

Dynamiczne struktury danych

- Statyczne: tablica (można ją nieco zdynamizować – bufor kołowy lub rozszerzanie tablicy)

Listy (linked lists)

- Kolekcja elementów (węzłów) połączonych wskaźnikami
- Jednokierunkowe / dwukierunkowe
- Posortowane / nieposortowane
- Z wartownikiem (sentinel) / bez wartownika
- Operacje:
 - Dokładanie elementu
 - Usuwanie elementu
 - Wyszukiwanie

```

TYPE
    List = POINTER TO ListDesc;
    ListDesc = RECORD
        key: ARRAY 64 OF CHAR;
        next: List;
    END;

VAR root: List;

PROCEDURE InitList;
BEGIN root := NIL
END InitList;

PROCEDURE Insert (key: ARRAY OF CHAR; unique:
BOOLEAN);
    VAR p, q: List;
BEGIN
    IF (root = NIL) OR (root.key > key) THEN
        NEW (q); COPY (key, q.key); q.next :=
root; root := q
    ELSE
        p := root; q := root;      (* -> to test
uniqueness with root.key *)
        WHILE (q # NIL) & (q.key < key) DO p :=
q; q := q.next END;
        IF unique & (q # NIL) & (q.key = key)
THEN HALT (99) END;
        NEW (q); COPY (key, q.key); q.next :=
p.next; p.next := q
    END
END Insert;

```

```

PROCEDURE Delete (key: ARRAY OF CHAR);
    VAR p, q: List;
BEGIN
    IF (root = NIL) OR (root.key > key) THEN
RETURN
    ELSIF root.key = key THEN
        WHILE (root # NIL) & (root.key = key) DO
root := root.next END
    ELSE
        p := root; q := root.next;
        WHILE (q # NIL) & (q.key < key) DO p :=
q; q := q.next END;
        WHILE (p.next # NIL) & (p.next.key = key)
DO
            p.next := p.next.next
        END
    END
END Delete;

PROCEDURE Find (key: ARRAY OF CHAR): List;
    VAR p: List;
BEGIN
    p := root;
    WHILE (p # NIL) & (p.key < key) DO p := p.next
END;
    IF (p = NIL) OR (p.key > key) THEN RETURN
NIL
    ELSE RETURN p
    END
END Find;

```

***Uporzędkowana lista bez wartownika
(źródło <http://www.oberon.ethz.ch>)***

```

TYPE
  List = POINTER TO ListDesc;
  ListDesc = RECORD
    key: ARRAY 64 OF CHAR;
    next: List;
  END;

VAR root: List;

PROCEDURE InitList;
BEGIN NEW (root); root.next := NIL
END InitList;

PROCEDURE Insert (key: ARRAY OF CHAR; unique:
BOOLEAN);
  VAR p, q: List;
BEGIN
  p := root;
  WHILE (p.next # NIL) & (p.next.key < key) DO p
:= p.next END;
  IF unique & (p.next # NIL) & (p.next.key = key)
THEN HALT (99) END;
  NEW (q); COPY (key, q.key); q.next := p.next;
p.next := q
END Insert;

```

```

PROCEDURE Delete (key: ARRAY OF CHAR);
  VAR p: List;
BEGIN
  p := root;
  WHILE (p.next # NIL) & (p.next.key < key) DO p
:= p.next END;
  IF (p.next # NIL) & (p.next.key = key) THEN
    p.next := p.next.next
  END
END Delete;

PROCEDURE Find (key: ARRAY OF CHAR): List;
  VAR p: List;
BEGIN
  p := root.next;
  WHILE (p # NIL) & (p.key < key) DO p := p.next
END;
  IF (p = NIL) OR (p.key > key) THEN RETURN
NIL
  ELSE RETURN p
  END
END Find;

```

***Uporządkowana lista z wartownikiem
(źródło <http://www.oberon.ethz.ch>)***

Stos (LIFO)

- Operacje:
 - Push – dodanie elementu
 - Pop – pobranie elementu
 - Peek – pobranie kopii elementu, bez usuwania go ze stosu

Kolejka FIFO

- Operacje:
 - Push – dodanie elementu
 - Pop – pobranie elementu
 - Peek – pobranie kopii elementu, bez usuwania go z kolejki

Bufor kołowy

- Tablica ze znacznikami początku i końca
- Odpowiada kolejce FIFO