CS 1723, Data Structures, Fall Semester, 1998
Programming Assignment 5, Due October 9, 1998
Translation to Reverse Polish

The Assignment: For this assignment you will write a program `translate` that will translate an ordinary arithmetic expression into *reverse Polish* form. This program should be used in conjunction with the program `evaluate` of the earlier Assignment 2 to find the final value of an input arithmetic expression. For the final runs after initial testing, you can pipe the output of `translate` in as input to `evaluate`, so that you use the command:

```
runner% translate <sourcefile | evaluate
```

The input source will be made up of the following elements:

- single-digit integer constants, `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, or `9`.
- operators: $\wedge$, `*` `/`, `+`, or `-`.
- parentheses: `(` and `)`.
- the special symbols: `#` to terminate the whole input.

This assignment limits to single-digit integer constants in order to keep things simpler.

For example, here is sample input source, which calculates one root of the equation $y = x^2 + 3x + 2$:

```
((3^2 - 4*1*2)^(1/2) - 3)/(2*1)#
```

Notice that all *whitespace* (= space, newline, tab, etc.) should be ignored on input.

Overall organization: All files and programs should be contained in a single directory: `assign5`. You will be combining a new program (in three files) with the old program from Assignment 2.

The program `translate.c`: Organize the program as (at least) *three* files: `translate.c`, `tstack.h`, and `tstack.c`.

The `tstack.h` and `tstack.c` files should implement a stack of characters. (You will probably want to add a function `top()` that will return the topmost character without popping it.)

In addition to `main()`, the `translate.c` file should contain functions with prototypes

```
void checkprec (char c, int *prec, char *assoc);
int getnext(void);
```

- The function `checkprec()` should return the precedence and associativity of the operator stored in the variable "c", according to the following table:

Operator	Precedence	Associativity
$\wedge$	5	R
*	4	L
/	4	L
+	3	L
–	3	L
(	1	R
)	1	R

  There *must* be a static array of structures inside `checkprec()` which stores this information. This structure *must* be initialized using a list of initializers enclosed in braces, as shown in Section 6.3 of the white book (page 133).

- The function `getnext()` should return the next non-whitespace character. (Use `isspace()` in `<ctype.h>`.)

The translation algoritm will be discussed in class. Notice that for each non-whitespace character, the program must decide whether to output it or to stack it. Operands (digits and letters) are always output immediately. Operators (including parentheses) are always stacked, but in some cases not before one or more other operators are popped and output. The following rules apply:

1. Always push a left parenthesis.
2. Always push an operator if the stack is empty, if an operator of lower precedence is on top, or if an operator of equal precedence is on top and right-to-left associativity applies.
3. Otherwise pop and output the stacktop, and try rules 1 and 2 again.
4. A left and a right parenthsis (right above left) on the stack are always simply deleted.
5. A # character terminates the current input. Ouput a # and terminate the program.

The result of the sample input above should be:

```
32^41*2*-12/^3-21*/#
```

The initial `makefile` for `translate.c`:

```
# makefile for the translate program
translate: translate.c tstack.c tstack.h
        cc -g -o translate translate.c tstack.c
tlint:
        lint -m -u translate.c tstack.c
```

The second program `evaluate.c`: Use the program that you wrote for Assignment 2.

Required Execution: You must try out the following table of input arithmetic expressions, with the translations and final values as shown. (Some students may have trouble getting all of these to work correctly; should should turn in your program anyway.)

Input arithmetic expression	Output reverse Polish	Final value
`2+3*4#`	`234*+#`	14.000000
`3*4+5#`	`34*5+#`	17.000000
`(2+3)*4#`	`23+4*#`	20.000000
`(3*(2+4)/(5+1))-2#`	`324+*51+/2-#`	1.000000
`(5+3)^(2+1)^2#`	`53+21+2^^#`	134217728.000000
`2+3*4^5*6+7#`	`2345^*6*+7+#`	18441.000000
`((3^2-4*1*2)^(1/2)-3)/(2*1)#`	`32^41*2*-12/^3-21*/#`	-1.000000
`((2-3)^((4+1)*5)/6-(2-4)*7)-8#`	`23-41+5*^6/24-7*-8-#`	5.833333