla każdego składnika określony jest jego poprzednik (i następnik - dla list dwukierunkowych)

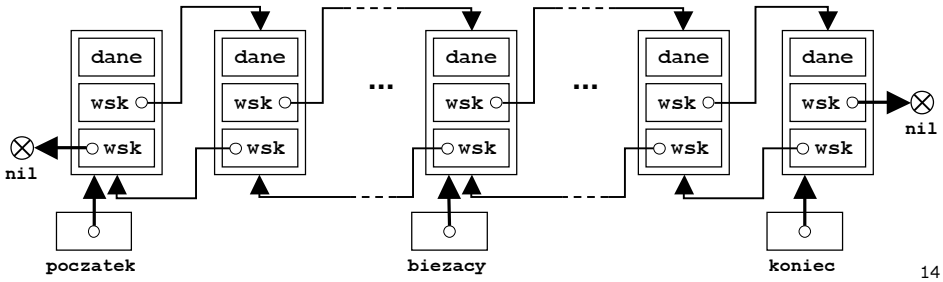
13

Listy jedno- i dwukierunkowe

Lista jednokierunkowa



Lista dwukierunkowa

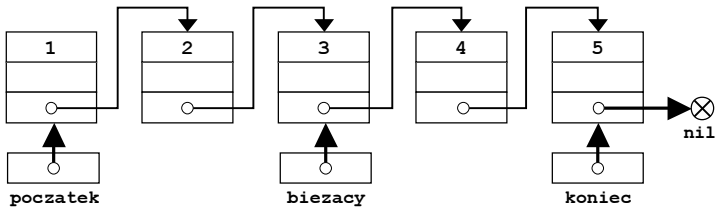


14

Listy liniowe - operacje

Niech dana będzie lista składająca się z elementów następującego typu:

```
TYPE element = record
                  klucz : integer;
                  dane : T0;
                  nast : ^element;
               end;
```

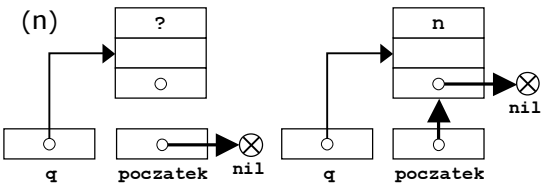


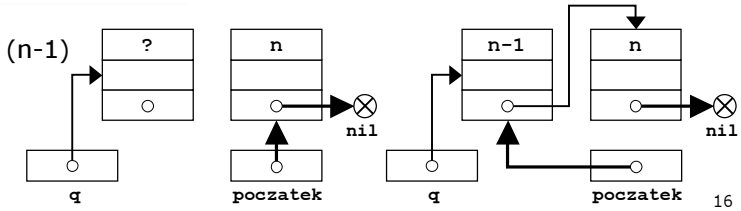
15

Tworzenie listy

Tworzenie listy przez dopisywanie n-elementów na jej początek (klucze: 1,2,3..n)

```
VAR q : ^element;
    {zmienna pomocnicza}

poczatek=nil;
WHILE n>0 DO
BEGIN
    new(q);
    q^.nast:=poczatek;
    poczatek:=q;
    poczatek^.klucz:=n;
    n:=n-1;
END;
```



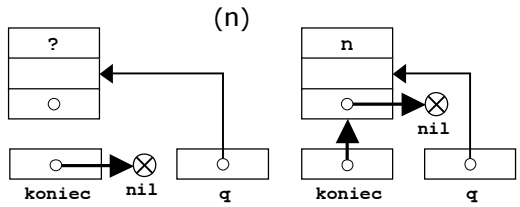


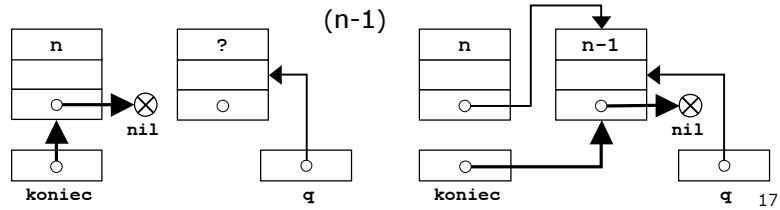
16

Tworzenie listy

Tworzenie listy przez dopisywanie n-elementów do jej końca (klucze: n..3,2,1)

```
VAR q : ^element;
    {zmienna pomocnicza}

koniec=nil;
WHILE n>0 DO
BEGIN
    new(q);
    q^.nast:=nil;
    IF koniec<>nil THEN
        koniec^.nast:=q;
    koniec:=q;
    koniec^.klucz:=n;
    n:=n-1;
END;
```





