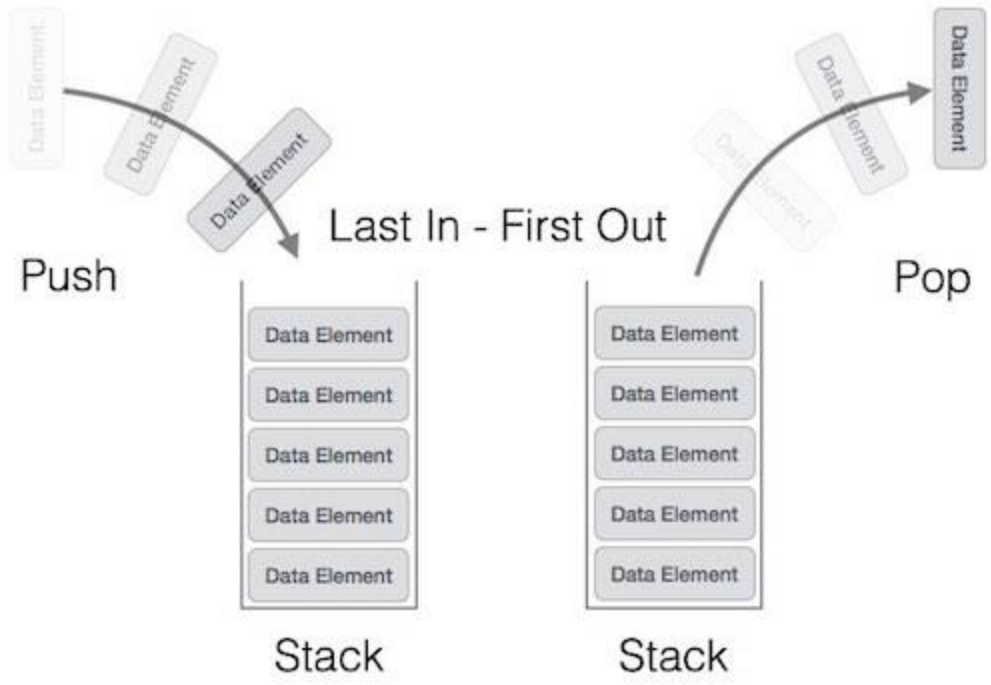
Chapter 2

Stack

Basic Concepts

- A stack an ADT. It is an ordered collection of items into which new items may be inserted and from which items may be deleted at one end called the ***top*** of the stack.
- Stack is also called **last-in-first-out (LIFO)** list because the element which is inserted last, is removed first.



Stack Operations

- Two primary operations are **push** and **pop**. In stack terminology, **push** is the insertion operation and **pop** is the removal operation. Because of **push** operation, stack is sometimes called a **pushdown list**. To use a stack efficiently, we also need some additional operations.
  - **peek** – get top data element of the stack, without removing it.
  - **isfull** – check if stack is full.
  - **isempty** – check if stack is empty.
- The result of an illegal attempt to **pop** or **peek** an item from an empty stack is called **underflow**. Array implementation may introduce **overflow** if the stack grows larger than the size of the array.

Example: Bracket Matching

- A phrase is **well-bracketed** if:
  - for every left bracket, there is a later matching right bracket
  - for every right bracket, there is an earlier matching left bracket
  - the subphrase between a pair of matching brackets is itself well-bracketed.
- Examples and counter-examples (math expressions):

$s \times (s - a) \times (s - b) \times (s - c)$ — well-bracketed

$(-b + \sqrt{[b^2 - 4ac]}) / 2a$ — well-bracketed

$s \times (s - a) \times (s - b \times (s - c)$ — ill-bracketed

$s \times (s - a) \times s - b) \times (s - c)$ — ill-bracketed

$(-b + \sqrt{[b^2 - 4ac)}] / 2a$ — ill-bracketed

Example: Bracket Matching

- **Bracket matching algorithm:**

  To test whether *phrase* is well-bracketed:

  1. Make *bracket-stack* empty.
  2. For each symbol *sym* in *phrase* (scanning from left to right), repeat:
     - 2.1. If *sym* is a left bracket:
       - 2.1.1. Add *sym* to the top of *bracket-stack*.
     - 2.2. If *sym* is a right bracket:
       - 2.2.1. If *bracket-stack* is empty, terminate with false.
       - 2.2.2. Remove a bracket from the top of *bracket-stack* into *left*.
       - 2.2.3. If *left* and *sym* are not matched brackets, terminate with false.
  3. Terminate with true if *bracket-stack* is empty, or false otherwise.

C-implementation using Array

```
#include <stdio.h>
#define MAXSIZE 10
int stack[MAXSIZE];
int top = -1;
int isempty()
{
        if(top == -1)
                return 1;
        else
                return 0;
}
int isfull()
{
        if(top == MAXSIZE - 1)
                return 1;
        else
                return 0;
}
```

```
int peek()
{
        if(isempty())
                printf("Stack is empty.\n");
        else
                return stack[top];
}

void push(int data)
{
        if(isfull())
                printf("Stack is full.\n");
        else

                stack[++top] = data;
}
```

C-implementation using Array

```
int pop()
{
        if(isempty())
                printf("Stack is empty.\n");
        else
                return stack[top--];
}
int main()
{
        push(1);
        push(12);
        push(15);
        push(20);
        printf("Element at top of the stack: %d\n", peek());
        printf("Popped element: %d\n", pop());
        printf("Popped element: %d\n", pop());
        printf("Element at top of the stack: %d\n", peek());
}
```