CHAPTER 6

STATEMENTS

brkwd.c

(Figure 6.1 – Page 126)

```
/*
 * Break input into words (letter followed by letters or digits).
 */
#include <stdio.h>
#include <ctype.h>

#define YES  1
#define NO   0

int main()
{
  int c;                        /* next input character */
  int word;                     /* flag: are we dealing with a word? */

  word = NO;
  while ((c = getchar()) != EOF)
    if (isalpha(c) || (word && isdigit(c)))      /* write letter */
      putchar(c), word = YES;
    else
      if (word)                 /* write newline if we were in word */
        putchar('\n'), word = NO;
  return 0;
}
```

Sample I/O – mtime

```
Enter military time (as xx:xx): 21:15
Standard time: 9:15
Enter military time (as xx:xx): 13:05
Standard time: 1:05
Enter military time (as xx:xx): 12:05
Standard time: 12:05
Enter military time (as xx:xx): 0:05
Standard time: 12:05
```

mtime.c

(Figure 6.2 – Page 128)

```c
/*
 * Convert military time to standard time.  No error checking.
 */
#include <stdio.h>

int main()
{
  unsigned int mhour;            /* input: military hour */
  unsigned int min;              /* input: military minutes */
  unsigned int stdhour;          /* standard hour */

  while (printf("Enter military time (as xx:xx): "),
         scanf("%u:%u", &mhour, &min) == 2)
  {
    stdhour = mhour;                  /* assume AM */
    if (mhour <= 12)
    {
      if (mhour == 0)
        stdhour = 12;                 /* midnight */
    }
    else
      stdhour = mhour - 12;           /* PM */
    printf("Standard time: %u:%02u\n", stdhour, min);
  }
  return 0;
}
```

Sample Output – countem

```
Total          1766

spaces          576  32.6%
letters         905  51.2%
digits           21   1.2%
puncts          264  14.9%
others            0   0.0%
```

countem.c

(Figure 6.3 – Page 130)

```
/*
 * Print counts of various types of characters appearing in the input.
 */
#include <stdio.h>
#include <ctype.h>

#define  CNT_WIDTH  11   /* field width for displaying counts */

int main()
{
  double        pct(unsigned long count, unsigned long total);
  int           c;                        /* next input character */
  unsigned long spaces;                   /* white space characters */
  unsigned long letters;                  /* a-z, A-Z */
  unsigned long digits;                   /* 0-9 */
  unsigned long puncts;                   /* punctuation characters */
  unsigned long others;                   /* anything else */
  unsigned long t;                        /* total characters */

  spaces = letters = digits = puncts = others = 0;
  while ((c = getchar()) != EOF)
    if (isspace(c))
      spaces++;                           /* whitespace */
    else if (isalpha(c))
      letters++;                          /* upper- or lowercase letter */
    else if (isdigit(c))
      digits++;                           /* digit */
    else if (ispunct(c))
      puncts++;                           /* punctuation */
    else
      others++;
```

countem.c

(Figure 6.3 – Page 130)

```
  t = spaces + letters + digits + puncts + others;
  printf("Total   %*lu\n\n", CNT_WIDTH, t);
  if (t != 0)
  {
    printf("spaces  %*lu %5.1f%%\n", CNT_WIDTH, spaces, pct(spaces, t));
    printf("letters %*lu %5.1f%%\n", CNT_WIDTH, letters, pct(letters, t));
    printf("digits  %*lu %5.1f%%\n", CNT_WIDTH, digits, pct(digits, t));
    printf("puncts  %*lu %5.1f%%\n", CNT_WIDTH, puncts, pct(puncts, t));
    printf("others  %*lu %5.1f%%\n", CNT_WIDTH, others, pct(others, t));
  }
  return 0;
}

double pct(unsigned long count, unsigned long total)
{
  return count * 100.0 / total;        /* compute/return percentage */
}
```

