CHAPTER 4

CHARACTERS

4.1 REPRESENTING AND STORING CHARACTERS
- Character Constants
- Formatted Reading and Writing of Characters

4.2 CHARACTERS VERSUS INTEGERS
- Signed versus Unsigned Characters
- Extended Character Sets

4.3 CHARACTER INPUT AND OUTPUT

4.4 CHARACTER TESTING FUNCTIONS

4.5 CONVERTING CHARACTERS INTO NUMBERS

4.6 CASE STUDY—A MORE GENERAL BASE CONVERTER

Sample I/O – checks

```
Starting balance? 100.00
Type and amount? c 10.50
Check for 10.50
Updated balance is: 89.50
Type and amount? d 345.10
Deposit of 345.10
Updated balance is: 434.60
Type and amount? c 34.00
Check for 34.00
Updated balance is: 400.60
Type and amount? c 19.95
Check for 19.95
Updated balance is: 380.65
Type and amount? d 100.00
Deposit of 100.00
Updated balance is: 480.65
Type and amount?
```

checks.c

(Not In Text)

```
/*
 * A program to balance a checkbook.
 */
#include <stdio.h>

main()
{
  char   trans_type;      /* transaction type */
  double amount;          /* transaction amount */
  double balance;         /* transaction balance */
  int    n;               /* number of values read in */

  printf("Starting balance? ");
  scanf("%lf", &balance);
  printf("Type and amount? ", trans_type, amount);
  while ((n=scanf(" %c %lf", &trans_type, &amount)) == 2)
  {
    if (trans_type == 'c')
    {
      printf("Check for %.2f\n", amount);
      balance = balance - amount;
    }
    else
      if (trans_type == 'd')
      {
        printf("Deposit of %.2f\n", amount);
        balance = balance + amount;
      }
      else
        printf("%c: unknown transaction type\n", trans_type);
    printf("Updated balance is: %.2f\n", balance);
    printf("Type and amount? ", trans_type, amount);
  }
  if (n == 2)
    printf("Fatal error: Expected two numeric input values.\n");
  return 0;
}
```

BACKSLASH ESCAPE SEQUENCES

CODE	CHARACTER	ASCII HEX VALUE
`\0`	null character	0
`\a`	audible alert	0x07
`\b`	backspace	0x08
`\f`	form feed	0x0C
`\n`	newline	0x0A
`\r`	carriage return	0x0D
`\t`	horizontal tab	0x09
`\v`	vertical tab	0x0B
`\'`	single quote	0x27
`\"`	double quote	0x22
`\\`	backslash	0x5C
`\?`	question mark	0x3F
`\`*ddd*	up to 3-digit octal value	
`\x`*ddd*	up to 3-digit hexadecimal value	

TRIGRAPHS

<table>
<tr><th>TRIGRAPH</th><th>CHARACTER</th></tr>
<tr><td>??=</td><td>=</td></tr>
<tr><td>??/</td><td>\</td></tr>
<tr><td>??′</td><td>^</td></tr>
<tr><td>??{</td><td>[</td></tr>
<tr><td>??}</td><td>]</td></tr>
<tr><td>??!</td><td>|</td></tr>
<tr><td>??<</td><td>{</td></tr>
<tr><td>??></td><td>}</td></tr>
<tr><td>??−</td><td>~</td></tr>
</table>

wakeup.c

(Figure 4.1 – Page 70)

```
/*
 * Example program using the various character codes.
 */
#include <stdio.h>

#define MAXBEEP 10

int main()
{
  int  i;                                   /* index to write beeps */
  char c;                                   /* character we read */

  for (i = 0; i < MAXBEEP; i = i + 1)       /* lots of beeps */
    printf("Wake up!\a\n");
  printf("Are you awake yet\?\?\? ");       /* query user */
  if (scanf("%c", &c) == 1)                 /* obtain response */
  {
     printf("You responded with a '%c'\n", c);
     if (c == 'y')
       printf("We're glad you're awake!\n");
     else
       printf("Well, we tried!\n");
  }
  else                                      /* couldn't read character */
    printf("Huh\x3F\n");
  return 0;
}
```

