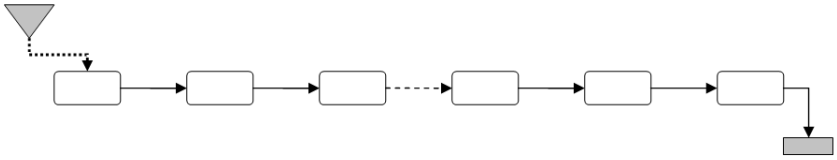
ASD - ćwiczenia VI

Liniowe struktury danych

- lista prosta
  - schemat graficzny struktury danych:

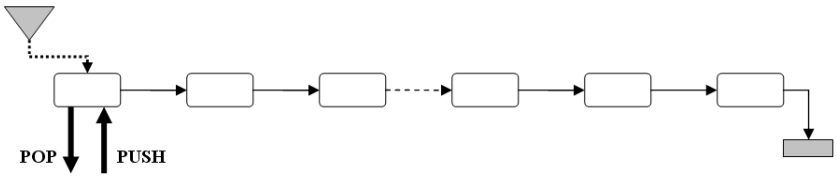


  - definicja wskaźnikowa struktury typu węzeł listy prostej w pseudokodzie

```
typedef struct ListNode List;

struct ListNode {
  element elem;
  struct ListNode next;
};
```

- implementacja **stosu** na liście prostej (lista LIFO – „last in first out“)
  - schemat graficzny struktury danych:



  - definicja wskaźnikowa struktury typu węzeł stosu w pseudokodzie

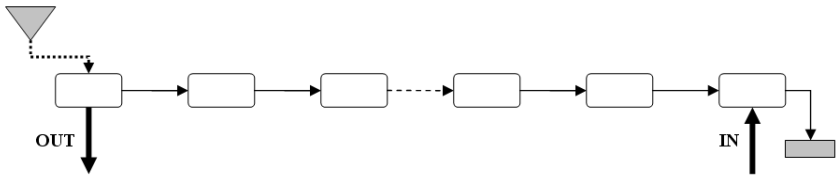
```
typedef struct StackNode Stack;

struct StackNode {
  element elem;
  struct StackNode prev;
};
```

  - operacje stosowe:
    - $EMPTY : \mathcal{S} \to \{TRUE, FALSE\}$, sprawdzenie czy struktura jest pusta, złożoność operacji $O(1)$,
    - $TOP : \mathcal{S} \to E$, „obejrzenie“ ostatnio wstawionego elementu struktury, złożoność operacji $O(1)$,
    - $PUSH : \mathcal{S} \times E \to \mathcal{S}$, wstawienie elementu „na początek“ struktury, złożoność operacji $O(1)$,
    - $POP : \mathcal{S} \to \mathcal{S}$, usunięcie ostatnio wstawianego elementu ze struktury, złożoność operacji $O(1)$,

gdzie $\mathcal{S}$ jest przestrzenią stosów, E zbiorem etykiet węzłów stosu,

- implementacja **kolejki** w liście prostej (lista FIFO – „first in first out“)
  - schemat graficzny struktury danych:



  - definicja wskaźnikowa struktury typu węzeł kolejki w pseudokodzie

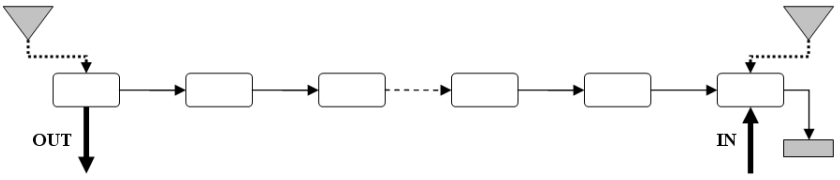
```
typedef struct QueueNode Queue;

struct QueueNode {
  element elem;
  struct QueueNode next;
};
```