Sample I/O – calc

```
35.6+23.9
59.500000
67/69
0.971014
87.12-56.11
31.010000
10*15
150.000000
1/0
Warning: division by zero.
0.000000
```

calc.c

(Figure 6.4 – Page 133)

```
/*
 * Simple calculator program.
 */
#include <stdio.h>

int main()
{
  int    vals;                        /* scanf return value */
  double op1;                         /* the operand on the right */
  double op2;                         /* the operand on the left */
  char   operator;                    /* operator */
  double result;                      /* result */
```

calc.c

(Figure 6.4 – Page 133)

```
  while ((vals = scanf("%lf%c%lf", &op1, &operator, &op2)) == 3)
  {
    switch(operator)
    {
      case '+':
        result = op1 + op2;
        break;
      case '-':
        result = op1 - op2;
        break;
      case '*':
        result = op1 * op2;
        break;
      case '/':
        if (op2 != 0.0)
          result = op1 / op2;
        else
        {
          printf("Warning: division by zero.\n");
          result = 0.0;
        }
        break;
      default:
        printf("Unknown operator: %c\n", operator);
        result = 0.0;             /* default result */
        break;
    }
    printf("%f\n", result);
  }

  if (vals != EOF)
    printf("Error detected in the input: program terminated.\n");

  return vals == EOF;
}
```

calc2.c

(Not In Text)

```
/*
 * Simple calculator program writting using an if rather than switch.
 */
#include <stdio.h>

int main()
{
  double calculate(char op, double op1, double op2);
  int    vals;                           /* scanf return value */
  double op1;                            /* the operand on the right */
  double op2;                            /* the operand on the left */
  char   operator;                       /* operator */

  while ((vals = scanf("%lf%c%lf", &op1, &operator, &op2)) == 3)
    printf("%f\n", calculate(operator, op1, op2));
  if (vals != EOF)
    printf("Error detected in the input: program terminated.\n");
  return vals == EOF;
}
```

result.c

(Not In Text)

```c
/*
 * Function to perform an operation on two values and return result.
 */
#include <stdio.h>

double calculate(char operator, double op1, double op2)
{
  double result;

  if (operator == '+')
    result = op1 + op2;
  else if (operator == '-')
    result = op1 - op2;
  else if (operator == '*')
    result = op1 * op2;
  else if (operator == '/')
    if (op2 != 0.0)
      result = op1 / op2;
    else
    {
      printf("Warning: division by zero.\n");
      result = 0.0;
    }
  else
  {
    printf("Unknown operator: %c\n", operator);
    result = 0.0;             /* default result */
  }
  return result;
}
```

Sample Output – countpct

```
Quotes              41
Brackets            28
Delimiters          51
Others            1171
```

countpct.c

(Figure 6.5 – Page 135)

```
/*
 * Count various types of punctuation characters.
 */
#include <stdio.h>

#define  CNT_WIDTH  11   /* field width for displaying counts */

int main()
{
  int      c;                     /* next input character */
  unsigned long quotes;           /* counts quotation marks */
  unsigned long brackets;         /* counts parens/brackets */
  unsigned long delims;           /* counts punctuation */
  unsigned long others;           /* counts everything else */
```

countpct.c

(Figure 6.5 – Page 135)

```
  quotes = brackets = delims = others = 0L;
  while ((c = getchar()) != EOF)
    switch(c)
    {
      case '\'':
      case '"':
      case '`':
        quotes++;                   /* single and double quotes */
        break;
      case '(':
      case ')':
      case '{':
      case '}':
      case '[':
      case ']':
        brackets++;                 /* punctuation brackets */
        break;
      case ',':
      case '.':
      case ';':
      case ':':
      case '!':
      case '?':
        delims++;                   /* nonpunctuation chars */
        break;
      default:
        others++;
    }
  printf("Quotes     %*lu\n", CNT_WIDTH, quotes);
  printf("Brackets   %*lu\n", CNT_WIDTH, brackets);
  printf("Delimiters %*lu\n", CNT_WIDTH, delims);
  printf("Others     %*lu\n", CNT_WIDTH, others);
  return 0;
}
```

