1. Introduction. This is the main segment for the calculator program from Chapter 4 of Kernighan and Ritchie's *The C Programming Language*, which I'm using as a test to see how `CWEB` handles function prototypes, separately compiled modules, and the like.

Since this is being typed in from the C book, which was not written with `CWEB` in mind, it probably won't seem as neatly presented as most `CWEB` code.

Here is the only unnamed code module in this file, giving an overview of the program:

```
⟨ Included header files 2 ⟩
⟨ Main program 4 ⟩
```

2. We need two header files from the C library. One provides the standard I/O functions, the other provides the function that converts strings to floating point numbers (*atof*).

```
⟨ Included header files 2 ⟩ ≡
#include <stdio.h>
#include <stdlib.h>      /* for atof( ) */
```

See also sections 3, 11, 22, 23, and 31.

This code is used in sections 1, 10, 21, and 30.

3. We also need header files from other segments of this program, declaring the interface that we need in order to recognize the functions defined in those segments.

```
⟨ Included header files 2 ⟩ +≡
#include "getop.h"      /* for getop( ) */
#include "stack.h"      /* for push( ) and pop( ) */
```

4. The main program. This is the top level loop for our reverse Polish calculator.

```
#define MAXOP 100 /* the maximum size allowed for a single operand or operator */
⟨ Main program 4 ⟩ ≡
  main( )
  {
    int type;
    char s[MAXOP];
    ⟨ Other local variables of main 6 ⟩
    while ((type = getop(s)) ≠ EOF) {
      switch (type) {
        ⟨ Case for numbers 8 ⟩
        ⟨ Cases for commutative operators 7 ⟩
        ⟨ Cases for non-commutative operators 5 ⟩
        ⟨ Case for newlines 9 ⟩
      default: printf("error: unknown command %s\n", s);
        break;
      }
    }
    return 0;
  }
```

This code is used in section 1.

5. Non-commutative operators are tricky. We'd like to be able to say something like

$$push(pop( ) - pop( ));$$

but that would be wrong, because it assumes that the $pop( )$s are executed in a certain order, which C does not guarantee (the compiler is free to determine order of evaluation of function calls in a single expression). So we have to use an explicit temporary to make sure that the topmost stack element becomes the second and not the first operand.

```
⟨ Cases for non-commutative operators 5 ⟩ ≡
case '-': op2 = pop( );
  push(pop( ) - op2);
  break;
case '/': op2 = pop( );
  if (op2 ≠ 0.0) push(pop( )/op2);
  else printf("error: zero divisor\n");
  break;
```

This code is used in section 4.

6. Here we declare the variable we used above.

```
⟨ Other local variables of main 6 ⟩ ≡
  double op2;
```

This code is used in section 4.

7. Having seen the handling of non-commutative operators, you can appreciate the comparative simplicity of handling commutative ones.

⟨ Cases for commutative operators 7 ⟩ ≡
```
case '+': push(pop() + pop());
  break;
case '*': push(pop() * pop());
  break;
```
This code is used in section 4.

8. The handling of numbers is easy: we parse the string that represents the number, obtaining an actual numerical value, and we push that value onto the stack.

⟨ Case for numbers 8 ⟩ ≡
```
case NUMBER: push(atof(s));
  break;
```
This code is used in section 4.

9. When we see a newline character, we print the top element of the stack.

⟨ Case for newlines 9 ⟩ ≡
```
case '\n': printf("\t%.8g\n", pop());
  break; ° def ° cweb{° .{CWEB}}
```
This code is used in section 4.

10. Introduction. This is a segment from the calculator program from Chapter 4 of Kernighan and Ritchie's *The C Programming Language*, which I'm using as a test to see how `CWEB` handles function prototypes, separately compiled modules, and the like.

This segment defines the *push* and *pop* procedures, which manage the operand stack.

Since this is being typed in from the C book, which was not written with `CWEB` in mind, it probably won't seem as neatly presented as most `CWEB` code.

Here is the only unnamed code module in this file.

⟨ Included header files 2 ⟩
⟨ Private variables for this source file 16 ⟩
⟨ Functions defined in this source file 12 ⟩

11. We need one header file from the C library. It provides the standard I/O functions.

⟨ Included header files 2 ⟩ +≡
```
#include <stdio.h>
```

12. This source file defines two functions.

⟨ Functions defined in this source file 12 ⟩ ≡
⟨ Definition of *push*( ) 18 ⟩
⟨ Definition of *pop*( ) 20 ⟩

See also sections 24 and 32.

This code is used in sections 10, 21, and 30.

13. Each function defined here has to have its prototype exported, so that functions in other source files that want to call the functions defined here will have the necessary declarations available.

⟨ Function prototypes to be exported 13 ⟩ ≡
⟨ Function prototype for *push*( ) 17 ⟩;
⟨ Function prototype for *pop*( ) 19 ⟩;

See also sections 25 and 33.