RANGES OF CHARACTER TYPES

CONSTANT	VALUE	DESCRIPTION
CHAR_BIT	**8**	maximum number of bits in a byte
CHAR_MAX	**127**	maximum value of a **char**
CHAR_MIN	**0**	minimum value of a **char**
SCHAR_MAX	**+127**	maximum value of a **signed char**
SCHAR_MIN	**−127**	minimum value of a **signed char**
UCHAR_MAX	**255**	maximum value of an **unsigned char**

charset.c

(Figure 4.2 – Page 73)

```
/*
 * Print character set in decimal, octal, and character (ASCII only).
 */
#include <stdio.h>
#include <limits.h>

int main()
{
  unsigned char c;

  for (c = 0; c <= SCHAR_MAX; c = c + 1)
    printf("%4i\t%4o\t%c\n", c, c, c);
  return 0;
}
```

Sample I/O – display

```
Don't you just love
Don't you just love
programming in C?
programming in C?
We thought so!
We thought so!
```

display.c

(Figure 4.3 – Page 74)

```
/*
 * Copy the input to the output.
 */
#include <stdio.h>

int main()
{
  int c;                                     /* next character */

  while ((c = getchar()) != EOF)
    putchar(c);
  return 0;
}
```

More Sample I/O – display

```
Don’t you just love
programming in C?
We thought so!
_ _ _ _ _
```

`Don’t you just love`
`programming in C?`
`We thought so!`

cntchars.c

(Not In Text)

```
/*
 * Count the number of characters in the input.
 */
#include <stdio.h>

int main()
{
  int c;                                     /* next character */
  long charcnt;                              /* # of characters */

  charcnt = 0;
  while ((c = getchar()) != EOF)
    charcnt = charcnt + 1;
  printf("%li characters.\n", charcnt);
  return 0;
}
```

cntlines.c

(Not In Text)

```
/*
 * Count the number of characters in the input.
 */
#include <stdio.h>

int main()
{
  int c;                                    /* next character */
  long charcnt;                             /* # of characters */
  long linecnt;                             /* # of lines */

  charcnt = 0;
  linecnt = 0;
  while ((c = getchar()) != EOF)
  {
    charcnt = charcnt + 1;
    if (c == '\n')
      linecnt = linecnt + 1;
  }
  printf("%li characters.\n", charcnt);
  printf("%li lines.\n", linecnt);
  return 0;
}
```

Sample Output – lineno

```
/*
 * 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;
  lastch = '\n';
  while ((c = getchar()) != EOF)
  {
    if (lastch == '\n')
    {                               /* hit end of line */
      lineno = lineno + 1;
      printf("%8li ", lineno);
    }
    putchar(c);
    lastch = c;
  }
  return 0;
}
```

lineno.c

(Figure 4.5 – Page 75)

```
/*
 * 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;
  lastch = '\n';
  while ((c = getchar()) != EOF)
  {
    if (lastch == '\n')
    {                              /* hit end of line */
      lineno = lineno + 1;
      printf("%8li ", lineno);
    }
    putchar(c);
    lastch = c;
  }
  return 0;
}
```

CHARACTER TESTING FUNCTIONS

FUNCTION	CHARACTER TYPE
`isalnum`	a letter or digit (a–z, A–Z, 0–9)
`isalpha`	a letter (a–z, A–Z)
`iscntrl`	a control or delete character (in ASCII, 0x00–0x1F and 0x7F)
`isdigit`	a digit (0–9)
`isgraph`	printable, not including space (in ASCII, 0x21–0x7E)
`islower`	a lowercase letter (a–z)
`isprint`	printable, including space (in ASCII, 0x20–0x7E)
`ispunct`	punctuation (**`isprint`** and **`!isalnum`** and not space)
`isspace`	a whitespace, **`\t`**, **`\v`**, **`\f`**, **`\r`**, or **`\n`**
`isupper`	an uppercase letter (A–Z)
`isxdigit`	a hexadecimal digit (0–9, a–f, A–F)

Sample I/O – brkwd

```
this is 1 short line!
_ _ _ _ _
this
is
short
line
```

brkwd.c

(Figure 4.7 – Page 78)

```
/*
 * Break input into words (groups of letters).
 */
#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) != 0)
    {                           /* it's a letter */
      putchar(c);               /* write it and note we're in word */
      word = YES;
    }
    else                        /* it's not a letter */
      if (word == YES)          /* write a newline if we were in word */
      {
        putchar('\n');
        word = NO;
      }
  return 0;
}
```

yesorno.c

(Figure 4.8 – Page 79)

```
/*
 * Print a 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: ");
  if (yesorno() == YES)
    printf("That was a YES!\n");
  else
    printf("That was a NO!\n");
  return 0;
}

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

  c = getchar();
  if (tolower(c) == 'y')
    answer = YES;                     /* YES for 'Y' or 'y' */
  else
    answer = NO;                      /* NO for anything else */
  if (c != EOF)
    while (c != '\n')                 /* skip other characters on line */
      c = getchar();
  return answer;
}
```