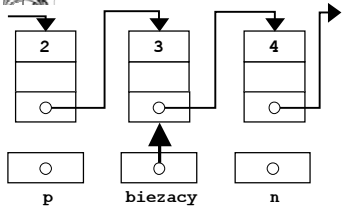
17

Dodawanie elementu do środka listy

(za elementem wskazywanym przez **`biezacy`**)

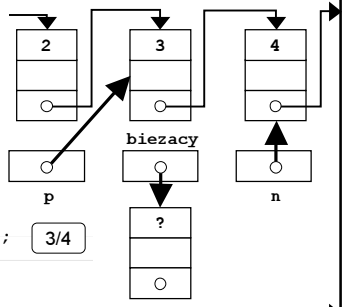
```
PROCEDURE DODAJZA(k:integer; W:T0)
VAR
    p, n: ^element;
BEGIN
```

1/4



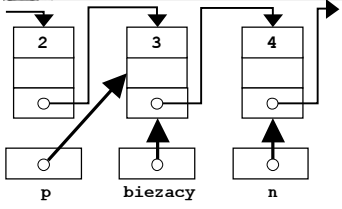

```
    new(biezacy);
```

3/4



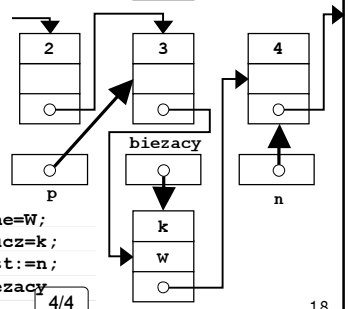

```
    p:=biezacy;
    n:=biezacy^.nast;
```

2/4




```
    biezacy^.dane=W;
    biezacy^.klucz=k;
    biezacy^.nast:=n;
    p^.nast:=biezacy
END
```

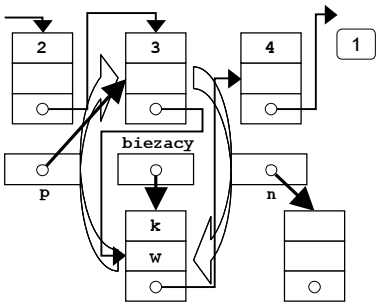
4/4

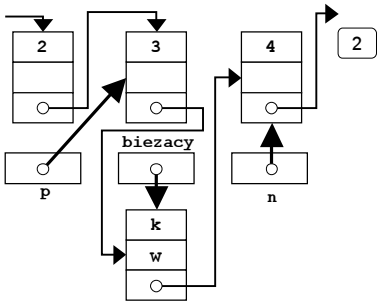


18

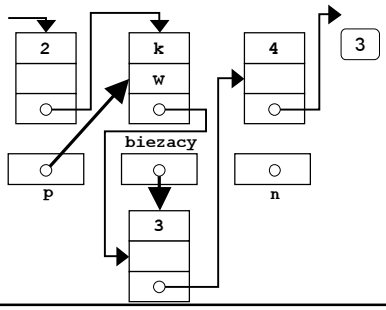
Dodawanie elementu do środka listy

(przed elementem wskazywanym przez **`biezacy`**)






```
{dodaj element za biezacym}
...
new(n);
n^.klucz:=p^.klucz;
n^.dane :=p^.dane;
p^.klucz:=biezacy^.klucz;
p^.dane :=biezacy^.dane;
biezacy^.klucz:=n^.klucz;
biezacy^.dane:=n^.dane;
dispose(n);
...
```



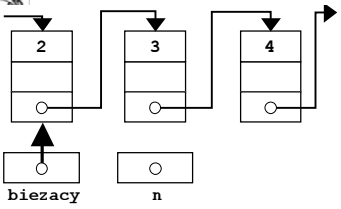
19

Usuwanie elementu ze środka listy

(za elementem wskazywanym przez **`biezacy`**)

