Stacks

Stacks

A stack is a linear data structure that store data in an order known as the Last In First Out (LIFO) order. This property is helpful in certain programming cases where the data needs to be ordered.

Stack

Stack

Stacks can be visualised like a stack of plates on a table. Only the top plate is accessible by the user at any given instant. The other plates are hidden and are not accessible by the user. The last plate that is kept on the stack is retrieved first.

Stack of dinner plates

A stack of dinner plates

Santeri Viinamäki [CC BY-SA 4.0]

Operations in a Stack

The two primary operations in a stack are the push and the pop operations:

Push Operation

This is used to add (or push) an element to the stack. The element always gets added to the top of the current stack items.

Pop Operation

This is used to remove (or pop) an element from the stack. The element always gets popped off from the top of the stack.

Peek Operation

The peek operation is used to return the first element of the stack without removing the element. It is a variation of the pop operation.

Overflow and Underflow Conditions

A stack may have a limited space depending on the implementation. We must implement check conditions to see if we are not adding or deleting elements more than it can maximum support.

The underflow condition checks if there exists any item before popping from the stack. An empty one cannot be popped further.

```
if (top == -1) {
    underflow condition
}
```

The overflow condition checks if the stack is full (or more memory is available) before pushing any element. This prevents any error if more space cannot be allocated for the next item.

```
if (top == sizeOfStack) {
    overflow condition
}
```

About the top pointer

To efficiently add or remove data, a special pointer is used which keeps track of the last element inserted in the structure. This pointer updates continuously and keeps a check on the overflow and underflow conditions.

Creating a stack

A stack can be created using both an array or through a linked list. For simplicity, we will create one with an array.

First, we create a one-dimensional array with fixed size (int stack[SIZE]). The SIZE value could be defined using a preprocessor.

Define a integer variable top and initialize with '-1' (ntitop = -1).

```
#define SIZE 10
int stack[SIZE];
int top = -1;
```

Pushing to the stack

Steps

Check whether stack is FULL. (top == SIZE-1)

If it is FULL, then terminate the function and throw an error.

If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (tack[top] = value).

```
void push(int value) {
```

```
if(top == SIZE-1)
printf("\nOverflow. Stack is Full");
else{
top++;
stack[top] = value;
printf("\nInsertion was successful");
    }
}
```

Popping from the stack

Steps

Check whether stack is EMPTY. (top == -1)

If it is EMPTY, then terminate the function and throw an error.

If it is NOT EMPTY, then delete stack[top] and decrement top value by one (dp--).

```
void pop() {
if(top == -1)
printf("\nUnderflow. Stack is empty");
else{
printf("\nDeleted  %d", stack[top]);
```

```
top--;
    }
}
```

Accessing the top element (Peeking)

Steps

Check whether stack is EMPTY (top == -1).

If it is EMPTY, then terminate the function and throw an error.

If it is NOT EMPTY, then return stack[top].

```
void peek() {
if(top == -1)
{
printf("\n The stack is empty");
break;
    }
else
printf("%d", stack[top]);
}
```

Stack Complexity

Access

An element in a stack can only be accessed by continuously removing the front element until the required element is found. This means that the time complexity is O(n).

Search

Similar to accessing an element, searching an element will involve continuously popping an element until the required element is found. The time complexity is hence O(n).

Insertion

Inserting an element is only possible at the top of the stack. There is no interaction needed with the rest of the elements. It is hence an O(1) operation.

Deletion

Similar to insertion, deleting an element is only possible from the top of the stack. There is no interaction needed with the rest of the elements. It is hence an O(1) operation.

Space Required

A stack only takes the space used to store the elements of the data type specified. This means that for storing n elements the space required is O(n).

Applications of Stacks in Programming

UNDO functionality in text editors: Every change in the document is added to stack and upon a UNDO request, the last change is referred by popping it.

Parentheses checker: The ordered manner of the stack could be used for checking the proper closing of parentheses. Every opening parentheses is pushed on to the stack and for every correct closing parentheses, it is popped off. Irregularities can then be detected if they mismatch.

Expression parsing: Using stacks can help evaluate expressions faster using postfix or prefix notation.