  - operacje kolejkowe:
    - $EMPTY \ : \ \mathcal{Q} \rightarrow \{TURE, FALSE\}$, sprawdzenie czy struktura jest pusta, złożoność operacji $O\left(1\right)$,
    - $FIRST \ : \ \mathcal{Q} \rightarrow E$, „obejrzenie“ najwcześniej wstawionego elementu struktury, złożoność operacji $O\left(1\right)$,
    - $IN \ : \ \mathcal{Q} \times E \rightarrow \mathcal{Q}$, wstawienie elementu „na koniec“ struktury, złożoność operacji $O\left(n\right)$,
    - $OUT \ : \ \mathcal{Q} \rightarrow \mathcal{Q}$, usunięcie najwcześniej wstawionego elementu ze struktury, złożoność operacji $O\left(1\right)$,

  gdzie $\mathcal{Q}$ jest przestrzenią kolejek , E zbiorem etykiet węzłów kolejki,

- implementacja **kolejki** w liście prostej z dwoma dowiązaniami (lista FIFO – „first in first out“)
  - schemat graficzny struktury danych:



  - definicja wskaźnikowa struktury typu węzeł kolejki w pseudokodzie

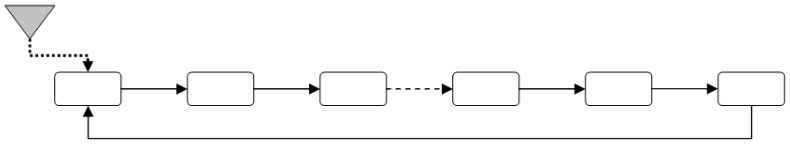
```
typedef struct QueueNode Queue;

struct QueueNode {
  element elem;
  struct QueueNode next;
};
```

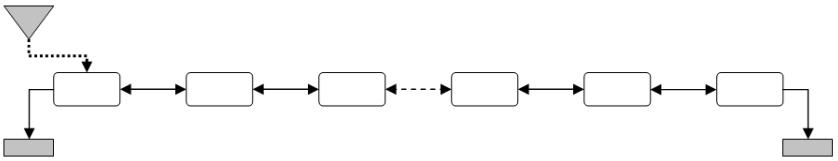
- operacje kolejkowe:
  - $EMPTY : \mathcal{Q} \to \{TURE, FALSE\}$, sprawdzenie czy struktura jest pusta, złożoność operacji $O(1)$,
  - $FIRST : \mathcal{Q} \to E$, „obejrzenie“ najwcześniej wstawionego elementu struktury, złożoność operacji $O(1)$,
  - $IN : \mathcal{Q} \times E \to \mathcal{Q}$, wstawienie elementu „na koniec“ struktury, złożoność operacji $O(1)$,
  - $OUT : \mathcal{Q} \to \mathcal{Q}$, usunięcie najwcześniej wstawionego elementu ze struktury, złożoność operacji $O(1)$,

  gdzie $\mathcal{Q}$ jest przestrzenią kolejek , E zbiorem etykiet węzłów kolejki,

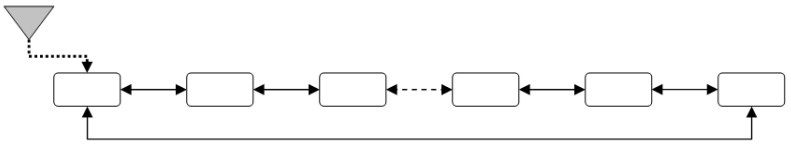
- lista cykliczna



- lista dwukierunkowa



- lista dwukierunkowa cykliczna



Zadania

1. Czy za pomocą jedynie pojedynczej struktury typu FIFO i stałej liczby zmiennych pomocniczych, można symulować strukturę typu LIFO? Jeżeli tak to jaki jest koszt takiej symulacji wyrażony liczbą niezbędnych operacji IN, OUT, $FIRST$, Q_EMPTY? Zakładamy, że dysponujemy funkcją $SIZE : \mathcal{Q} \to \mathbb{N}$, której rezultatem jest liczba elementów rozważanej struktury typu FIFO.

