data_tructures(&algorithms, lecture06)

Doan Trung Tung, PhD – University of Greenwich (Vietnam)

Plan

Stack ADT

Introduction, stack operation, stack application

01

Array Implementation

How to implement stack with Array

02

Dynamic Array Implementation

How to implement stack with Dynamic Array

03

Linked List Implementation

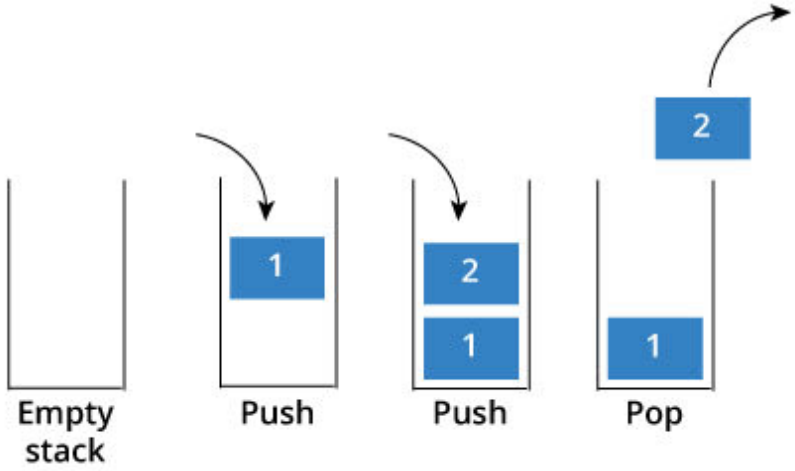
How to implement stack with Linked List

Stack ADT

Introduce Stack data structure, some operations

What Is A Stack

- A linear data structure that can be accessed only at one of its ends for storing and retrieving data
- A stack is a Last In, First Out (LIFO) data structure



[4]

Operations On Stack

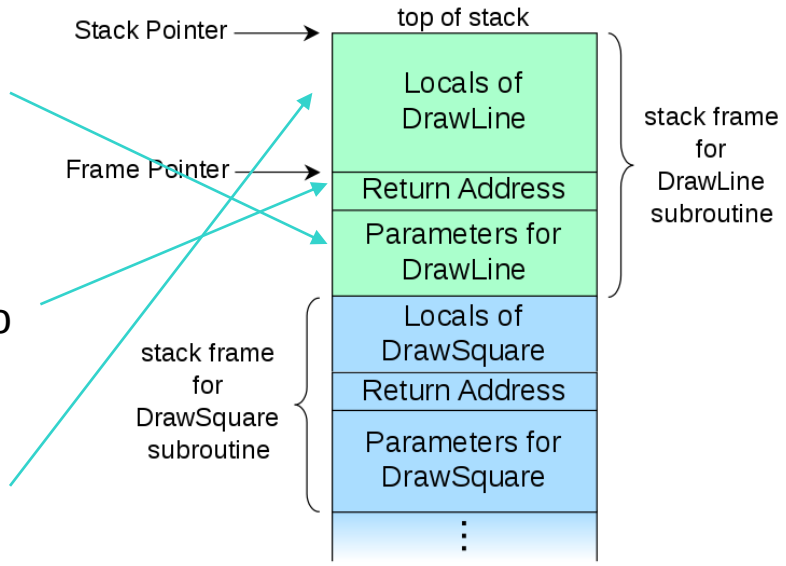
- clear: clear the stack
- is empty: check if a stack is empty
- push: put element on the top of the stack
- pop: take the element on the top out of the stack
- top: get top element without removing it from the stack
- size: get size of the stack

Applications Of Stack

- Any sort of nested checking
- Evaluating arithmetic expressions (and other sorts of expression)
- Implementing function or method calls
- Keeping track of previous choices (backtracking)
- Undo sequence of activities
- Auxiliary data structure for algorithms
- Component of other data structure

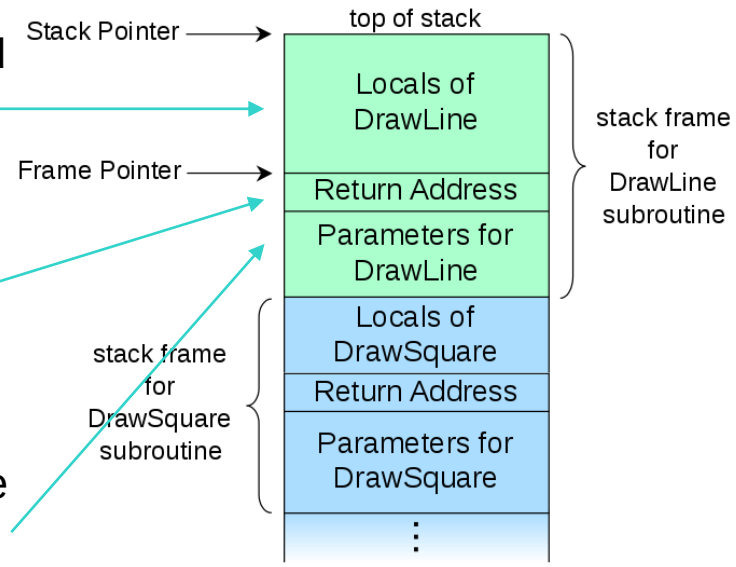
Example: Function Calls

- When a function is called:
  - Any parameters that the function is expecting are pushed onto the stack frame in reverse order
  - The return address of the caller function is pushed onto the stack.
  - Now the called function is in control. Any local variables defined within the called function will be pushed onto the stack as well.