yesorno.c

(Figure 6.6 – Page 137)

```
/*
 * Print prompt and then get a yes or no answer using "yesorno".
 */
#include <stdio.h>
#include <ctype.h>

#define YES 1
#define NO  0
#define BAD (-1)

int main()
{
  int yesorno(void);

  printf("Enter a YES or NO answer: ");
  switch (yesorno())
  {
    case YES: printf("YES!\n");
              break;
    case NO:  printf("NO!\n");
              break;
    default:  printf("EOF!\n");
  }
  return 0;
}
```

yesorno.c

(Figure 6.6 – Page 137)

```
int yesorno(void)
{
  int c;                              /* holds input character */
  int answer;                         /* holds YES or NO answer */

  do
  {
    answer = BAD;
    switch (c = tolower(getchar()))
    {
      case EOF:                       /* nothing to do */
                 break;
      case 'y':  answer = YES;
                 break;
      case 'n':  answer = NO;
                 break;
      default:   printf("Please answer with a YES or NO: ");
    }
    while (c != '\n' && c != EOF)
        c = getchar();                /* skip rest of line */
  }
  while (answer == BAD && c != EOF);
  return answer;
}
```

yesorno2.c

(Figure 6.7 – Page 138)

```
/*
 * Print prompt and then get a yes or no answer using "yesorno".
 */
#include <stdio.h>
#include <ctype.h>

#define YES 1
#define NO  0

int main()
{
  int yesorno(void);

  printf("Enter a YES or NO answer: ");
  (yesorno() == YES) ? printf("YES!\n") : printf("NO!\n");
  return 0;
}

int yesorno(void)
{
  int answer;                         /* holds input character */
  int c;                              /* for skipping characters */

  for (;;)
  {
    if ((answer = tolower(getchar())) == EOF)
      return NO;                      /* EOF is NO */
    c = answer;
    while (c != '\n' && c != EOF)
      c = getchar();
    if (answer == 'y' || answer == 'n')
      return (answer == 'y') ? YES : NO;
    printf("Please answer with a YES or NO: ");
  }
}
```

power.c

(Figure 6.8 – Page 139)

```c
/*
 * Compute powers, more concise version.
 */
double power(double x, int exp)
{
  double p;

  p = 1.0;
  if (exp > 0)                                      /* positive exponent */
    for (; exp > 0; exp--)
      p *= x;
  else                                              /* negative exponent */
    for (; exp < 0; exp++)
      p /= x;
  return p;
}
```

lineno.c

(Figure 6.9 – Page 139)

```
/*
 * Copy input to output, giving each line a number.
 */
#include <stdio.h>

int main()
{
  int  c;                          /* current and */
  int  lastch;                     /*    previous characters */
  long lineno;                     /* lines printed so far */

  lineno = 0;
  for (lastch = '\n'; (c = getchar()) != EOF; putchar(lastch = c))
    if (lastch == '\n')            /* hit end of line */
      printf("%8li ", ++lineno);
  return 0;
}
```

avg.c

(Figure 6.10 – Page 140)

```c
/*
 * Compute average of its input values with no error checking.
 */
#include <stdio.h>

int main()
{
  int    next;                         /* next input value */
  long   sum;                          /* running total */
  int    n;                            /* number of input values */
  double avg;                          /* average of input values */

  for (sum=n=0; scanf("%i", &next) == 1; sum += next, n++)
    ;
  avg = (n == 0) ? 0.0 : (double) sum / n;
  printf("Average of %i values is %f.\n", n, avg);
  return 0;
}
```

usepower.c

(Figure 6.11 – Page 142)

```
/*
 * Using our own function to compute exponents.
 */
#include <stdio.h>

int main()
{
  double power(double base, int exp);
  double x;                            /* user-supplied base */
  int    y;                            /* user-supplied exponent */
  int    r;                            /* scanf return value */

  while (printf("Enter x,y: "), (r = scanf("%lf %i", &x, &y)) != EOF)
  {
    if (r != 2)
    {
      printf("Error: Didn't read two input values.\n");
      break;
    }
    printf("%g^%i = %g\n", x, y, power(x, y));
  }
  return r == EOF;
}
```

