CHAPTER 3

NUMBERS

3.1 IDENTIFIERS AND NAMING CONVENTIONS

3.2 INTEGERS
- Integer Constants
- Reading and Writing Integers

3.3 REALS

3.4 ARITHMETIC OPERATORS
- Signed Integer Arithmetic
- Unsigned Integer Arithmetic
- Floating Point Arithmetic
- Arithmetic Functions

3.5 TYPE CONVERSIONS
- Assignment Conversions
- Arithmetic Conversions
- Casts

3.6 CASE STUDY—A BASE CONVERSION PROGRAM

IDENTIFIER RULES

- Identifiers can only contain letters, digits, and the underscore.
- Identifiers must begin with a letter or underscore.
- Case is significant in identifiers.
- No limit to length of identifier.
- First 31 characters are significant on internal names (local variables).
- First 8 characters are significant on global names (functions used across files)
- Can’t use keywords as identifiers.

RESERVED WORDS

`auto`	`break`	`case`	`char`
`const`	`continue`	`default`	`do`
`double`	`else`	`enum`	`extern`
`float`	`for`	`goto`	`if`
`int`	`long`	`register`	`return`
`short`	`signed`	`sizeof`	`static`
`struct`	`switch`	`typedef`	`union`
`unsigned`	`void`	`volatile`	`while`

MINIMUM SIZE AND RANGE OF INTEGERS

INTEGER TYPE	BITS	RANGE
`short`	16	$-32,767$ to $+32,767$
`int`	16	$-32,767$ to $+32,767$
`long`	32	$-2,147,483,6487$ to $+2,147,483,647$
`unsigned short`	16	0 to $65,535$
`unsigned int`	16	0 to $65,535$
`unsigned long`	32	0 to $4,294,967,295$

SPECIAL INTEGER CONSTANTS

CONSTANT	VALUE	DESCRIPTION
`INT_MAX`	**`+32767`**	largest value of an **`int`**
`INT_MIN`	**`−32767`**	smallest value of an **`int`**
`LONG_MAX`	**`+2147483647L`**	largest value of a **`long`**
`LONG_MIN`	**`−2147483647L`**	smallest of a **`long`**
`SHRT_MAX`	**`+32767`**	largest value of a **`short`**
`SHRT_MIN`	**`−32767`**	smallest value of a **`short`**
`UINT_MAX`	**`65535U`**	largest value of an **`unsigned int`**
`ULONG_MAX`	**`4294967295UL`**	largest value of an **`unsigned long`**
`USHRT_MAX`	**`65535U`**	largest value of an **`unsigned short`**

EXAMPLE INTEGER CONSTANTS

CONSTANT VALUE	BITS IN AN `int`	CONSTANT TYPE
1000	16 or 32	`int`
100000	16	`long`
100000	32	`int`
2500000000	16 or 32	`unsigned long`
0x7FFF	16	`int`
0xFFFF	16 or 32	`unsigned int`
0x7FFFFFFF	16	`long`
0x7FFFFFFF	32	`int`
0xFFFFFFFF	16	`unsigned long`
0xFFFFFFFF	32	`unsigned int`

FORMATTING CODES FOR INTEGERS

TYPE	READING WITH **scanf**	PRINTING WITH **printf**
short	**%hd** or **%hi**	**%d** or **%i**
int	**%d** or **%i**	**%d** or **%i**
long	**%ld** or **%li**	**%ld** or **%li**
unsigned short	**%hu**	**%u**
unsigned int	**%u**	**%u**
unsigned long	**%lu**	**%lu**
octal **short**	**%ho**	**%o**
octal **int**	**%o**	**%o**
octal **long**	**%lo**	**%lo**
hex **short**	**%hx**	**%x**
hex **int**	**%x**	**%x**
hex **long**	**%lx**	**%lx**

SAMPLE OUTPUT – range

```
Smallest int:   -2147483648
Largest int:    2147483647
Smallest long:  -2147483648
Largest long:   2147483647
Smallest short: -32768
Largest short:  32767
```

range.c
(Not In Text)

```
/*
 * Program to print ranges of several integer types.
 */
#include <stdio.h>
#include <limits.h>

main()
{
  printf("Smallest int:   %i\n", INT_MIN);
  printf("Largest int:    %i\n", INT_MAX);
  printf("Smallest long:  %li\n", LONG_MIN);
  printf("Largest long:   %li\n", LONG_MAX);
  printf("Smallest short: %i\n", SHRT_MIN);
  printf("Largest short:  %i\n", SHRT_MAX);
  return 0;
}
```

