Arithmetic forms

1. Infix form. You are accustomed to infix notation. The operation is between the two operands. For example, the expression 5 + 6 is equal to 11. The operator, +, is between the two operands, 5 and 6.
2. Prefix form. The operator precedes the two (or more) operands. For example, + 5 6 yields 11. Functions are in prefix form. The operator is the function and operands are arguments. Consider function max that returns the larger of the two arguments. max(5,6) returns a value of 6.
3. Postfix form. In postfix notation, the operator follows the operands. You are familiar with this type of arithmetic if you use Hewlett-Packard calculators. The advantage of this form is that it requires no parentheses. For example, 5 6 + yields 11.

Consider a longer sequence:
5 6 * 64 32 - - 2 / 1 +
The result of evaluating the sequence is 0.

A stack can be used to evaluate postfix notation.
Rule 1. Push operands.
Rule 2. Operators pop operands from the stack, perform the operation, and push results on to the stack.

Stack (after operation)										
				32						
	6		64	64	32		2		1	
5	5	30	30	30	30	-2	-2	-1	-1	0
5	6	*	64	32	-	-	2	/	1	+

Program **p03** evaluates postfix expressions. The problem is to recognize integer operands and arithmetic operators. Ordinarily a scanner is used to recognize tokens. Program **p03** contains a scanner component.

File	Description
p03.cpp	Processes command line arguments and evaluates postfix expressions.
Stack03.cpp	Stack ADT implementation using a dynamically allocated elements.
Stack03.h	Class Stack
Scan03.l	Lex specification for the scanner required by program **p03**.
Scan03.h	Scanner interface
p03make	Instructions for the UNIX utility make. Instructions direct make to compile and bind program **p03**.

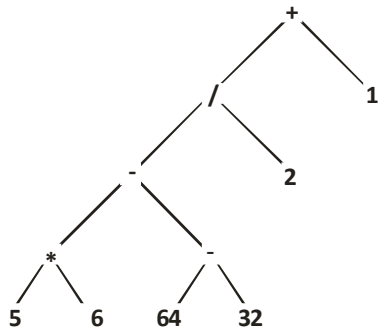

Figure 1. Expression Tree for Postfix Expression

Project **p03** notes.

1. You must open the input file using function *fopen* defined in include file *cstdio*.

```
…
#include <cstdio>
…
char ifn[255];                  //Input file name
FILE* i=fopen(ifn,"r");         //Open the file whose name is stored in string ifn.
…
```

2. You must close the input file using function *fclose* defined in include file *cstdio*.

```
…
#include <cstdio>
…
char ifn[255];                  //Input file name
FILE* i=fopen(ifn,"r");         //Open the file whose name is stored in string ifn.
…
fclose(i);                      //Close FILE i.
…
```

3. You must redirect the input file from the keyboard to the input file whose name was given on the command line before program **p03** reads any input data. The constructor for the scanner defined in file **Scan03.h** performs this function.

```
void PostfixMgr(FILE* i,ostream& o)
{   Scan L(i);                  //Redirect the input from the keyboard to FILE i
    …
}
```

4. A token consists of an integer code and a string. When member function *Lex* is called, it returns the integer code for the token recognized. Member function *Lex* returns a value of zero (**0**) when the end-of-file mark is read. After function Lex is called, member function *FetchSpelling* can be called to return the string accepted by the scanner.

```
#include "Scan03.h"
void PostfixMgr(FILE* i,ostream& o)
{   Scan L(i);                      //Redirect the input from the keyboard to FILE i
    …
    for (;;) {
        int t=L.Lex();              //Obtain the token code of the next token
        if (t==0) break;            //Stop when the EOF is read
        cout << endl;
        cout << setw(4) << t;       //Print the token code
        cout << setw(30)            //Print the string accepted by scanner L.
             << L.FetchSpelling();
    }
}
```

5. If the token accepted is a string that can be interpreted as an integer, it must be converted to an actual integer.

```
#include "Scan03.h"
void PostfixMgr(FILE* i,ostream& o)
{   Scan L(i);                      //Redirect the input from the keyboard to FILE i
    …
    for (;;) {
        int t=L.Lex();              //Obtain the token code of the next token
        if (t==0) break;            //Stop when the EOF is read
        if (t==INTLIT) {            //Was the token an integer literal?
            int k=L.Intlit();       //Obtain the integer from member function Intlit
        }

}
```