2. Czy za pomocą jedynie pojedynczej struktury typu LIFO i stałej liczby zmiennych pomocniczych, można symulować strukturę typu FIFO? Jeżeli tak to jaki jest koszt takiej symulacji wyrażony liczbą niezbędnych operacji $PUSH$, POP, TOM, S_EMPTY? Zakładamy, że dysponujemy funkcją $SIZE : \mathcal{S} \to \mathbb{N}$, której rezultatem jest liczba elementów rozważanej struktury typu LIFO.

3. Zaproponuj możliwie prostą strukturę danych typu $\mathcal{S}$ nad zbiorem liniowo uporządkowanym E, umożliwiającą wykonanie w czasie $O(1)$ następujących operacji:

(a) $EMPTY : \mathcal{S} \to \{TURE, FALSE\}$, sprawdzenie czy struktura jest pusta,

(b) $PUSH : \mathcal{S} \times E \to S$, wstawienie elementu do struktury,

(c) $POP : \mathcal{S} \to \mathcal{S}$, usunięcie ostatnio wstawianego elementu ze struktury,

(d) $FINDMIN : \mathcal{S} \to E$, wyznaczenie minimalnego elementu z przestrzeni E, aktualnie przechowywanego w strukturze,

(e) $FINDMAX : \mathcal{S} \to E$, wyznaczenie maksymalnego elementu z przestrzeni E, aktualnie przechowywanego w strukturze.

4. Zaproponuj możliwie prostą listową strukturę danych typu $\mathcal{L}$ nad zbiorem liniowo uporządkowanym E, umożliwiającą wykonanie w czasie $O(1)$ następujących operacji:

(a) $IN : \mathcal{L} \times E \to \mathcal{L}$, wstawienie elementu do struktury,

(b) $OUT : \mathcal{L} \to \mathcal{L}$, usunięcie „najstarszego“ względem kolejności wstawiania elementu struktury,

(c) $INVERT : \mathcal{L} \to \mathcal{L}$, zmiana kolejności wstawienie elementów (tj. „najstarsze“ stają się „najmłodszymi“),

(d) $MERGE : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$, wstawienie do pierwszej struktury (za elementem „najstarszym“) elementów z drugiej struktury w kolejności od „najstarszego“ do „najmłodszego“.

5. Gra „koło graniaste“ - $n > 1$ dzieci poindeksowanych liczbami $1, 2, \ldots, n$ stoi w kole i recytuje słowa m wyrazowej wyliczanki zaczynając od dziecka z indeksem i. Dziecko, które wypowie ostatnie słowo, np. dziecko k-te, odchodzi z koła i zabawa rozpoczyna się od dziecka z indeksem $k + 1$. Gra kończy się wtedy, kiedy w kole pozostanie tylko jedno dziecko.

(a) Zadeklaruj typ danych `Str`, którego zmienne pozwolą efektywnie oddać reguły gry w koło graniaste.

(b) Zaprojektuj funkcję

`int WINNER(int n, int m, int i)`,

która bazującą na strukturze typu `Str`, wyznaczy indeks zwycięzcy gry.

(c) Oszacuj złożoność czasową rozwiązania względem liczby operacji na strukturze typu `Str`.

6. Niech E^* będzie dowolnym n-znakowym poprawnym wyrażeniem algebraicznym (tablicą znaków) nad alfabetem

$$\Sigma = \{x, y, z, +, -, \cdot, /, (,)\},$$

nadawanym przez nadajnik $\mathcal{N}$ w notacji infiksowej. Załóżmy, że na skutek zakłóceń transmisji danych formuła E^* zostaje zniekształcona do postaci E, którą odbiera odbiornik $\mathcal{O}$. Przyjmijmy, że deformacja może dotyczyć jedynie zmiany znaków nawiasów z otwierających na zamykające bądź zamykających na otwierające. Zaproponuj funkcję

`bool VERIFY(char E[], int n)`,

weryfikującą w czasie $O(n)$, czy wyrażenie E jest poprawne w sensie domykania nawiasów, np.:

- $E = ((x + y) \cdot (z))$ – wyrażenie poprawnie nawiasowane,
- $E = ((x + y) \cdot)z))$ – wyrażenie niepoprawnie nawiasowane,
- $E = ((x + y(\cdot (z))$ – wyrażenie niepoprawnie nawiasowane.