Stack and Queue

Part-I

Stack

Data structure with Last-In First-Out (LIFO) behavior

In

Out

C  B  A

B  C
Typical Operations on Stack

- **isempty**: determines if the stack has no elements
- **isfull**: determines if the stack is full in case of a bounded sized stack
- **top**: returns the top element in the stack
- **push**: inserts an element into the stack
- **pop**: removes the top element from the stack

*push* is like inserting at the front of the list
*pop* is like deleting from the front of the list

Creating and Initializing a Stack

**Declaration**

```c
#define MAX_STACK_SIZE 100
typedef struct {
    int key; /* just an example, can have any type of fields depending on what is to be stored */
} element;

typedef struct {
    element list[MAX_STACK_SIZE];
    int top; /* index of the topmost element */
} stack;
```

**Create and Initialize**

```c
stack Z;
Z.top = -1;
```
Operations

int isfull (stack *s)
{
    if (s->top >= MAX_STACK_SIZE – 1)
        return 1;
    return 0;
}

int isempty (stack *s)
{
    if (s->top == -1)
        return 1;
    return 0;
}

void pop( stack *s )
{
    (s->top)--;  
}

void push( stack *s, element e )
{
    (s->top)++;  
    s->list[s->top] = e;
}

element top( stack *s )
{
    return s->list[s->top];
}

void pop( stack *s )
{
    (s->top)--;  
}
Application: Parenthesis Matching

- Given a parenthesized expression, test whether the expression is properly parenthesized
  - Examples:
    - ()({}{(){}()}()) is proper
    - ()[[]] is not proper
    - {}{} is not proper
    - )([] is not proper
    - (][) is not proper

- Approach:
  - Whenever a left parenthesis is encountered, it is pushed in the stack
  - Whenever a right parenthesis is encountered, pop from stack and check if the parentheses match
  - Works for multiple types of parentheses ( ), { }, [ ]
Parenthesis matching

\[
\text{while (not end of string) do}
\{
\quad a = \text{get_next_token()};
\quad \text{if (a is ‘‘ or ‘{’ or ‘[’) push (a);}
\quad \text{if (a is ‘)’ or ‘}’ or ‘]’)}
\quad \{
\quad \quad \text{if (is_stack_empty( ))}
\quad \quad \{
\quad \quad \quad \text{print ("Not well formed"); exit();}
\quad \quad \} \\
\quad \quad x = \text{top();}
\quad \quad \text{pop();}
\quad \quad \text{if (a and x do not match)}
\quad \quad \{
\quad \quad \quad \text{print ("Not well formed"); exit();}
\quad \quad \}
\quad \}
\}
\text{if (not is_stack_empty( )) print ("Not well formed");}
\]

Recursion can be implemented as a stack

Fibonacci recurrence:
\[
fib(n) = 1 \text{ if } n = 0 \text{ or } 1; \\
\quad = fib(n – 2) + fib(n – 1) \text{ otherwise;}
\]
Fibonacci Recursion Stack

```
0  0  0  1  1  2  3  3  3
5  4  4  4  3  3  3  3
1  2  2  1  1  1  2  1
4  4  4  4  4  4  4  4
11
```