Subsections

Looping and Iteration

This chapter will look at C's mechanisms for controlling looping and iteration. Even though some of these mechanisms may look familiar and indeed will operate in standard fashion most of the time. **NOTE:** some non-standard features are available.

The `for` statement

The C `for` statement has the following form:

```
  for  (expression1; 2; expression3)
                                 statement;
                                 or {block of statements}
```

expression1 initialises; *expression2* is the terminate test; *expression3* is the modifier (which may be more than just simple increment);

NOTE: C basically treats `for` statements as `while` type loops

For example:

```
  int x;

                 main()
                                {
                               for  (x=3;x>0;x-)
                                                {
                                                printf("x=%d\n",x);

                                                }
                                }
```

...outputs:

```
  x=3
                x=2
                x=1
```

```
...to the screen

All the following are legal for statements in C. The practical application of
such statements is not important here, we are just trying to illustrate
peculiar features of C for that may be useful:-

  for (x=0;((x>3) && (x<9)); x++)

                 for (x=0,y=4;((x>3) && (y<9)); x++,y+=2)

                 for (x=0,y=4,z=4000;z; z/=10)

The second example shows that multiple expressions can be separated a ,.
```

In the third example the loop will continue to iterate until z becomes 0;

The `while` statement

The `while` statement is similar to those used in other languages although more can be done with the `expression` statement -- a standard feature of C.

The `while` has the form:

```
  while (expression)
                                  statement
```

For example:

```
  int x=3;

                 main()
                                  { while (x>0)
                                                  { printf("x=%d\n",x);

                                                                  x-;
                                                  }
                                  }
```

...outputs:

```
  x=3
                x=2
                x=1
```

...to the screen.

Because the while loop can accept expressions, not just conditions, the following are all legal:-

```
  while (x-);
                while (x=x+1);
                while (x+=5);
```

Using this type of expression, only when the result of x-, x=x+1, or x+=5, evaluates to 0 will the while condition fail and the loop be exited.

We can go further still and perform complete operations within the while ***expression***:

```
  while (i++ < 10);

                while ( (ch = getchar()) != `q')
                  putchar(ch);
```

The first example counts i up to 10.

The second example uses C standard library functions (See Chapter 18) getchar() - reads a character from the keyboard - and putchar() - writes a given char to screen. The while loop will proceed to read from the keyboard and echo characters to the screen until a 'q' character is read. **NOTE:** This type of operation is used a lot in C and not just with character reading!! (See Exercises).

The `do-while` statement

C's `do-while` statement has the form:

```
  do
                                statement;
                                while (expression);
```

It is similar to PASCAL's `repeat` ... `until` except `do while` ***expression*** is true.

For example:

```
  int x=3;

                main()
                                { do {
                                                printf("x=%d\n",x-);

                                                }
                                while (x>0);
                                }
```

..outputs:-

```
  x=3
                x=2
                x=1
```

NOTE: The postfix x- operator which uses the current value of x while printing and ***then*** decrements x.

break and continue

C provides two commands to control how we loop:

- `break` -- exit form loop or switch.
- `continue` -- skip 1 iteration of loop.

Consider the following example where we read in integer values and process them according to the following conditions. If the value we have read is negative, we wish to print an error message and abandon the loop. If the value read is great than 100, we wish to ignore it and continue to the next value in the data. If the value is zero, we wish to terminate the loop.

```
   while (scanf( ``%d'', &value ) == 1 && value != 0) {

                                if (value < 0) {
                                                printf(``Illegal value\n'');

                                                break;
                                                /* Abandon the loop */
                                }

                                if (value > 100) {
                                                printf(``Invalid value\n'');

                                                continue;
                                                /* Skip to start loop again */
                                }

                                /* Process the value read */
                                  /* guaranteed between 1 and 100 */
                                     ....;

                                ....;
                } /* end while value != 0 */
```

Exercises

Exercise 12327

Write a program to read in 10 numbers and compute the average, maximum and minimum values.

Exercise 12328

Write a program to read in numbers until the number -999 is encountered. The sum of all number read until this point should be printed out.

Exercise 12329

Write a program which will read an integer value for a base, then read a positive integer written to that base and print its value.

Read the second integer a character at a time; skip over any leading non-valid (i.e. not a digit between zero and ``base-1") characters, then read valid characters until an invalid one is encountered.

```
      Input       Output
   ==========     ======
   10    1234      1234
    8      77        63   (the value of 77 in base 8, octal)
    2    1111        15   (the value of 1111 in base 2, binary)
```

The base will be less than or equal to 10.

Exercise 12330

Read in three values representing respectively

a capital sum (integer number of pence),

a rate of interest in percent (float),

and a number of years (integer).

Compute the values of the capital sum with compound interest added over the given period of years. Each year's interest is calculated as

interest = capital * interest_rate / 100;

and is added to the capital sum by

capital += interest;

Print out money values as pounds (pence / 100.0) accurate to two decimal places.

Print out a floating value for the value with compound interest for each year up to the end of the period.

Print output year by year in a form such as:

```
Original sum 30000.00 at  12.5 percent for 20 years

Year Interest  Sum
----+-------+--------
   1  3750.00 33750.00
   2  4218.75 37968.75
   3  4746.09 42714.84
   4  5339.35 48054.19
   5  6006.77 54060.96
   6  6757.62 60818.58
```

```
 7  7602.32 68420.90
 8  8552.61 76973.51
 9  9621.68 86595.19
10 10824.39 97419.58
```

Exercise 12331

Read a positive integer value, and compute the following sequence: If the number is even, halve it; if it's odd, multiply by 3 and add 1. Repeat this process until the value is 1, printing out each value. Finally print out how many of these operations you performed.

Typical output might be:

```
 Inital value is 9
 Next value is  28
 Next value is  14
 Next value is   7
 Next value is  22
 Next value is  11
 Next value is  34
 Next value is  17
 Next value is  52
 Next value is  26
 Next value is  13
 Next value is  40
 Next value is  20
 Next value is  10
 Next value is   5
 Next value is  16
 Next value is   8
 Next value is   4
 Next value is   2
 Final value 1, number of steps 19
```

If the input value is less than 1, print a message containing the word

```
       Error
```

and perform an

```
        exit( 0 );
```

Exercise 12332

Write a program to count the vowels and letters in free text given as standard input. Read text a character at a time until you encounter end-of-data.

Then print out the number of occurrences of each of the vowels a, e, i, o and u in the text, the total number of letters, and each of the vowels as an integer percentage of the letter total.

Suggested output format is:

```
        Numbers of characters:
        a   3 ; e   2 ; i   0 ; o   1 ; u   0 ; rest  17
        Percentages of total:
        a  13%; e   8%; i   0%; o   4%; u   0%; rest  73%
```

Read characters to end of data using a construct such as

```
        char ch;
        while(
            ( ch = getchar() ) >= 0
        ) {
            /* ch is the next character */    ....
        }
```

to read characters one at a time using `getchar()` until a negative value is returned.

Exercise 12333

Read a file of English text, and print it out one word per line, all punctuation and non-alpha characters being omitted.

For end-of-data, the program loop should read until "getchar" delivers a value <= 0. When typing input, end the data by typing the end-of-file character, usually control-D. When reading from a file, "getchar" will deliver a negative value when it encounters the end of the file.

Typical output might be

```
Read
a
file
of
English
text
and
print
it
out
one
```

etc.

Dave Marshall
1/5/1999