Sample I/O – getdigmn

```
Select an entry: 8
Enter a value between 0 and 5: what?
Enter a value between 0 and 5: 4
You chose 4!
```

getdigmn.c

(Figure 4.9 – Page 81)

```
/*
 * Program illustrating converting a character to a number.
 */
#include <stdio.h>

#define MAX_DIGIT   5        /* largest possible input value */

int main()
{
  int get_base10_dig(int max);
  int digvalue;              /* value of user-entered digit */

  printf("Select an entry: ");
  if ((digvalue = get_base10_dig(MAX_DIGIT)) != EOF)
    printf("You chose %i!\n", digvalue);
  return 0;
}
```

getdig.c

(Figure 4.10 – Page 82)

```
/*
 * Functions to read a character and convert it to a number.
 *   get_base10_dig - get a digit between 0 and "max" from user.
 *   get_first - read an input line, returning its first character.
 */
#include <stdio.h>
#include <ctype.h>

int get_base10_dig(int max)
{
  int get_first(void);
  int c;                     /* character that might be a digit */
  int value;                 /* value of character as number */

  while ((c = get_first()) != EOF)
  {
    if (isdigit(c) != 0)
      if ((value = c - '0') <= max)
        return value;        /* return value of character */
    printf("Enter a value between 0 and %i: ", max);
  }
  return c;
}

int get_first(void)
{
  int first;                 /* first character on line */
  int junk;                  /* holds other characters */

  if ((first = getchar()) != EOF)
  {
    junk = first;            /* skip remaining characters */
    while (junk != '\n')
      junk = getchar();
  }
  return first;
}
```

getnum.c

(Figure 4.11 – Page 83)

```
/*
 * Read a line's worth of characters and convert them into a number,
 * one character at a time.  It only works for positive numbers.  It's
 * an error if the line contains anything other than digits.
 */
#include <stdio.h>
#include <ctype.h>

int main()
{
  long get_base10_num(void);
  long val;                             /* input value read */

  printf("Enter a number: ");
  if ((val = get_base10_num()) == -1)
    printf("You didn't enter a number.\n");
  else
    printf("You entered %li\n", val);
  return 0;
}
```

getnum.c

(Figure 4.11 – Page 83)

```
long get_base10_num(void)
{
  int  c;                             /* next input character */
  long sum;                           /* running total */

  if ((c = getchar()) == EOF)         /* EOF is error */
    sum = -1;
  else
  {
    if (isdigit(c) == 0)              /* not digit, so its an error */
      sum = -1;
    else
    {                                 /* in number, convert it */
      sum = 0;
      while (isdigit(c) != 0)
      {
        sum = 10L * sum + (c - '0');  /* convert value */
        c = getchar();
      }
      if (c != '\n')                  /* not ending with a newline? */
        sum = -1;                     /* then it's an error */
    }
    while (c != '\n')                 /* skip rest of line */
      c = getchar();
  }

  return sum;
}
```

Sample I/O – convert

```
Enter initial base: 16
Enter target base: 2
Enter number to convert: FFFF
FFFF in base 16 is 1111111111111111 in base 2.
Enter another base 16 number to convert: A8B4
A8B4 in base 16 is 1010100010110100 in base 2.
Enter another base 16 number to convert: 8EF5
8EF5 in base 16 is 1000111011110101 in base 2.
Enter another base 16 number to convert:
```

convert.c

(Figure 4.12 – Page 85)

```
/*
 * Convert from any base to any other base.
 */
#include <stdio.h>

#define  MINBASE   2                /* smallest base */
#define  MAXBASE  36                /* largest base */

int main()
{
  void display_value_in_base(long v, int b);
  long get_baseN_num(int b);
  int  get_base(int minbase, int maxbase);
  int  base;                        /* initial base */
  int  newbase;                     /* final base*/
  long num;                         /* initial value, in base 10 */

  printf("Enter initial base: ");   /* grab initial base (in base 10) */
  if ((base = get_base(MINBASE, MAXBASE)) != -1)
  {
    printf("Enter target base: ");             /* grab target base */
    if ((newbase = get_base(MINBASE, MAXBASE)) != -1)
    {
      printf("Enter number to convert: ");     /* grab number */
      while ((num = get_baseN_num(base)) != -1)
      {
        display_value_in_base(num, base);
        printf(" in base %i is ", base);
        display_value_in_base(num, newbase);
        printf(" in base %i.\n", newbase);
        printf("Enter another base %i number to convert: ", base);
      }
      putchar('\n');
    }
  }
  return 0;
}
```

