Stacks

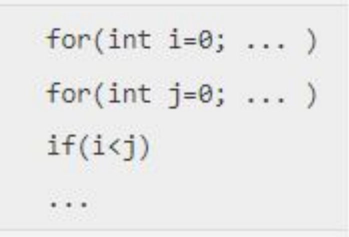
tutorial

Good coding style

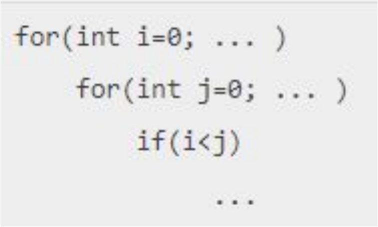
- Self-explanatory - Comments should be added to those parts of the code where explanations are needed.
- should not be clever if it is not necessary – do things in an obvious straightforward way as long as it is efficient
- organize the code in small units (structs/functions) - make it reusable

Few guidelines

Indentation



```
for(int i=0; ... )
for(int j=0; ... )
if(i<j)
...
```



```
for(int i=0; ... )
    for(int j=0; ... )
        if(i<j)
            ...
```

Meaningful names of variables and functions

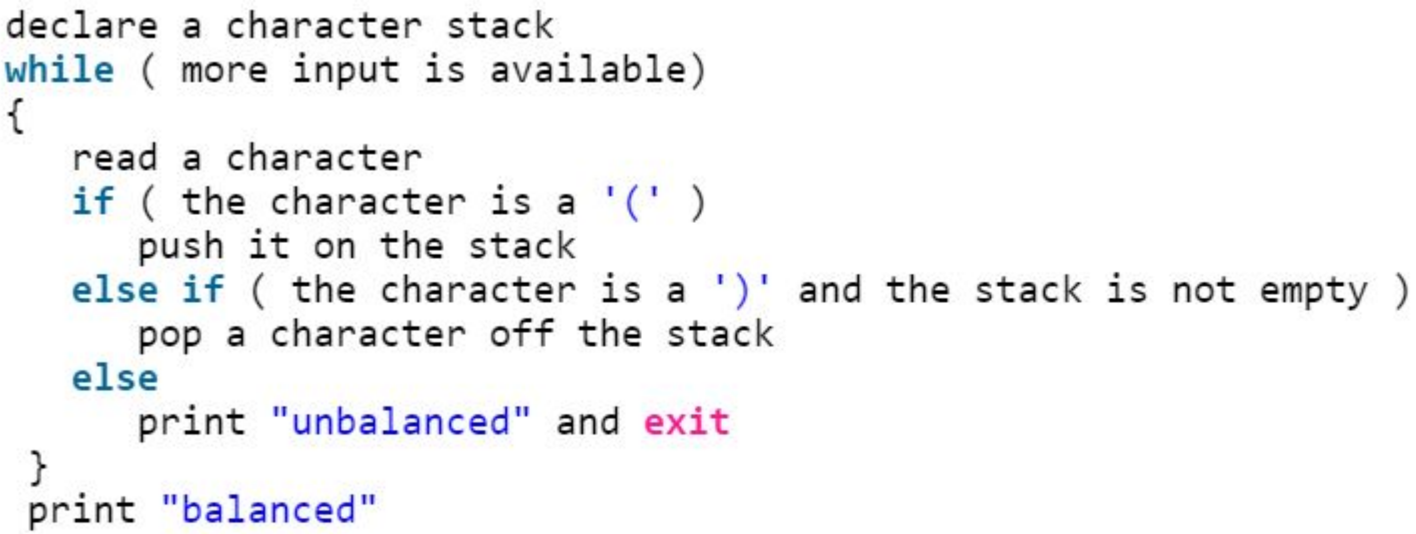
Follow consistent style

Upto 80 characters in a line

Obtain the binary representation of a number

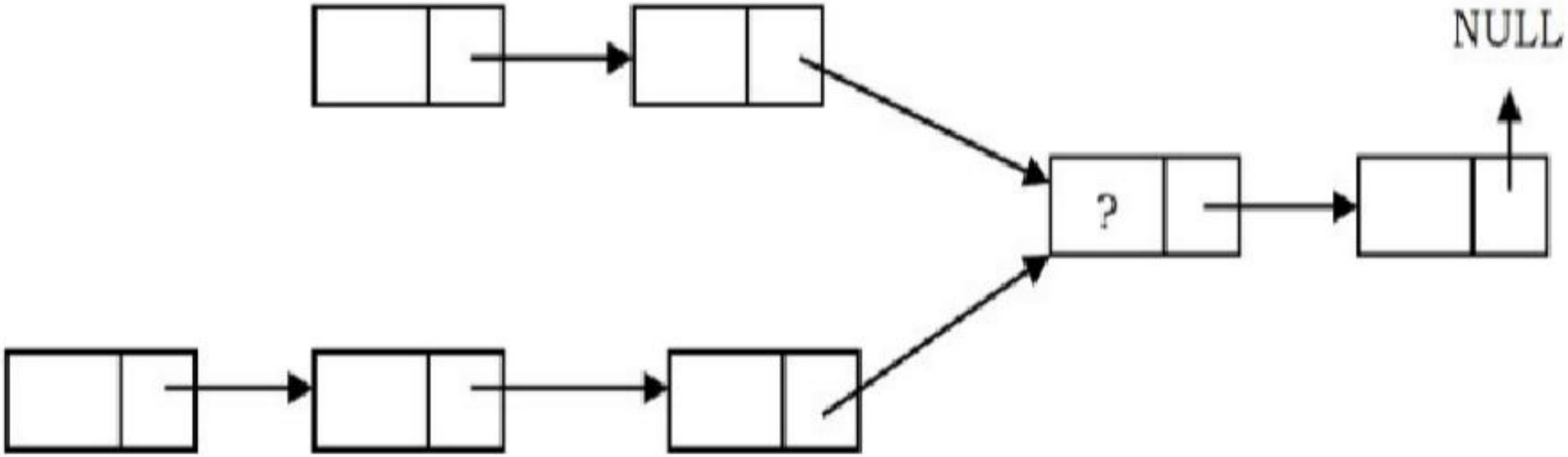
