Stack Applications

• The Stack implementation is simple, but there are many applications, including:
  • Evaluating mathematical expressions
  • Searching (Depth-first search)
  • Removing recursion for optimization
  • …

See textbook for details because this is VERY useful!
Evaluating Arithmetic Expressions

- Computer programs regularly use stacks to evaluate arithmetic expressions

- Example: $x \cdot y + z$
  - First rewrite as $xy \cdot z +$
    - *we’ll look at this rewriting process in more detail soon*
  - Then:
    - push $x$
    - push $y$
    - * (pop twice, multiply popped items, push result)
    - push $z$
    - + (pop twice, add popped items, push result)
Converting Expressions

• We (humans) primarily use **infix** notation to evaluate expressions
  • \((x+y)\times z\)

• Computers traditionally used **postfix** (also called Reverse Polish) notation
  • \(xy+z^*\)
  • Operators appear after operands, parentheses are not necessary

• How do we convert between the two?
  • (Compilers do this for us)
Converting Expressions

• Example: \(x\cdot y + z\cdot w\)

• Conversion

  1) Add full parentheses to preserve order of operations
     \(((x\cdot y) + (z\cdot w))\)

  2) Move all operators (\(+\cdot-\)/) after operands
     \(((x\cdot y)(z\cdot w))\+)

  3) Remove parentheses
     \(x\cdot y + z\cdot w\)
Use Stack to Evaluate Postfix Exp

- While there are input “tokens” (i.e., symbols) left:
  - Read the next token from input.
  - If the token is a value, push it onto the stack.
  - Else, the token is an operator that takes n arguments. (It is known that an operator takes n arguments by its definition.)
    - If there are fewer than n values on the stack \(\rightarrow\) error.
    - Else, pop the top n values from the stack and:
      - Evaluate the operator, with the values as arguments.
      - Push the returned result, if any, back onto the stack.
  - The top value on the stack is the result of the calculation.
  - Note that results can be left on stack to be used in future computations:
    - Eg: 3 2 * 4 + followed by 5 / yields 2 on top of stack
Symbolic Example: Converting then Evaluating

- \((x*y) + (z*w) \rightarrow xy*zw*+\)
- Evaluate \(xy*zw*+\):
  - Push \(x\)
  - Push \(y\)
  - Mult: Pop \(y\), Pop \(x\), Push \(x*y\)
  - Push \(z\)
  - Push \(w\)
  - Mult: Pop \(w\), Pop \(z\), Push \(z*w\)
  - Add: Pop \(x*y\), Pop \(z*w\), Push \((x*y)+(z*w)\)
  - Result is now on top of stack
Concrete Example: Converting then Evaluating

- \((x\cdot y) + (z\cdot w) \rightarrow xy\cdot zw\) +

- Evaluate \(xy\cdot zw\) + :
  - Push x
  - Push y
  - Mult: Pop y, Pop x, Push \(x\cdot y\)
  - Push z
  - Push w
  - Mult: Pop w, Pop z, Push \(z\cdot w\)
  - Add: Pop \(x\cdot y\), Pop \(z\cdot w\), Push \((x\cdot y) + (z\cdot w)\)
  - Result is now on top of stack

- Try with: \(w=3, x=4, y=5, z=6\)
PostScript

- PostScript is a programming language used for generating vector graphics
  - Best-known application: describing pages to printers
- It is a stack-based language
  - Values are put on stack
  - Operators pop values from stack, put result back on
  - There are numeric, logic, string values
  - Many operators
- Let’s try it: The ‘gs’ command runs a PostScript interpreter....
- Implementing a tiny part of gs is something we will do in lab... it’s a lot of fun!