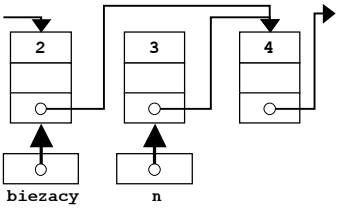
```
PROCEDURE USUNZA(var W:T0)
VAR
   n: ^element;
BEGIN
```

1/4



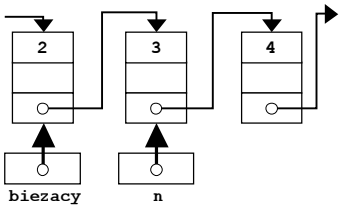

```
biezacy^.nast:=n^.nast;
```

3/4



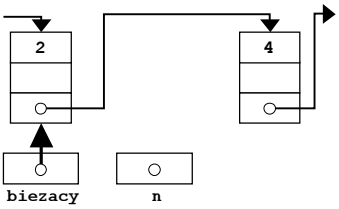

```
n:=biezacy^nast;
W:=n^.dane;
```

2/4



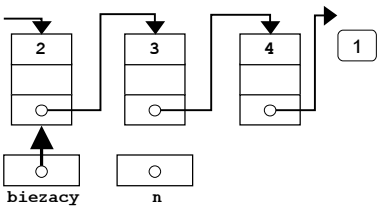

```
dispose(n)
END
```

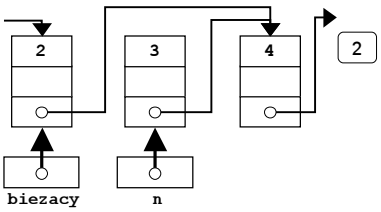
4/4



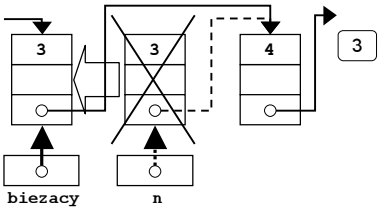
20

Usuwanie elementu ze środka listy
(elementu wskazywanego przez **`biezacy`**)





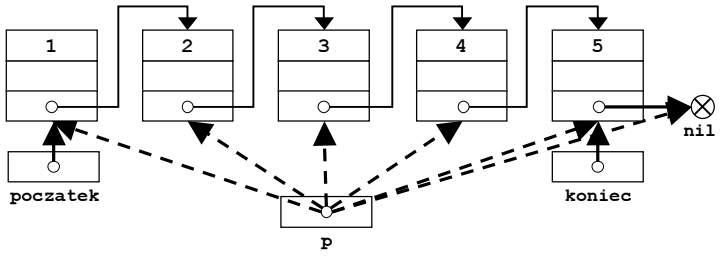

```
PROCEDURE USUN(var W:T0)
VAR
   n: ^element;
BEGIN
   n:=biezacy^nast;
   W:=biezacy^.dane;
   biezacy^.nast:=n^.nast
   biezacy^.klucz:=n^.klucz;
   biezacy^.dane :=n^.dane;
   dispose(n)
END
```



21

Przeglądanie elementów listy
(od początku do końca)

```
p:=poczatek;
WHILE p<>nil DO
BEGIN
   { operacja na elemencie p^ }
   p:=p^.nast;
END;
```



22

Wyszukiwanie elementu listy o danym kluczu

```
p:=poczatek;
WHILE (p<>nil) AND (p^.klucz<>x) DO
   p:=p^.nast;

IF p<>nil THEN
BEGIN
   { operacja na wyszukanym elemencie p^ };
END
```

podczas obliczania warunku, gdy **p=nil** nastąpi próba odwołania do nieistniejącego elementu **p^**, co może spowodować błędu w programie (np. odwołanie do nieistniejącego fragmentu pamięci)

Wersja poprawna:

```
p:=poczatek;
temp:=true;
WHILE (p<>nil) AND temp DO
   IF p^.klucz=x THEN
       temp:=false
   ELSE
       p:=p^.nast;

IF p<>nil THEN
BEGIN
   { operacja na wyszukanym elemencie p^ };
END
```

23

Przykład - lista jednokierunkowa skorowidz słów

Dany jest tekst, dla którego należy sporządzić listę częstości występowania słów. Słowa, po odczytaniu są zamieniane na unikalną wartość numeryczną, która stanowi klucz każdego elementu listy.

```
TYPE slowo =  record
                klucz : integer;
                licznik : integer;
                nast : ^slowo;
              end;
```