Sample I/O – bases

```
Range of values: 1 24
Base 10   Base 8    Base 16
   1         1         1
   2         2         2
   3         3         3
   4         4         4
   5         5         5
   6         6         6
   7         7         7
   8        10         8
   9        11         9
  10        12         a
  11        13         b
  12        14         c
  13        15         d
  14        16         e
  15        17         f
  16        20        10
  17        21        11
  18        22        12
  19        23        13
  20        24        14
  21        25        15
  22        26        16
  23        27        17
  24        30        18
```

bases.c

(Not In Text)

```c
/*
 * Program to count in different bases.
 */
#include <stdio.h>

main()
{
  int first;        /* first value to print */
  int last;         /* last value to print */
  int next;         /* next value to print */

  printf("Range of values: ");
  scanf("%i %i", &first, &last);
  printf("Base 10   Base 8    Base 16\n");
  for (next = first; next <= last; next = next + 1)
    printf("%4i      %4o      %4x\n", next, next, next);
  return 0;
}
```

Sample Output – converr

```
2500000000 -1794967296
25000 25000
```

converr.c

(Figure 3.1 – Page 48)

```
/*
 * Write a large and a small unsigned value as signed.
 */
#include <stdio.h>

int main()
{
  unsigned long value;

  /* write a large value as both an unsigned long and a long */

  value = 2500000000ul;
  printf("%lu %li\n", value, value);

  /* write a small value as both an unsigned long and a long */

  value = 25000ul;
  printf("%lu %li\n", value, value);

  return 0;
}
```

REPRESENTATION OF REAL NUMBERS

0	1 ⋯ 8	9 ⋯ 31
SIGN	EXPONENT	FRACTION

EXAMPLE REAL NUMBERS

FLOATING POINT	'e' NOTATION	SCIENTIFIC NOTATION
`12.45`	`1.245e1`	1.245×10^{1}
`-211.0`	`-2.110e2`	-2.110×10^{2}
`0.0056`	`5.600e-3`	5.600×10^{-3}
`-0.000123`	`-1.230e-4`	-1.230×10^{-4}
`1000000.0`	`1e6`	1.000×10^{6}

MINIMUM RANGE OF REALS

CONSTANT	VALUE	DESCRIPTION
FLT_DIG	**6**	decimal digits of precision for **float**
FLT_EPSILON	**1E−5**	smallest *x* such that $1.0 + x \neq 1.0$
FLT_MAX	**1E+37**	largest **float**
FLT_MIN	**1E−37**	smallest **float**
FLT_MAX_10_EXP	**+37**	largest integer power of 10 in a **float**
FLT_MIN_10_EXP	**−37**	smallest integer power of 10 in a **float**
DBL_DIG	**10**	decimal digits of precision for **double**
DBL_EPSILON	**1E−9**	smallest x such that $1.0 + x \neq 1.0$
DBL_MAX	**1E37**	largest **double**
DBL_MIN	**1E−37**	smallest **double**
DBL_MAX_10_EXP	**+37**	largest integer power of 10 in a **double**
DBL_MIN_10_EXP	**−37**	smallest integer power of 10 in a **double**
LDBL_DIG	**10**	decimal digits of precision for **long double**
LDBL_EPSILON	**1E−9**	smallest x such that $1.0 + x \neq 1.0$
LDBL_MAX	**1E37**	largest **long double**
LDBL_MIN	**1E−37**	smallest **long double**
LDBL_MAX_10_EXP	**+37**	largest integer power of 10 in a **long double**
LDBL_MIN_10_EXP	**−37**	smallest integer power of 10 in a **long double**
FLT_RADIX	**2**	radix of exponent representation
FLT_ROUNDS	**0**	addition rounds toward nearest value (1), toward zero (0), toward $+\infty$ (2), toward $-\infty$ (3), or indeterminate (-1)
FLT_GUARD	**0**	guard digits are used (1) or not used (0) for multiplication
FLT_NORMALIZE	**0**	floating values must be normalized (1) or need not be (0)