Example: Function Calls

- When a function is returned:
  - Any local variables in the called function are popped off the stack.
  - The called function returns control to the calling function via the return address of the calling function that we pushed onto the stack.
  - The calling function is in charge of cleaning up the rest of the stack — the parameters that the called function was expecting.



Array Implementation

How to implement stack by array

Stack Implementation: Static Array

- Stack size is fixed
- Need global variables

```
void push(const int n)
{
    if (!is_full()) stack[++top] = n;
    else printf("Stack overflow!\n");
}
int pop()
{
    if (!is_empty()) return stack[top--];
    else exit(1);
}
```

```
#define STACK_SIZE 100

extern int stack[];
extern int top;

void clear_stack(void);
int is_empty(void);
int is_full(void);
void push(const int n);
int pop(void);
int size(void);
```

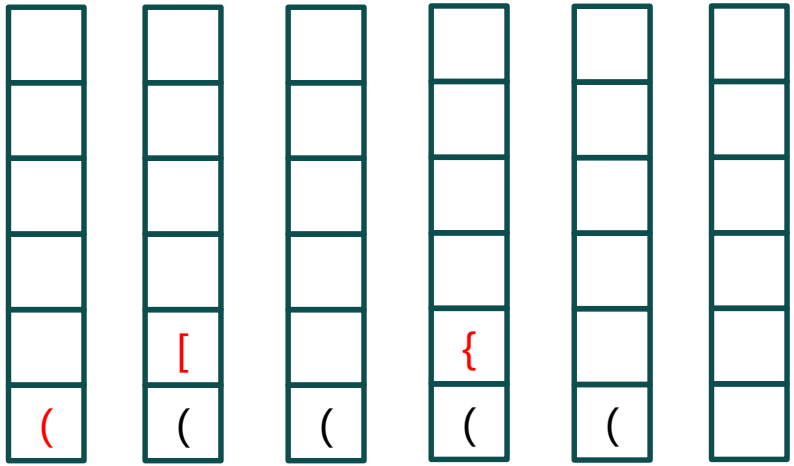
Example: Check opened/closed parentheses

```
For each character c in expression
    If c is opened parentheses push c to stack
    Else If c is closed parentheses
        If stack is empty, there is no match for c
        Else pop a character and match it with c

If stack is empty then all opened parentheses have close one
Else there is at least one that doesn't match;
```

Example: Check opened/closed parentheses

- Expression to check: (a+b[‘name’] – d{0})/2



=> Validated

(a + b [‘name’] – d { 0 })/2

Dynamic Array Implementation

How to implement stack by dynamic array

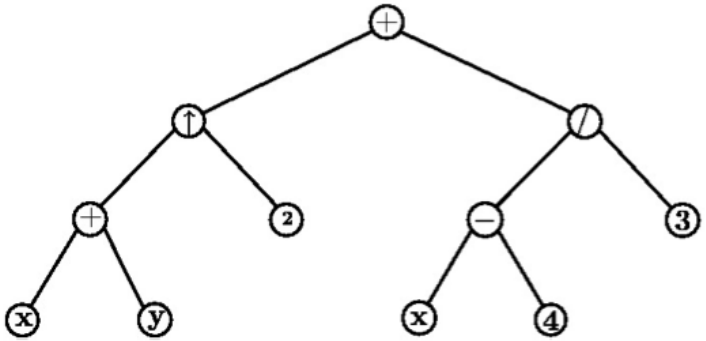
Stack Implementation: Dynamic Array

- If push operation makes stack full => grow up more spaces
  - Double size
  - Add more half size
  - Add constant spaces

```
void push(const int n)
{
    if (is_full())
    {
        stack_size += INC_SIZE;
        stack = (int*) realloc(stack, stack_size * sizeof(int));
    }
    stack[++top] = n;
}
```

