```
        tmp=tmp.nastepny      # Idź dalej
    if znaleziono==True:
        return tmp, True      # Zwróć wynik poszukiwań (referencja do rekordu)
    else:
        return None, False    # Nic nie znaleziono!
```

Jej użycie w metodzie usun() może wyglądać tak:

```
def usun(self, pNazw):          # Odszukaj i usuń rekord pasujący do kryterium wyszukiwania
    res, znaleziono = self.szukaj(pNazw)
    if znaleziono==False:
        print(f"Brak [{pNazw}] na liście")
        return
    print(f"Usuwam [{pNazw}] z listy")

    if (res.poprzedni) != None and res.nastepny!=None:    # Środek
        (res.poprzedni).nastepny = res.nastepny
        (res.nastepny).poprzedni = (res.poprzedni).nastepny
        return

    if res == self.glowa:                                 # Usuwam z przodu
        self.glowa=res.nastepny
        (res.nastepny).poprzedni=None
    else:                                                 # Usuwam z tyłu
        (res.poprzedni).nastepny = None
        self.ogon = res.poprzedni
```

Zawartość listy dwukierunkowej może być wypisywana zarówno z przodu, jak i z tyłu:

```
    def wypiszWprzod(self, s):
        print(s)
        tmp=self.glowa
        while (tmp !=None):
            print(f"[{tmp.nazwisko}, {tmp.wiek}]", end=" ")
            tmp = tmp.nastepny
        print("")

    def wypiszWstecz(self,s ):
        print(s)
        tmp=self.ogon
        while (tmp != None):
            print(f"[{tmp.nazwisko}, {tmp.wiek}]", end=" ")
            tmp = tmp.poprzedni
        print("")
```

Dysponując takim zestawem, można już spróbować przetestować nasz kod (*Lista2KierMain.py*):

```
from MojeTypy import Lista2Kier as l
lista = l.Lista2Kier()
lista.wstaw("A", 12), lista.wstaw("B", 34), lista.wstaw("C", 34)
lista.wstaw("D", 55), lista.wstaw("E", 67);
lista.wypiszWprzod("Lista od przodu:")
lista.wypiszWstecz("Lista od tyłu:")
```

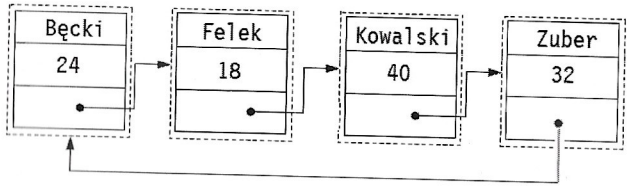

Wyniki:

```
Lista od przodu:
[A, 12] [B, 34] [C, 34] [D, 55] [E, 67]
Lista od tyłu:
[E, 67] [D, 55] [C, 34] [B, 34] [A, 12]
Usuwam [A] z listy
Lista od początku:
[B, 34] [C, 34] [D, 55] [E, 67]
```

Lista cykliczna

W niektórych algorytmach przydaje się jeszcze inny wariant listy, tzw. lista cykliczna, która jest zamknięta w pierścień: wskaźnik ostatniego elementu wskazuje pierwszy element. Przykład takiej listy znajdziesz na rysunku 7.17.

RYSUNEK 7.17. Lista cykliczna



Lista cykliczna pozwala na zapętlenie polegające na przypisaniu następnika ostatniego elementu do pierwszego elementu, co oznacza faktyczny brak głowy i ogona listy.

Można się umówić, że pewien element zostanie określony jako pierwszy i posłuży wyłącznie do wejścia w magiczny krąg wskaźników listy cyklicznej.

Przy okazji omawiania list cyklicznych często przywoływany jest problem-zagadka Józefa Flawiusza, żydowskiego historyka urodzonego w 37 roku n.e. w Jerozolimie. W trakcie powstania żydowskiego przeciwko Rzymianom dostał się on do niewoli i został wraz z grupą 41 żydowskich powstańców otoczony w jaskini. Otoczeni, woląc samobójstwo od niewoli, zdecydowali się utworzyć krąg i zabijać co trzecią osobę, aż nikt nie zostanie przy życiu. Podobno Flawiusz zdołał szybko wyliczyć, gdzie on i jego przyjaciel powinni stanąć w kręgu, aby uniknąć śmierci. (Ustawił się na 16. miejscu, dzięki czemu pozostał, jako przedostatnia osoba, przy życiu. Następnie, wraz z osobą z numerem 31., oddał się w ręce Rzymian i przeżył).

Pomijając wątek historyczny (którego prawdziwości nie chcę oceniać), mamy tu do czynienia z kombinatorycznym problemem eliminacji elementów zbioru. Można go łatwo zamodelować za pomocą listy cyklicznej, w której odliczamy i usuwamy kolejny element aż do wyczerpania się zbioru danych!

Jak zaimplementować listę cykliczną?

1. Poeksperymentuj, używając kodu pokazanego nieco wcześniej (plik *Lista2Kier.py*). Jak się łatwo zorientujesz, drobna modyfikacja dokładania