getbase.c

(Figure 4.13 – Page 86)

```
/*
 * Read in the base using get_baseN_num.
 */
#include <stdio.h>

int get_base(int minbase, int maxbase)
{
  long get_baseN_num(int b);
  long base;

  if ((base = get_baseN_num(10)) == -1)
    printf("Error: nonnumeric base detected\n");
  else
  {
    if (base < minbase)            /* set to smallest if too small */
    {
      printf("Error: Smallest base is %i.\n", minbase);
      base = -1;
    }
    if (base > maxbase)            /* set to largest if too large */
    {
      printf("Error: Largest base is %i.\n", maxbase);
      base = -1;
    }
  }
  return (int) base;          /* cast tells compiler not to complain */
}
```

getbasen.c

(Figure 4.14 – Page 87)

```
/*
 * Read a base N number, one character at a time.
 */
#include <stdio.h>

long get_baseN_num(int base)
{
  int  todecimal(int c, int b);
  int  c;                         /* next input character */
  int  value;                     /* decimal value of next char */
  long sum;                       /* running total */

  if ((c = getchar()) == EOF)     /* EOF is error */
    sum = -1;
  else
  {
    if ((value = todecimal(c, base)) == -1)
      sum = -1;                   /* not a digit in desired base */
    else
    {                             /* in number, update it */
      sum = 0;
      while (value != -1)
      {                           /* valid next digit in base */
        sum = base * sum + value; /* update number */
        c = getchar();
        value = todecimal(c, base);
      }
      if (c != '\n')
        sum = -1;                 /* indicate error! */
    }
    while (c != '\n')             /* skip rest of line */
      c = getchar();
  }
  return sum;
}
```

convchar.c

(Figure 4.15 – Page 88)

```
/*
 * Functions to convert digits to and from characters (ASCII ONLY).
 *   todecimal - convert char to equivalent decimal number.
 *   tochar - convert decimal number to equivalent character.
 */
#include <ctype.h>

int todecimal(int c, int base)
{
  int value;

  value = -1;                      /* simplify later error handling */
  if (isdigit(c))
    value = c - '0';
  if (islower(c))
    value = c - 'a' + 10;
  if (isupper(c))
    value = c - 'A' + 10;
  if (value != -1)                 /* if we got a value */
    if (value >= base)             /* make sure it's legal for base */
      value = -1;
  return value;
}

int tochar(int value)
{
  int c;

  if (value < 10)
    c = '0' + value;
  else
    c = 'A' + value - 10;
  return c;
}
```

convfunc.c

(Figure 4.16 – Page 89)

```
/*
 * Display a value in a specified base between 2 and 10.
 */
#include <stdio.h>
#include <math.h>

void display_value_in_base(long v, int b)
{
  int tochar(int value);            /* addition since Chapter 3 */
  int  k;                           /* digits needed in result */
  long divisor;                     /* initially B^(# of digits - 1) */

  if (v == 0)                       /* zero is the same in any base */
    printf("0");
  else
  {
    k = floor(log10(v)/log10(b)) + 1;
    divisor = pow(b, k - 1);        /* first divisor: b^(k - 1) */

    /* Run through value, calculating and displaying the value of each
       of the digits in the new base (left to right) */

    while (divisor >= 1)
    {
      putchar(tochar((int)(v / divisor)));  /* changed from Chapter 3 */
      v = v % divisor;
      divisor = divisor / b;
    }
  }
}
```

REMINDERS

- Automatic conversions between characters and integers are safe only with legal integer representations in the machine's character set.

- Whether `char` is `signed` or `unsigned` is implementation-defined.

CHAPTER SUMMARY

- C provides a special integral type, **`char`**, for holding the integer representations of characters.

- C automatically converts back and forth between **`char`**s and **`int`**s.

- C provides character constants, a character enclosed in single quotes, as well as escape sequences to identify special characters.

- The standard library contains a pair of functions for doing I/O a character at a time: **`getchar`** reads the next character in the standard input, and **`putchar`** writes a character onto the standard output.

- The library also provides a set of character testing functions used to determine whether a particular character has certain properties, such as being a lower- or uppercase letter.

- Reading numerical data a character at a time and doing our own conversions allows us to improve on **`scanf`**'s error handling abilities.