check if string is a palindrome

Check if string has balanced parentheses - contains only ‘(‘, ‘)’



```
declare a character stack
while ( more input is available)
{
   read a character
   if ( the character is a '(' )
      push it on the stack
   else if ( the character is a ')' and the stack is not empty )
      pop a character off the stack
   else
      print "unbalanced" and exit
 }
 print "balanced"
```

Two singly linked lists both of which intersect at some point. Find the merging point



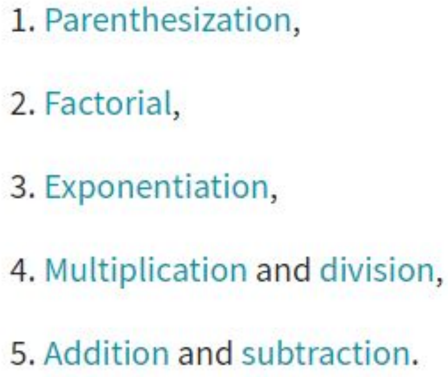
Infix, Prefix, Postfix

Infix : the notation used commonly. Eg 2 + 2 ; <exp> OP <exp>

Prefix : operators precede operands. Eg + 2 2

Postfix : operators succeed operands. Eg 2 2 +

Pre/Postfix express the intended order of operations without the need for parentheses

