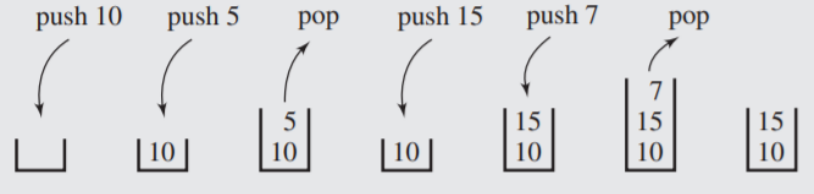
ALGORYTMY I STRUKTURY DANYCH

STRUKTURY DANYCH:
STOS, KOLEJKA

Abstrakcyjny typ danych jest to zbiór operacji określających, *co* jest robione, lecz nie *jak*.

Stos

Stos jest *abstrakcyjnym typem danych*, w którym przepływ elementów przebiega na zasadzie LIFO (ang. *Last In – First Out*). Łatwo zobrazować go na podstawie stosu książek – mamy dostęp jedynie do ostatniej książki, odłożonej na wierzch.



top() – zwraca wartość elementu na szczycie stosu;
empty() – zwraca *true*, jeśli stos jest pusty; *false* – w przeciwnym przypadku;
size() – zwraca liczbę elementów umieszczonych na stosie;
push(e) – umieszcza element *e* na szczycie stosu;
pop() – zdjęcie(usuniecie) *istniejącego* elementu ze szczytu stosu; funkcja nic nie zwraca.

Struktura danych	Operacja	Złożoność
Stos	`push()`	O(1)
	`pop()`	O(1)
	`empty()`	O(1)

Zadanie. Napisz implementację *stosu* opartego na liście w Pythonie, definiując powyższe funkcje. (*Plik do wykorzystania*: *zadanie.py*)

Zadanie 1. Napisz program wykorzystujący stos, który sprawdza, czy nawiasy w danym wyrażeniu są prawidłowo zagnieżdżone. Na przykład wyrażenie "(()()(()))" jest prawidłowe, a ")(" oraz "(()" są nieprawidłowe.

Zadanie 2. *Napisz program czytający wyrażenie arytmetyczne w notacji postfiksowej (*Odwrotna Notacja Polska*) i obliczający jego wartość z wykorzystaniem stosu.

ONP (ang. RPN - Reverse Polish Notation) jest sposobem zapisu wyrażeń arytmetycznych bez stosowania nawiasów, w którym symbol operacji występuje po argumentach. Poniżej podajemy przykłady wyrażeń w ONP:

Notacja normalna	ONP
2 + 2 =	2 2 + =
3 + 2 * 5 =	3 2 5 * + =
2 * (5 + 2) =	2 5 2 + * =
(7 + 3) * (5 - 2) ^ 2 =	7 3 + 5 2 - 2 ^ * =
4 / (3 - 1) ^ (2 * 3) =	4 3 1 - 2 3 * ^ / =

Algorytm obliczania wartości wyrażenia ONP wykorzystuje stos do składowania wyników pośrednich. Zasada pracy tego algorytmu jest bardzo prosta:

> Z wejścia odczytujemy kolejne elementy wyrażenia. Jeśli element jest liczbą, zapisujemy ją na stosie. Jeśli element jest operatorem, ze stosu zdejmujemy dwie liczby, wykonujemy nad nimi operację określoną przez odczytany element, wynik operacji umieszczamy z powrotem na stosie. Jeśli element jest końcowym znakiem '=', to na wyjście przesyłamy liczbę ze szczytu stosu i kończymy. Inaczej kontynuujemy odczyt i przetwarzanie kolejnych elementów.

Przykładowo obliczmy tą metodą wartość wyrażenia **7 3 + 5 2 - 2 ^ * =**

we	stos	wy
7	7	
3	7 3	
+	10	
5	10 5	
2	10 5 2	
-	10 3	
2	10 3 2	
^	10 9	
*	90	
=		90

Kolejka

Kolejka jest taką strukturą danych, w której przepływ elementów przebiega na zasadzie FIFO (ang. *First In – First Out*). Łatwo można to sobie wyobrazić, korzystając z przykładu zwykłej kolejki w sklepie: osoba, która pierwsza stanęła w kolejce, jako pierwsza zapłaci za swoje zakupy. W kolejce mamy dostęp jedynie do pierwszego elementu.

front() – zwraca wartość pierwszego elementu w kolejce;
back() – zwraca wartość ostatniego elementu w kolejce;
empty() – zwraca *true*, jeśli kolejka jest pusta; *false* – w przeciwnym przypadku;
size() – zwraca ilość elementów umieszczonych w kolejce;
push(e) – umieszczenie nowego elementu *e* na końcu kolejki;
pop() – usunięcie istniejącego elementu z początku kolejki.

Struktura danych	Operacja	Złożoność
Kolejka	`push()`	O(1)
	`pop()`	O(1)
	`empty()`	O(1)

Zadanie. Napisz implementację *kolejki* opartej na liście w Pythonie, definiując powyższe funkcje. Następnie wypełnij kolejkę wartościami 10, 20,..., 100. Wypisz na ekranie pierwszy i ostatni element kolejki oraz liczbę elementów znajdujących się w kolejce. Następnie dokonaj przetworzenia kolejki według zasady FIFO.