CONTENTS

Chapters and Topics **page**

CHAPTER – 22

INFIX, POSTFIX, PREFIX

PRECEDENCE OBSERVATIONS

Since **precedence** plays such an important role in transforming **infix** to **postfix**, let us assume the existence of a function ***prcd*** **(***op1*, *op2*), where *op1* and *op2* are characters representing operators. This function returns ***TRUE*** if op1 has **precedence** over *op2* when *op1* appears to the left of *op2* in an **infix** expression without parentheses. ***prcd*** (*op1*, *op2*) returns ***FALSE*** otherwise. ***prcd*** ('*', '+') and ***prcd*** ('+', '+') are ***TRUE***, whereas ***prcd*** ('+', '*') is ***FALSE***.

Note that at each point of the simulation, an **operator** on the **stack** has a lower **precedence** than all the **operators** above it. This is because the initial **empty stack** trivially satisfies this condition, and an **operator** is **pushed** onto the **stack** only if the **operator** currently on **top** of the **stack** has a lower **precedence** than the incoming **operator**.

When an opening parenthesis is read, it must be pushed on the **stack**. This can be done by establishing the convention that ***prcd*** (*op*, '(') equals ***FALSE***, for any operator symbol ***op*** other than a right parenthesis. In addition, ***prcd*** ('(', *op*) should be ***FALSE*** for any operator symbol ***op***. This ensures that an **operator** symbol appearing after a left parenthesis is pushed onto the **stack**.

When a closing parenthesis is read, all **operators** up to the first opening parenthesis must be **popped** from the **stack** into the **postfix string**. This can be achieved by defining ***prcd*** (*op*, '**)**') as ***TRUE*** for all operators op other than a left parenthesis. When these **operators** have been **popped** off the **stack** and the opening parenthesis is uncovered, special action must be taken. The opening parenthesis must be **popped** off the **stack** and the closing parenthesis is discarded rather than placed in the **postfix string** or

on the **stack**. Defining ***prcd*** (‘**(**‘, ‘**)**’) to ***FALSE***, ensures that upon reaching an opening parenthesis, the loop is skipped, so that the opening parenthesis is not inserted into the **postfix string**. The closing parenthesis should not be **pushed** onto the **stack**.

EVALUATING A PREFIX EXPRESSION

Algorithm to evaluate prefix expression:

```
operandstk    =  the empty stack;
prevoperand  =  false;
/* Need to read the multiple digits in the right order after prefix
conversion.  Need to convert the multiple digits to numeric values.*/
while (not end of input)  /* reading from end to beginning  */
{
        symb = next input character;

        if (symb is an operand)
        {
           push (operandstk, symb);
        }       /* if symb is an operand */
        else   /* symb is an operator    */
        {
           operator    =  symb;
           operand1  =  pop (operandstk);
           operand2  =  pop (operandstk);
            value       =  oper (operator, operand1, operand2);
            push (operandstk, value);

         }      /* else symb is an operator                */

}               /* end of not end of input while loop   */
```

Process the operandstk for leftovers

```
return (pop (operandstk) );
```

CONVERTING AN INFIX EXPRESSION TO PREFIX

Algorithm to convert infix string without parentheses into a prefix:

```
/*  need to reverse the string sometimes. prcd  may need to be
    reversed for left and right params */
opstk = the empty operator stack
while (not end of input buffer reading from back)
{
      /* symbol can be space too */
       symb = next input character;
       if (symb is an operand)
          add symb to the prefix string;
       else
       {
          popandtest (opstk, topsymb, underflow);
          while (!underflow && prcd (stacktop (opstk), symb) )
          {
              add topsymb to the prefix string;
              popandtest (opstk, topsymb, underflow);
          }  /* end of not empty while loop */
          if (!underflow)
             push (opstk, symb);
          if (underflow or hit open parenthesis)
             push (opstk, symb);
          else
              topsymb = pop (opstk);
       }  /* end of else operator  */
}   /* end of not end of input while loop  */

/* output any remaining operators */
while (!empty (opstk) )
{
       topsymb = pop (opstk);
       add topsymb to the prefix string;
}  /*end of while not  empty */
```