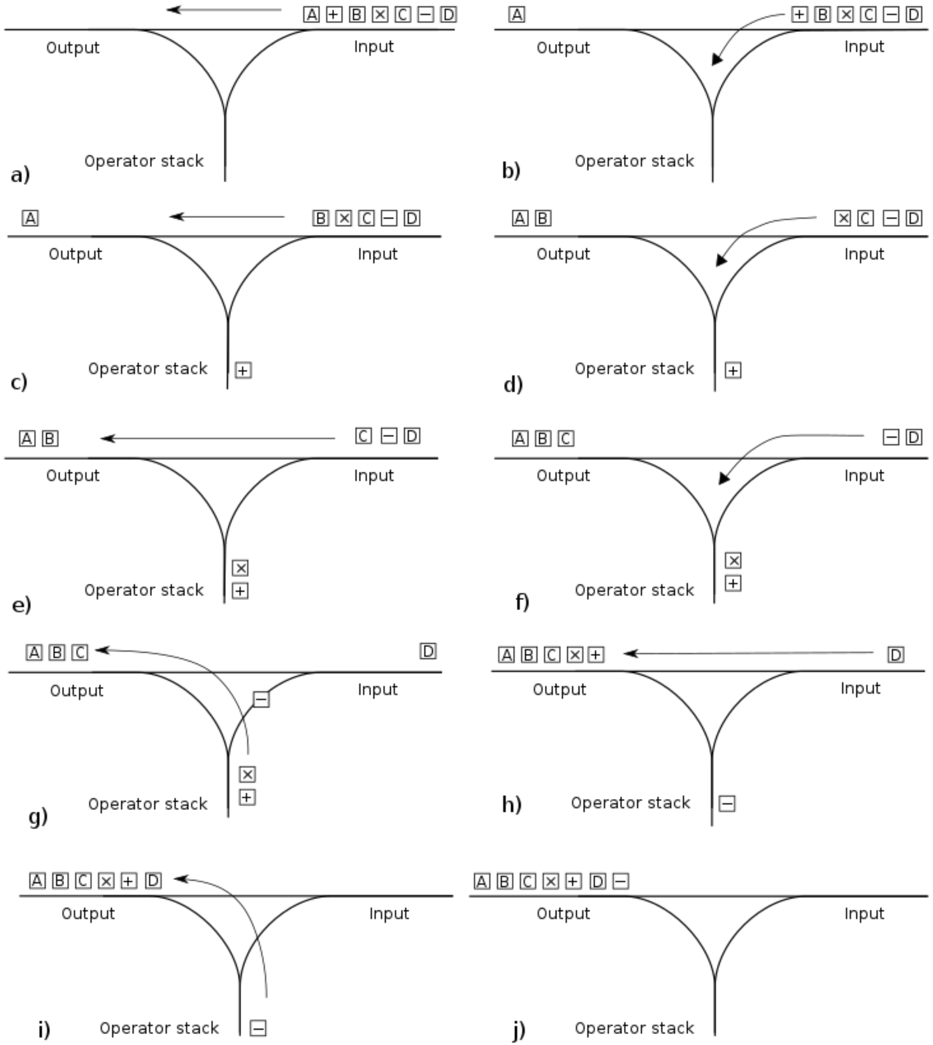


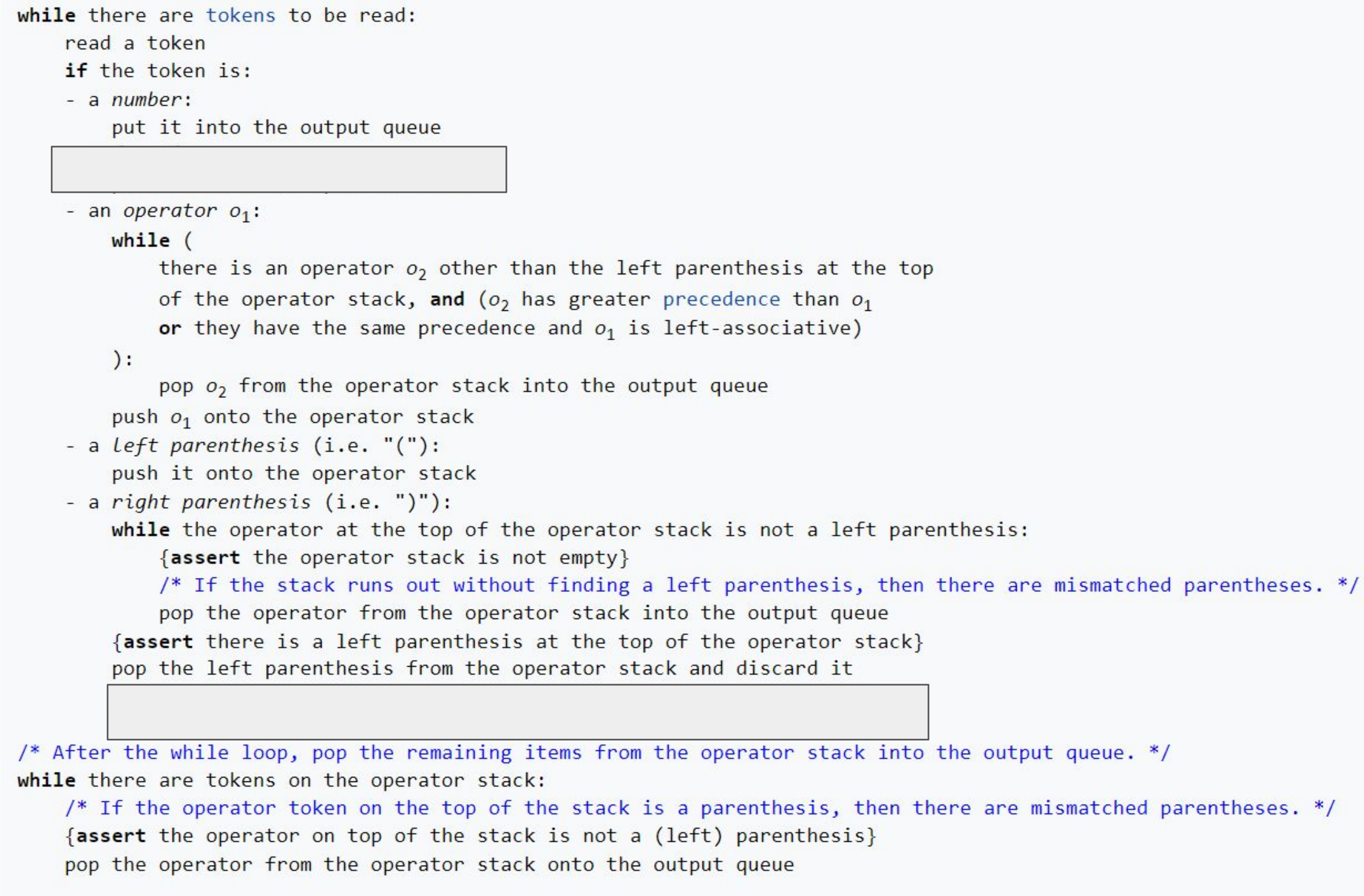
1. Parenthesization,
2. Factorial,
3. Exponentiation,
4. Multiplication and division,
5. Addition and subtraction.

Infix to Postfix

2 + 2 -> 2 2 +

A * B - (C + D) + E →


a)
A + B × C − D
Output
Input
Operator stack
b)
A
+ B × C − D
Output
Input
Operator stack
c)
A
B × C − D
Output
Input
Operator stack
+
d)
A B
× C − D
Output
Input
Operator stack
+
e)
A B
C − D
Output
Input
×
Operator stack
+
f)
A B C
− D
Output
Input
×
Operator stack
+
g)
A B C
D
Output
−
Input
×
+
Operator stack
h)
A B C × +
D
Output
Input
Operator stack
−
i)
A B C × + D
Output
Input
Operator stack
−
j)
A B C × + D −
Output
Input
Operator stack



```
while there are tokens to be read:
    read a token
    if the token is:
    - a number:
        put it into the output queue

    - an operator $o_1$:
        while (
            there is an operator $o_2$ other than the left parenthesis at the top
            of the operator stack, and ($o_2$ has greater precedence than $o_1$
            or they have the same precedence and $o_1$ is left-associative)
        ):
            pop $o_2$ from the operator stack into the output queue
        push $o_1$ onto the operator stack
    - a left parenthesis (i.e. "("):
        push it onto the operator stack
    - a right parenthesis (i.e. ")"):
        while the operator at the top of the operator stack is not a left parenthesis:
            {assert the operator stack is not empty}
            /* If the stack runs out without finding a left parenthesis, then there are mismatched parentheses. */
            pop the operator from the operator stack into the output queue
        {assert there is a left parenthesis at the top of the operator stack}
        pop the left parenthesis from the operator stack and discard it

/* After the while loop, pop the remaining items from the operator stack into the output queue. */
while there are tokens on the operator stack:
    /* If the operator token on the top of the stack is a parenthesis, then there are mismatched parentheses. */
    {assert the operator on top of the stack is not a (left) parenthesis}
    pop the operator from the operator stack onto the output queue
```

Evaluate Postfix

```
While there are input tokens left
Read next token
If token is an operand:
    Push
Else:
    Pop, Operate on the elements and push
Return the top
```

123*+5-