```
        self.dlugosc = 0
        return

    if self.glowa == biezacy:                    # Przypadek b) – usuwamy z przodu
        self.glowa = biezacy.nastepny            # Przestawiamy wskaźnik "glowa"
        poprzedni.nastepny = biezacy.nastepny    # Przestawiamy wskaźnik "ogon"
        return

    if self.ogon == biezacy:                     # Przypadek c) – usuwamy z tyłu
        self.ogon = poprzedni                    # Przestawiamy wskaźnik "ogon"
        poprzedni.nastepny = None                # Zaznaczamy nowy koniec listy
```

Ćwiczenie 7.3

Dokończ realizację ćwiczenia z listingu *BazaDanychMain.py* i przyjrzyj się realizacji metody `usunWybrany()` (klasa `Kartoteka`) oraz utwórz jej odpowiednik o nazwie `UsunIndeksy()`, tak dostosowując kod i typy danych, aby usunąć rekordy indeksowe wskazujące na adres rekordu danych zwrócony przez `usunWybrany()`. Uzyskasz wówczas komplet metod potrzebnych do pełnego usunięcia rekordu z naszej bazy danych wraz z towarzyszącymi metodami indeksującymi. (Moja implementacja była niekompletna w tym zakresie!)

„Tablicowa" implementacja list

W Pythonie nie ma już klasycznych tablic znanych np. z języka C++, ale nawet posiłkując się ich faktyczną realizacją z użyciem list, można pokusić się o interesujące realizacje algorytmiczne. Struktury listowe, tworzone od zera, wymagają również sprawnego zarządzania obiektami i referencjami, a nie wszyscy lubią taki styl rozwiązywania zagadnień. Na szczęście, podobnie zresztą jak i w życiu, to samo zadanie można często zrealizować na kilka sposób, o czym niejednokrotnie zapominamy.

Tak też jest z listami. Okazuje się, że istnieje kilka sposobów tablicowej implementacji list, niektóre z nich charakteryzują się nawet dość istotnymi zaletami, niemożliwymi do uzyskania w realizacji klasycznej (czyli tej, którą mieliśmy okazję poznać wcześniej).

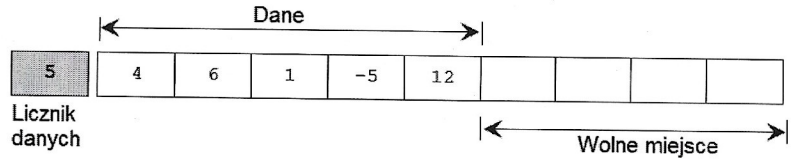
W podanych poniżej realizacjach przez chwilę „zapomnimy", że listy Pythona w nich użyte ciągle mogą *technicznie* powiększać swój rozmiar (metoda `append()` nie zostanie zablokowana!), nawet jeśli umówimy się, że deklarujemy tablice o stałym rozmiarze!

Klasyczna reprezentacja tablicowa

Jedną z najprostszych metod zamiany (umownej) tablicy na listę jest umówienie się co do sposobu interpretacji jej zawartości. Jeśli powiemy sobie głośno (i nie zapomnimy zbyt szybko o tym), że n-temu indeksowi tablicy będzie odpowiadać

Oprócz tego konieczne będzie wybranie jakiejś zmiennej do zapamiętywania aktualnej liczby elementów wstawionych wcześniej do listy.

Ideę tę ilustruje rysunek 7.12, gdzie możemy zobaczyć tablicową implementację listy pięcioelementowej złożonej z elementów: 4, 6, 1, -5 i 12:



RYSUNEK 7.12. Tablicowa implementacja listy

Programowa realizacja jest bardzo prosta — deklaracja przykładowej klasy nie powinna mieć żadnych niespodzianek dla czytelnika. Aby nasz przykład był nieco bardziej życiowy, zamiast listy liczb wprowadziłem listę rekordów osobowych pewnej niezbyt skomplikowanej klasy o nazwie `Fiszka` (fragment pliku *list-tab.py*):

```
class Fiszka:                    # Element składowy listy danych (rekord informacyjny)
    def __init__(self, pNazwisko="", pWiek=0):
        self.nazwisko = pNazwisko
        self.wiek = pWiek

class ListaTab:                  # Realizacja listy-tablicy elementów typu 'Fiszka'
    def __init__(self, MaxTab):
        self.tab= [ Fiszka() for x in range(MaxTab)] # Tworzy „statyczną" tablicę
                                                     # o rozmiarze MaxTab
        self.Licznik = 0         # Liczba wpisów na początku to 0
        self.Rozmiar=MaxTab      # Zapamiętajmy maksymalny rozmiar

    def wypisz(self,s ):
        print(s)
        for i in range(0, self.Licznik):
            print(f"[{self.tab[i].nazwisko}, {self.tab[i].wiek}]", end=" ")
        print()
```

Błyskawicznie omówmy wybrane funkcje usługowe klasy, tak aby pokazać różnicę w realizacji listy przy użyciu struktury tablicowej:

- brak referencji do następnego elementu, gdyż jest zawsze dostępny w kolejnej komórce;
- łatwe usuwanie obiektów z listy, nawet jeśli usuwany element znajduje się gdzieś „w środku" listy!

Przypuśćmy, że chcemy mieć możliwość usunięcia odszukanego elementu naszej listy. Po zbadaniu sensu takiej operacji (element musi istnieć!) wystarczy przesunąć zawartość tablicy o jeden w lewo od pozycji, na której został odnaleziony rekord danych. Podczas przesuwania elementów w tablicy jest on nadpisywany przez swojego sąsiada.