Fundamentals of Computing: Lecture 33

Piyush P Kurur
Office no: 224
Dept. of Comp. Sci. and Engg.
IIT Kanpur

October 30, 2009

Summary of the last class

Summary of the last class

- The `static` key word.

Summary of the last class

- The `static` key word.
- The `extern` key word

Reverse polish calculator

Reverse polish calculator

A calculator where the operators are given in postfix notation

Reverse polish calculator

A calculator where the operators are given in postfix notation
The expression (2 + 3) * 5 is given by 2 3 + 5 *

Reverse polish calculator

A calculator where the operators are given in postfix notation
The expression $(2 + 3)$ * 5 is given by 2 3 + 5 *
In the reverse polish notation there is no need to provide any bracket

Reverse polish calculator

A calculator where the operators are given in postfix notation
The expression $(2 + 3)$ * 5 is given by 2 3 + 5 *
In the reverse polish notation there is no need to provide any bracket Proof ?

Reverse polish calculator

A calculator where the operators are given in postfix notation
The expression $(2 + 3)$ * 5 is given by 2 3 + 5 *
In the reverse polish notation there is no need to provide any bracket Proof ?
Our goal is to write a program for reverse polish notation.

The stack data structure

The stack data structure

A stack is a last in first out data structure.

The stack data structure

A stack is a last in first out data structure.

It supports two operations

The stack data structure

A stack is a last in first out data structure.
It supports two operations

- `push(x)` : 'pushes' the value `x` on top of the stack.

The stack data structure

A stack is a last in first out data structure.
It supports two operations

- `push(x)` : 'pushes' the value `x` on top of the stack.
- `pop()` : 'pops' the top of the stack.

Evaluating a RPN expression using stack

Evaluating a RPN expression using stack

Evaluating a RPN expression using stack

- As you get numbers, keep pushing it.

Evaluating a RPN expression using stack

- As you get numbers, keep pushing it.
- Whenever you get an operator, pop its operand(s), perform the operation and push the result back.

Lexical analysis

Splitting up the input into “tokens”.

Lexical analysis

Splitting up the input into “tokens”.

Lexical analysis

Splitting up the input into “tokens”.

- Number

Lexical analysis

Splitting up the input into “tokens”.

- Number
- operators '+' '-' '*' and '/'

Lexical analysis

Splitting up the input into “tokens”.

- Number
- operators `'+'` `'-'` `'*'` and `'/'`
- `p` for popping the stack and

Lexical analysis

Splitting up the input into “tokens”.

- Number
- operators `'+'` `'-'` `'*'` and `'/'`
- `p` for popping the stack and
- `P` for emptying the stack

Organisation of the code

- `stack.c` is the code for stack manipulation.
- `lex.c` performes a lexical analysis.
- `calc.c` is the main program.