This code is used in sections 14, 26, and 34.

14. In this module we collect up information that needs to be written to the header file `stack.h` so that other source files that want to make use of the function defined here will have the necessary declarations available.

⟨ `stack.h` 14 ⟩ ≡
⟨ Function prototypes to be exported 13 ⟩

15. The functions *push*( ) and *pop*( ).

16. This defines the stack data structure that the routines *push*( ) and *pop*( ) share.

```
#define MAXVAL 100 /* maximum depth of val stack */
⟨ Private variables for this source file 16 ⟩ ≡
  static int sp = 0; /* next free stack position */
  static double val[MAXVAL]; /* value stack */
```

See also section 36.

This code is used in sections 10 and 30.

17.

```
⟨ Function prototype for push( ) 17 ⟩ ≡
  void push(double f)
```

This code is used in sections 13 and 18.

18.

```
⟨ Definition of push( ) 18 ⟩ ≡
  ⟨ Function prototype for push( ) 17 ⟩
  {
    if (sp < MAXVAL) val[sp++] = f;
    else printf("error: stack full, can't push %g\n", f);
  }
```

This code is used in section 12.

19.

```
⟨ Function prototype for pop( ) 19 ⟩ ≡
  double pop(void)
```

This code is used in sections 13 and 20.

20.

```
⟨ Definition of pop( ) 20 ⟩ ≡
  ⟨ Function prototype for pop( ) 19 ⟩
  {
    if (sp > 0) return val[--sp];
    else {
      printf("error: stack empty\n");
      return 0.0;
    }
  }
  ° def ° cweb{° .{CWEB}}
```

This code is used in section 12.

21. Introduction. This is a segment from the calculator program from Chapter 4 of Kernighan and Ritchie's *The C Programming Language*, which I'm using as a test to see how `CWEB` handles function prototypes, separately compiled modules, and the like.

This segment defines the *getop* procedure, which reads the input looking for an operator or operand.

Since this is being typed in from the C book, which was not written with `CWEB` in mind, it probably won't seem as neatly presented as most `CWEB` code.

Here is the only unnamed code module in this file.

⟨ Included header files 2 ⟩
⟨ Public # **define** statements to be exported 29 ⟩
⟨ Functions defined in this source file 12 ⟩

22. We need two header files from the C library. One provides functions for recognizing digits and other character classes. The other provides standard I/O definitions, and we need it only for the definition of `EOF`.

⟨ Included header files 2 ⟩ +≡
```
#include <ctype.h>
#include <stdio.h>
```

23. We also need a header file from another segment of this program, declaring the interface that we need in order to recognize the functions defined in that segment.

⟨ Included header files 2 ⟩ +≡
```
#include "getch.h"
```

24. As it happens, this file defines only one function: *getop*( ).

⟨ Functions defined in this source file 12 ⟩ +≡
⟨ Definition of *getop*( ) 28 ⟩

25. The function defined here has to have its prototype exported, so that functions in other source files that want to call this one will have the necessary declaration available.

⟨ Function prototypes to be exported 13 ⟩ +≡
⟨ Function prototype for *getop*( ) 27 ⟩;

26. In this module we collect up information that needs to be written to the header file `getop.h` so that other source files that want to make use of the function defined here will have the necessary declarations available.

⟨ `getop.h` 26 ⟩ ≡
⟨ Public # **define** statements to be exported 29 ⟩
⟨ Function prototypes to be exported 13 ⟩

27. The function *getop*( ).

⟨ Function prototype for *getop*( ) 27 ⟩ ≡
  int *getop*(**char** *s*[ ])

This code is used in sections 25 and 28.

28.

```
⟨ Definition of getop( ) 28 ⟩ ≡
  ⟨ Function prototype for getop( ) 27 ⟩
  {
    int i, c;
    while ((s[0] = c = getch( )) ≡ '␣' ∨ c ≡ '\t') ;
    s[1] = '\0';
    if (¬isdigit(c) ∧ c ≠ '.') return c;      /* not a number */
    i = 0;
    if (isdigit(c))      /* collect integer part */
      while (isdigit(s[++i] = c = getch( ))) ;
    if (c ≡ '.')      /* collect fraction part */
      while (isdigit(s[++i] = c = getch( ))) ;
    s[i] = '\0';
    if (c ≠ EOF) ungetch(c);
    return NUMBER;
  }
```

This code is used in section 24.

29. This defines the signal that *getop*( ) returns when it sees a number (any number). This is used within the code of *getop*( ) *and* in the routine that calls *getop*( ) (which means it must be included in the header file `getop.h`).

```
⟨ Public # define statements to be exported 29 ⟩ ≡
#define NUMBER  '0'
  ° def ° cweb{°.{CWEB}}
```

This code is used in sections 21 and 26.