24

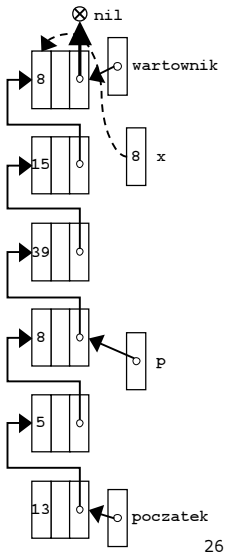
Skorowidz - program

```
PROGRAM skorowidz;

TYPE slowo =  record
                 klucz : integer;
                 licznik : integer;
                 nast : ^slowo;
              end;
VAR
   k: integer;
   poczatek: ^slowo;

PROCEDURE szukaj(x:integer;
                 var poczatek:^slowo);
VAR
   p:^slowo;
   temp:boolean;
BEGIN
   p:=poczatek;
   temp:=true;
   WHILE (p<>nil) AND temp DO
      IF p^.klucz=x THEN
            temp:=false
      ELSE p:=p^.nast;
   IF temp=true THEN
   {dodaj słowo na początku listy}
   BEGIN
      p:=poczatek; new(poczatek);
      poczatek^.klucz:=x;
      poczatek^.licznik:=1;
      poczatek^.nast:=p
   END
   ELSE
      p^.licznik:=p^.licznik + 1;
END;

PROCEDURE drukujliste(q:pslowo);
BEGIN
   WHILE q<>nil DO
   BEGIN
      writeln(q^.klucz, q^.licznik);
      q:=q^.nast;
   END
END;

BEGIN
   { na początku lista pusta }
   poczatek:=nil;
   { czytaj klucz kolejnego słowa}
   read(k);
   WHILE k<>0 DO
   BEGIN
      szukaj(k,poczatek);
      read(k);
   END;
   drukujliste(poczatek);
END.
```

Skorowidz – wyszukiwanie z wartownikiem

Procedurę szukaj można znacznie uprościć ustawiając tzw. **wartownika na ostatni element** listy, tj. wskaźnik wskazujący na ostatni element listy, co ułatwia tworzenie kryterium zatrzymania procedury szukania, jeśli brak na liście elementu o szukanym kluczu. Nowe elementy (o nieistniejącym kluczu) są tworzone na początku listy.

```
VAR
   wartownik:^slowo;
...
PROCEDURE szukaj(x:integer; var poczatek:^slowo);
VAR
   p:^slowo;
BEGIN
   p:=poczatek;
   wartownik^.klucz:=x;
   WHILE p^.klucz<>x DO
      p:=p^.nast;
   IF p<>wartownik THEN
      p^.licznik:=p^.licznik+1
   ELSE {dodaj slowo na poczatku listy}
   BEGIN
      p:=poczatek; new(poczatek);
      poczatek^.klucz:=x;
      poczatek^.licznik:=1;
      poczatek^.nast:=p
   END
END;
{ inicjacja listy w programie głównym }
   new(wartownik);
   poczatek:=wartownik
```



26

Lista uporządkowana

Tworzenie listy uporządkowanej: lista jest przeszukiwana tylko do momentu znalezienia elementu o większym lub równym kluczu i nowy element jest wstawiany tak aby zachować uporządkowanie.

Przyspiesza to przeszukiwanie, zwłaszcza gdy istnieje dużo elementów różnych kluczach, ale o małej lub wyrównanej częstości występowania.

27

Lista uporządkowana (skorowidz c.d.)

```
VAR
   wartownik:^slowo;
...
PROCEDURE szukaj(x:integer; var poczatek:^slowo);
VAR
   w1,w2,w3:^slowo; {pomocnicze wskaźniki}
BEGIN
   w2:=poczatek; w1:=w2^.nast;
   wartownik^.klucz:=x;
   WHILE w1^.klucz<x DO
   BEGIN
      w2:=w1; w1:=w2^.nast;
   END;
   IF (w1^.klucz=x) AND (w1<>wartownik) THEN
      w1^.licznik:= w1^.licznik +1
   ELSE
   BEGIN {wstaw nowy element pomiędzy w1 i w2}
      new(w3);
      w3^.klucz:=x;
      w3^.licznik:=1;
      w3^.nast:=w1;
      w2^.nast:=w3;
   END
END;
   { inicjacja listy w programie głównym }
   new(poczatek);
   new(wartownik);
   poczatek^.nast:=wartownik;
```