mtime2.c

(Figure 6.12 – Page 143)

```
/*
 * Convert military time to standard time, now with error checking.
 */
#include <stdio.h>

int main()
{
  unsigned int mhour;            /* input: military hour */
  unsigned int min;              /* input: military minutes */
  unsigned int stdhour;          /* standard hour */

  while (printf("Enter military time (as xx:xx): "),
         scanf("%u:%u", &mhour, &min) == 2)
  {
    if (mhour >= 24 || min >= 60)
    {
      printf("Error: military time is from 0:00 to 23:59\n");
      continue;
    }
    stdhour = mhour;                  /* assume AM */
    if (mhour <= 12)
    {
      if (mhour == 0)
        stdhour = 12;                 /* midnight */
    }
    else
      stdhour = mhour - 12;           /* PM */
    printf("Standard time: %u:%02u\n", stdhour, min);
  }
  return 0;
}
```

Sample I/O – octdump

```
this is a test
_ _ _ _ _
000000  164 t   150 h   151 i   163 s
000004  040     151 i   163 s   011 \t
000008  011 \t  141 a   040     164 t
000012  145 e   163 s   164 t   012 \n
```

octdump.c

(Figure 6.13 – Page 146)

```
/*
 * Dump input in octal (and character if printable).
 */
#include <stdio.h>
#include <ctype.h>

#define MAXCHARS  4                       /* max # of chars on a line */

int main()
{
  void          print_char(int c);
  int           c;                        /* next character */
  int           linecnt;                  /* # of chars on line */
  unsigned long inpcnt;                   /* # of chars in input */

  for (linecnt = 0, inpcnt = 0; (c = getchar()) != EOF; inpcnt++)
  {
    if (linecnt++ == 0)
      printf("%06li  ", inpcnt);
    else
      printf("  ");
    print_char(c);
    if (linecnt == MAXCHARS)
    {
      putchar('\n');
      linecnt = 0;
    }
  }
  if (linecnt > 0)                       /* finish up last line */
    putchar('\n');
  return 0;
}
```

octdump.c

(Figure 6.13 – Page 146)

```
void print_char(int c)
{
  printf("%03o ", c);               /* write octal code for character */
  switch (c)                        /* write escape sequence or char */
  {
    case '\a':  printf("\\a");  break;
    case '\b':  printf("\\b");  break;
    case '\f':  printf("\\f");  break;
    case '\n':  printf("\\n");  break;
    case '\r':  printf("\\r");  break;
    case '\t':  printf("\\t");  break;
    case '\v':  printf("\\v");  break;
    default:    printf("%c ", isprint(c) ? c : ' ');
  }
}
```

REMINDERS

- Watch out for accidentally writing syntactically legal expression statements that accomplish nothing.

- Don't forget to place a **`break`** at the end of each case to prevent accidentally falling through.

- Remember that we need the semicolons in the **`for`** even when we leave out the expressions.

- Avoid placing a null statement on the same line as a **`for`** or a **`while`**.

- Avoid **`continue`** except when checking for error conditions at the top of a loop.

- Avoid **`goto`**s whenever possible.

CHAPTER SUMMARY

- C provides expression statements (any expression followed by a colon) and compound statements (any set of statements surrounded by braces).

- C provides an **`if`** statement for doing different actions if an expression is true or false.

- C provides a **`switch`** statement for selecting an action based on different values of a single expression.

- C provides three looping constants: **`while`**, which tests before the loop body; **`for`**, which also lets us specify actions to perform before we enter the loop and each time we complete the loop body; and **`do-while`**, which tests after the loop body.

- C provides several control-flow altering statements: **`break`**, **`continue`**, and **`goto`**. **`break`** exits the nearest enclosing loop or **`switch`**, **`continue`** skips the rest of a loop, and **`goto`** transfers control to an arbitrary label. These statements should be avoided if at all possible.

- Finally, C provides a **`return`** statement, which generally should appear only at the end of a function.