30. Introduction. This is a segment from the calculator program from Chapter 4 of Kernighan and Ritchie's *The C Programming Language*, which I'm using as a test to see how `CWEB` handles function prototypes, separately compiled modules, and the like.

This segment defines the *getch* and *ungetch* procedures, which perform character-by-character reading and un-reading of the input stream.

Since this is being typed in from the C book, which was not written with `CWEB` in mind, it probably won't seem as neatly presented as most `CWEB` code.

Here is the only unnamed code module in this file.

⟨ Included header files 2 ⟩
⟨ Private variables for this source file 16 ⟩
⟨ Functions defined in this source file 12 ⟩

31. We need one header file from the C library. It provides the standard I/O functions.

⟨ Included header files 2 ⟩ +≡
```
#include <stdio.h>
```

32. This source file defines two functions.

⟨ Functions defined in this source file 12 ⟩ +≡
⟨ Definition of *getch*( ) 38 ⟩
⟨ Definition of *ungetch*( ) 40 ⟩

33. Each function defined here has to have its prototype exported, so that functions in other source files that want to call the functions defined here will have the necessary declarations available.

⟨ Function prototypes to be exported 13 ⟩ +≡
⟨ Function prototype for *getch*( ) 37 ⟩;
⟨ Function prototype for *ungetch*( ) 39 ⟩;

34. In this module we collect up information that needs to be written to the header file `getch.h` so that other source files that want to make use of the function defined here will have the necessary declarations available.

⟨ `getch.h` 34 ⟩ ≡
⟨ Function prototypes to be exported 13 ⟩

35. The functions *getch*( ) and *ungetch*( ).

36. First we define the buffer that the routines *getch*( ) and *ungetch*( ) share.

```
#define BUFSIZE 100 /* maximum depth of val stack */
⟨ Private variables for this source file 16 ⟩ +≡
  static char buf[BUFSIZE]; /* buffer for ungetch */
  static int bufp = 0; /* next free position in buf */
```

37.

```
⟨ Function prototype for getch( ) 37 ⟩ ≡
  int getch(void)
```

This code is used in sections 33 and 38.

38.

```
⟨ Definition of getch( ) 38 ⟩ ≡
  ⟨ Function prototype for getch( ) 37 ⟩
  {
    return (bufp > 0) ? buf[--bufp] : getchar( );
  }
```

This code is used in section 32.

39.

```
⟨ Function prototype for ungetch( ) 39 ⟩ ≡
  void ungetch(int c)
```

This code is used in sections 33 and 40.

40.

```
⟨ Definition of ungetch( ) 40 ⟩ ≡
  ⟨ Function prototype for ungetch( ) 39 ⟩
  {
    if (bufp > BUFSIZE) printf("ungetch: too many characters\n");
    else buf[bufp++] = c;
  }
```

This code is used in section 32.

41. Index.

⟨ Case for newlines 9 ⟩ Used in section 4.
⟨ Case for numbers 8 ⟩ Used in section 4.
⟨ Cases for commutative operators 7 ⟩ Used in section 4.
⟨ Cases for non-commutative operators 5 ⟩ Used in section 4.
⟨ Definition of *getch*( ) 38 ⟩ Used in section 32.
⟨ Definition of *getop*( ) 28 ⟩ Used in section 24.
⟨ Definition of *pop*( ) 20 ⟩ Used in section 12.
⟨ Definition of *push*( ) 18 ⟩ Used in section 12.
⟨ Definition of *ungetch*( ) 40 ⟩ Used in section 32.
⟨ Function prototype for *getch*( ) 37 ⟩ Used in sections 33 and 38.
⟨ Function prototype for *getop*( ) 27 ⟩ Used in sections 25 and 28.
⟨ Function prototype for *pop*( ) 19 ⟩ Used in sections 13 and 20.
⟨ Function prototype for *push*( ) 17 ⟩ Used in sections 13 and 18.
⟨ Function prototype for *ungetch*( ) 39 ⟩ Used in sections 33 and 40.
⟨ Function prototypes to be exported 13, 25, 33 ⟩ Used in sections 14, 26, and 34.
⟨ Functions defined in this source file 12, 24, 32 ⟩ Used in sections 10, 21, and 30.
⟨ Included header files 2, 3, 11, 22, 23, 31 ⟩ Used in sections 1, 10, 21, and 30.
⟨ Main program 4 ⟩ Used in section 1.
⟨ Other local variables of *main* 6 ⟩ Used in section 4.
⟨ Private variables for this source file 16, 36 ⟩ Used in sections 10 and 30.
⟨ Public # **define** statements to be exported 29 ⟩ Used in sections 21 and 26.
⟨ `getch.h` 34 ⟩
⟨ `getop.h` 26 ⟩
⟨ `stack.h` 14 ⟩

KRCWSAMP