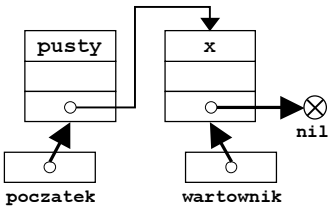
Przeszukiwanie listy wymaga teraz dwóch pomocniczych wskaźników w1, w2, ustawionych na sąsiednie elementy, tak aby możliwe było odwołanie się do elementu poprzedzającego znaleziony element o większym (lub równym) kluczu. Trzeci wskaźnik w3 używany jest to tworzenia nowego elementu listy, włączanego do jej środka.

metoda te wymaga utworzenia na początku listy pustego, pomocniczego elementu.

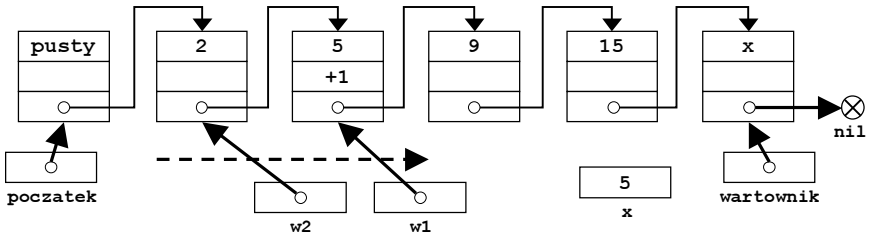
28

Lista uporządkowana (skorowidz) - interpretacja

Na początku tworzona jest lista pusta składająca się z elementu pomocniczego na początku oraz wartownika na końcu. Taka inicjacja lista umożliwia zawsze wstawianie każdego nowego elementu pomiędzy dwa już istniejące elementy listy, co upraszcza konstrukcję procedury **`szukaj()`**.



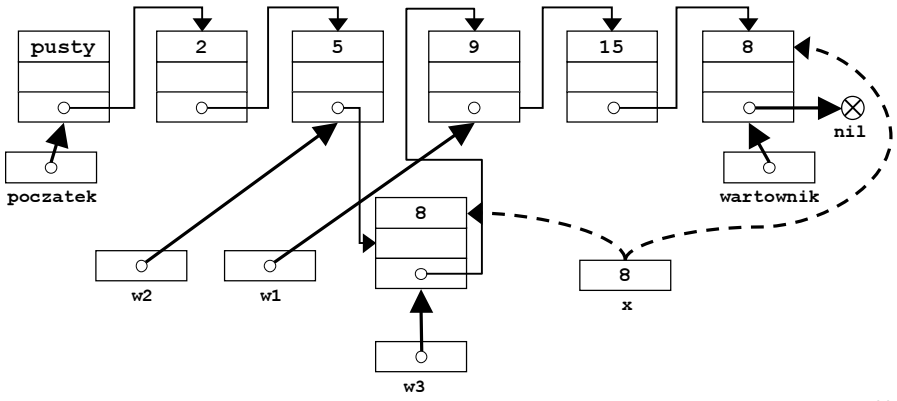
Do wyszukiwania elementu o kluczu x służą pomocnicze wskaźniki (polujące) `w1` i `w2`, które przesuwane są od początku listy, aż do wartownika.
W przypadku znalezienia elementu o danym kluczu `w1` wskazuje na dany element i zwiększany jest licznik danego elementu.



29

Lista uporządkowana (skorowidz) - interpretacja

W przypadku nie znalezienia takiego elementu, nowy element wstawiany jest pomiędzy wskaźniki **`w1`** i **`w2`**, które zatrzymują się po znalezieniu elementu o kluczu większym od **`x`**. Operacja wstawienia nowego elementu wymaga jeszcze jednego pomocniczego wskaźnika w3.



30

Lista reorganizowana

Tworzenie listy reorganizowanej: lista jest przeszukiwana do momentu znalezienia elementu o równym kluczu, licznik znalezionego elementu jest zwiększany, a następnie element jest „przenoszony” na początek listy. Nowe elementy (o nieistniejących w liście kluczach) są zawsze tworzone na początku listy.