Example: Postfix Evaluation

- Consider the expression $(x+y)^2 + (x - 4) / 3$

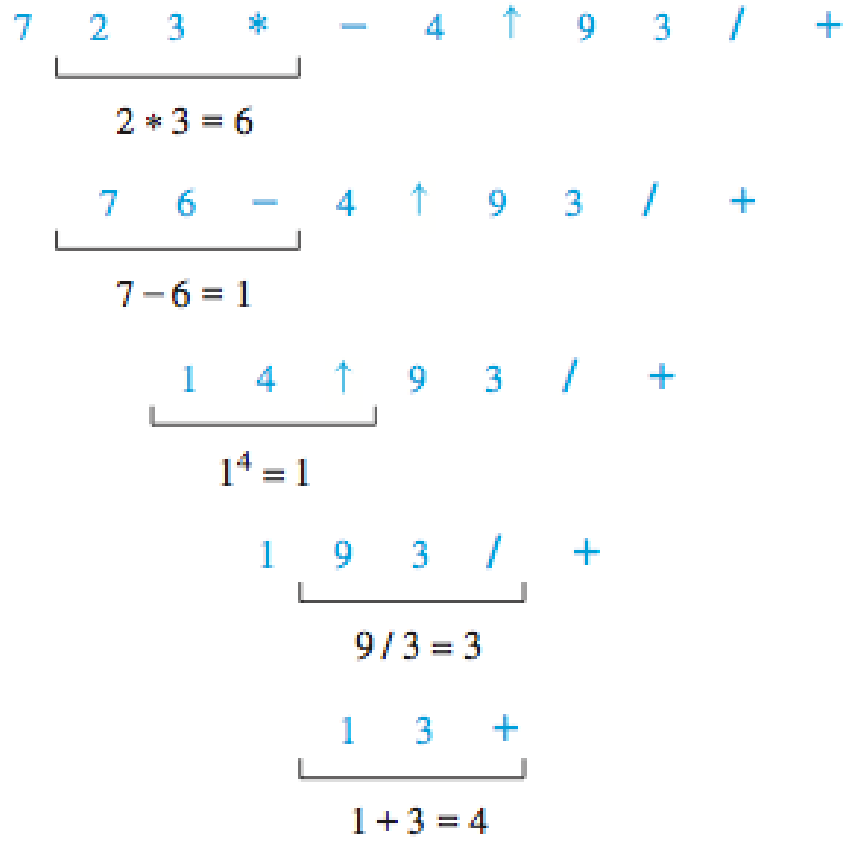


- Prefix notation: +^+xy2/-x43
- Postfix notation: xy+2^x4-3/+
- Infix notation: x+y^2+x-4/3

Example: Postfix Evaluation

- For postfix form, work from left to right, carrying out operations whenever an operator follows two operands.
- After each operation is carried out, the result of this operation is a new operand.
- For prefix form, follows the same procedure, but work from right to left instead.

Example: Postfix Evaluation



Example: Postfix Evaluation

```
While there is token from expression
    If token is number, push it to stack
    Else if token is operator
        pop 2 values from stack
        evaluate with operator then push result to stack
    Else stop because it is not valid postfix expression

If stack size is not 1 then it is not valid postfix
Else pop the result
```

Linked List Implementation

How to implement stack by linked list

Stack Implementation: Linked List

- Stack is Linked List with limited operations and/or other interfaces

Linked List	Stack
Add to head	Push
Remove from head	Pop
Clear list	Clear stack
Is empty	Is empty
Get size	Get size
	To array
	Top

Undo Using Stack: Tic-Tac-Toe Game

- Game board is a matrix 3 x 3
- Move is a struct of (x, y, c)
  - (x, y): coordinate
  - c: character (‘X’ / ‘O’)
- To support player undo, we need to use stack to store ‘states’ of the game
- Which should we store and why?
  - stack of game boards?
  - stack of moves?

Undo Using Stack: Tic-Tac-Toe Game

- Main loop of game play

```
Player make a move
Check if it is undo move
    Undo player's move
    Redraw board
Else (normal move)
    Push player's move to stack
    Redraw board
    Check if game is not over
        Computer make a move
        Push computer's move to stack
        Redraw board
        Check if game is over
```

- Undo player's move

```
Check if stack is not empty
    Undo 2 last moves
Else
    Print error
```

Backtracking With Stack

- Similar idea as “Undo” problem
  - At some “point”, there are options to be chosen
  - Choose one option, save “point” to a stack
  - At some point there is no more “options” then go back to the previous point (by popping from the stack) to take another option
- Popular problem: Find a way to get out of a maze

```
~~~~ MAZE~~~~~
  1   1   1   1   1   1   1   1
  2   0   1   0   0   0   0   1
  1   0   0   0   1   1   0   1
  1   1   1   1   0   0   0   1
  1   0   0   0   0   1   0   1
  1   0   1   0   1   1   1   1
  1   0   0   0   0   0   0   3
  1   1   1   1   1   1   1   1
```

Backtracking With Stack

```
Push start position
Set current position to start
While not found exit position
    If cannot move next from current position
        If cannot go back (stack empty) then no way out
        Else go back by pop the last position
    Else
        Push current position
        Check if current position is exit then stop searching
```