Stack applications

Writing a program/function to evaluate a mathematical expression

- If we have a mathematical expression with some variables …
- How does the computer evaluate that expression…..

Total = (((a + 4*a- 300) +(b*c+b/20 +100))/2.0 +(a+200)*(c-d))/total

- Think !!! How will you write a program to evaluate above expression.

Postfix and Prefix representations (Polish notations)

- Sum of A and B can be represented in three ways:

1. +AB Prefix
2. A + B Infix
3. AB + Postfix

- The “pre”, “post” and “in” refer to the relative position of the operator with respect to the two operands.
- **Prefix and Postfix do not require parentheses**

Cont..

Let us examine following mathematical expressions:

- A + B +C

  postfix: AB+C+ prefix: ++ABC

- A + B -C

postfix: AB+C- prefix: -+ABC

- (A + B)* (C +D)

  postfix: AB+CD-* prefix: *+AB-CD

Evaluation of expression A +B*C

- It requires the knowledge of the hierarchy of operators, that is the order in which operators are performed.
- We consider the precedence of * higher than + and therefore assume above expression as---- A +(B*C)
- Hence, to convert A +B*C into postfix or prefix we have to have the knowledge of precedence of operators

Cont...

- The rules for converting from infix to postfix are simple provided we know the order of precedence
- We consider five operators and the precedence from highest to lower is as follows: exponentiation, mul/div, add/sub
- Examples:

(a+b)*(c+d) –z

((a+b)/(c-d)) +z

(a+b)*(c+d)-((z+d)/(m-p) +t) + x*y

Steps/ rules to convert to Postfix

- The rules to remember during conversion are :

1. Parentheses are given highest priority
2. Operations with highest precedence are converted first
3. Operator is placed after first and second operand
4. After a portion of the expression that has been converted to postfix, it is treated as a single operand
5. When **unparenthesized** operators of the same precedence are scanned, the order is assumed from left to right except in the case of exponentiation

A+B+C means (A+B) + C and ABC means $(A)^{(B\$C)}$

Steps/ rules to prefix…

1. Parentheses are given highest priority
2. Operations with highest precedence are converted first
3. Operator is placed before the first and second operand
4. After a portion of the expression that has been converted to prefix, it is treated as a single operand
5. When **unparenthesized** operators of the same precedence are scanned, the order is assumed from left to right except in the case of exponentiation

Note that prefix form is not the mirror image of postfix form

Evaluating a Postfix Expression

- Each operation in the postfix string refers to the previous two operands in the string

Suppose expression is abcd*/+

- Can you write an algorithm to evaluate a postfix expression

Evaluating a Postfix Expression

S = empty stack
Scan the input string one element at a time in **symb** //symb is a char variable
While(not end of input)
{ symb = next input character
If(**symb is an operand**)
Push(s,symb) (push after converting to integer)
else
{ d1 = **pop(s)**
   d2 = **pop(s)**
 value = apply operator on d1 and d2
Push(s,value)
}}
Return(pop(s))

Program to Evaluate a Postfix Expression

```
#include <ctype.h>
 main()
{char pfx[50],ch; int i=0,op1,op2; stack s;
 printf("\n\nRead the Postfix Expression ? ");
 scanf("%s",pfx);
while( (ch=pfx[i++]) != '\0')
{ if(isdigit(ch)) push(&s,ch-'0'); /* Push the operand */
else { op2=pop(&s); op1=pop(&s);
```

Program to Evaluate a Postfix Expression

```
switch(ch)
 { case '+': push(&s,op1+op2);break;
   case '-':push(&s,op1-op2);break;
   case '*':push(&s,op1*op2);break;
   case '/':push(&s,op1/op2);break; }
 }
 } printf("\n Given Postfix Expn: %s\n",pfx); printf("\n Result after Evaluation: %d\n“,pop(&s));
}
```

Converting an expression from Infix to postfix **without parentheses**

- To convert an infix expression to postfix, we have to implement precedence rules
- Define a function prcd(op1,op2) --- returns true if op1 has precedence over op2 (op1 appears to the left of op2 in an infix expression without parentheses)
- Prcd(‘*’, ’+’) and Prcd(‘+’, ’+’) is 1
- Prcd(‘+’, ’*’) is 0

Algorithm to convert from infix to Postfix(without parentheses)

OPS = empty stack

While(not end of input)

{ symb = next input character

If(symb is an operand)

Add symb to the postfix string

Else

Algorithm to convert from infix to Postfix(without parentheses)

```
Else
{  while(!empty(ops) && prcd(stacktop(ops),symb))
   { topsymb = pop(ops)
     add topsymb to the postfix string
   }
 push(ops,symb)
}}
While(!empty(ops)) {
topsymb = pop(ops), add to postfix string
}
```

Converting an expression from Infix to postfix with parentheses

- To convert an infix expression to postfix containing parentheses, we have to implement precedence rules for parentheses:
- Prcd(‘op,’)’) is 1 or true for all operators other than left parentheses
- prcd(‘(’, ’op’) is 0 for false for any operator
- prcd(’op’, ‘(’) is 0 for false for any operator other than ‘)’
- Prcd(’)’, ‘op,) is undefined
- Set prcd(‘(’, ’)’) is false but special action is taken, the opening/left parenthesis is popped and discarded and the closing parenthesis/right is also discarded

Algorithm to convert from infix to Postfix with parentheses

OPS = empty stack
While(not end of input)
{ symb = next input character
If(symb is an operand)
Add symb to the postfix string
Else
{ while(!empty(ops) && prcd(stacktop(ops),symb))
{ topsymb = pop(ops)
add topsymb to the postfix string
} ***** Difference**

- Prcd('op,')') is 1 , prcd('(', 'op') is 0 , prcd('op', '(') is 0, prcd(')', 'op,) is undefined, Set prcd('(', ')') is false

Algorithm to convert from infix to Postfix with parentheses

if(empty(ops) || symb != ‘)’)

push(ops,symb) else topsymb = pop(ops)

}}

While(!empty(ops)) {

topsymb = pop(ops), add to postfix string

}

- Prcd(‘op,’)’) is 1 , prcd(‘(’, ’op’) is 0 , prcd(’op’, ‘(’) is 0, prcd(’)’, ‘op,) is undefined, Set prcd(‘(’, ’)’) is false

Let us write a program to convert infix expression to postfix

- Declare a stack of characters
- Write 5 functions of stack
- Write a function to define the precedence of operators (+,-),(*,/),(^)
- Write a function to check whether a character is an operator or not
- Main program to convert from infix to postfix

1. Read infix expression into infix_array[]

2. while not end of infix_array

Check a character ch from infix_array

1. If(isalphnumeric(ch)) copy into postfix_array
2. else if (ch == '(') push ch on the stack S
3. else if (ch == ')') { while ((stack S not empty) && stack top element != '(‘)

- pop from the stack and copy it into the postfix_array
- throw the right parenthesis by just popping

4. else if (operator(ch))

- while ((stack S not empty) && preced(stack top element) >= preced(ch))
- copy into postfix string after popping from the stack

5. Otherwise Push on the stack

Once whole array is processed pop and copy into postfix array

Some applications of stack

- Parenthesis checking
- Postfix/prefix evaluation
- Infix to postfix/prefix conversion
- Recursion
- Function calling
- Undo and redo in text editor
- Search history