Przyspiesza to przeszukiwanie, wówczas gdy elementy różnią się znacznie częstością występowania. W liście reorganizowanej, elementy częściej występujące znajdują się zawsze na początkowych miejscach listy.

31

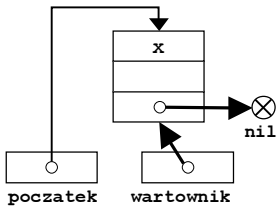
Lista reorganizowana (skorowidz c.d.)

Przeszukiwanie listy wymaga również dwóch pomocniczych wskaźników **w1**, **w2**.

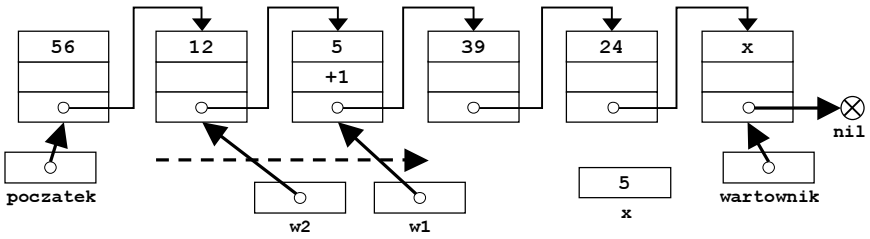
```
VAR
wartownik:^slowo;
...
PROCEDURE szukaj(x:integer;
          var poczatek:^slowo);
VAR
   w1,w2:^slowo;
   {pomocnicze wskaźniki}
BEGIN
   w1:=poczatek;
   wartownik^.klucz:=x;
   IF w1=wartownik THEN
   BEGIN{wstaw pierwszy element}
      new(poczatek);
      poczatek^.klucz:=x;
      poczatek^.licznik:=1;
      poczatek^.nast:=wartownik;
   END
   ELSE
      {szukany element pierwszy ?}
      IF (w1^.klucz=x) THEN
         w1^.licznik:=w1^.licznik+1
      ELSE
      BEGIN
         REPEAT {znajdź element}
            w2:=w1;
            w1:=w2^.nast;
         UNTIL w1^.klucz=x;
         IF w1=wartownik THEN
         {wstaw nowy element na początek}
         BEGIN
            w2:=poczatek;
            new(poczatek);
            poczatek^.klucz:=x;
            poczatek^.licznik:=1;
            poczatek^.nast:=w2;
         END
         ELSE
         BEGIN
            w1^.licznik:=w1^.licznik+1;
            {reorganizacja listy}
            w2^.nast:=w1^.nast;
            w1^.nast:=poczatek;
            poczatek:=w1;
         END
      END
END;
...
   {inicjacja listy w programie głównym}
   new(wartownik);
   poczatek:=wartownik;
```

Lista reorganizowana (skorowidz) - interpretacja

Na początku tworzona jest lista pusta składająca wartownika na końcu. Nowe elementy (o nieistniejącym kluczu) są zawsze dodawana na początek listy, bez zachowania uporządkowania kluczy.



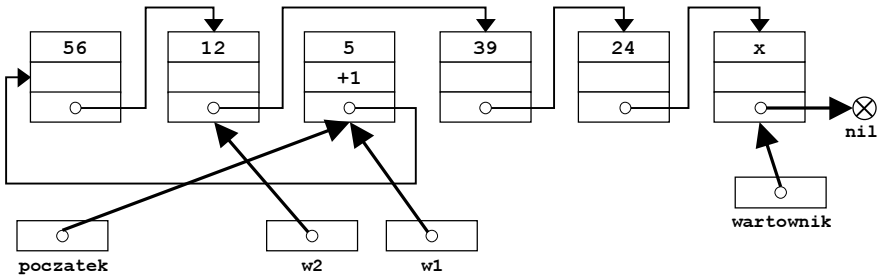
Do wyszukiwania elementu o kluczu x służą pomocnicze wskaźniki (polujące) **w1** i **w2**, które przesuwane są od początku listy, aż do wartownika.
W przypadku znalezienia elementu o danym kluczu **w1** wskazuje na dany element i zwiększany jest licznik danego elementu.



33

Lista reorganizowana (skorowidz) - interpretacja

Po znalezieniu elementu o kluczu **x** i zwiększeniu jego licznika, element jest przesuwany na początek listy za pomocą zmiany wskazań początku oraz wskaźników elementów.



34