Sample I/O – area

```
Enter radius: 34.5
Area in floating point notation: 3739.280592
Area in exponential notation:    3.739281e+03
Area in smallest notation:       3739.28
```

area.c

(Figure 3.2 – Page 50)

```c
/*
 * Compute the area of a circle.
 */
#include <stdio.h>

#define  PI  3.1415926          /* our favorite constant! */

int main()
{
  double radius;                /* user entered radius of a circle */
  double area;                  /* computed area of that circle */

  printf("Enter radius: ");
  if (scanf("%lf", &radius) == 1)
  {
    area = PI * radius * radius;
    printf("Area in floating point notation: %f\n", area);
    printf("Area in exponential notation:    %e\n", area);
    printf("Area in smallest notation:       %g\n", area);
  }
  else
    printf("Error: expected numeric radius.\n");
  return 0;                     /* always return success */
}
```

Sample Output – unsign

```
Largest unsigned int: 65535
Largest plus one:     0
Largest plus two:     1
```

unsign.c

(Figure 3.3 – Page 52)

```
/*
 * Display largest unsigned int and the numbers following it.
 */
#include <stdio.h>
#include <limits.h>             /* for UINT_MAX */

int main()
{
  unsigned int i;

  i = UINT_MAX;
  printf("Largest unsigned int: %u\n", i);
  printf("Largest plus one:     %u\n", i+1);
  printf("Largest plus two:     %u\n", i+2);
  return 0;
}
```

power.c

(Figure 3.4 – Page 54)

```
/*
 * A function to compute x to the y.
 */
double power(double x, int exp)
{
  double p;                              /* power computed so far */

  p = 1;
  if (exp > 0)
    while (exp > 0)                      /* positive exponent */
    {
      p = p * x;
      exp = exp - 1;
    }
  else
    while (exp < 0)                      /* negative exponent */
    {
      p = p / x;
      exp = exp + 1;
    }
  return p;
}
```

Sample I/O – usepower

```
3.567 4
3.567^4 = 161.887
1.2 20
1.2^20 = 38.3376
1.2 40
1.2^40 = 1469.77
3.4 20
3.4^20 = 4.26166e+10
```

usepower.c

(Figure 3.5 – Page 54)

```
/*
 * Using our own function to compute exponents (no error checking).
 */
#include <stdio.h>

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

  while (scanf("%lf %i", &x, &y) == 2)
    printf("%g^%i = %g\n", x, y, power(x, y));
  return 0;
}
```

MATH LIBRARY FUNCTIONS

FUNCTION	COMPUTES	RESTRICTIONS
`exp(x)`	e^x	
`ldexp(x,n)`	$x \times 2^n$	
`log(x)`	$\log_e x$	$x > 0$
`log10(x)`	$\log_{10}$ x	$x > 0$
`pow(x,y)`	x^y	$x \neq 0$ if $y \leq 0$, and $x > 0$ if y not integer
`sqrt(x)`	$\sqrt{x}$	$x \geq 0$
`ceil(x)`	nearest whole number $\geq x$	
`floor(x)`	nearest whole number $\leq x$	
`fabs(x)`	$\|x\|$ (absolute value)	
`fmod(x,y)`	real remainder of x/y	$y \neq 0$
`sin(x)`	sine of x in radians	
`cos(x)`	cosine of x in radians	
`tan(x)`	tangent of x in radians	
`asin(x)`	$\sin^{-1} x$, in range $[-\pi/2, \pi/2]$	$-1 \leq x \leq 1$
`acos(x)`	$\cos^{-1} x$, in range $[0, \pi]$	$-1 \leq x \leq 1$
`atan(x)`	$\tan^{-1} x$, in range $[-\pi/2, \pi/2]$	
`atan2(x,y)`	$\tan^{-1} x/y$, in range $[-\pi, \pi]$	$x \neq 0$ and $y \neq 0$
`sinh(x)`	hyperbolic sine of x	
`cosh(x)`	hyperbolic cosine of x	
`tanh(x)`	hyperbolic tangent of x	

usepow.c

(Figure 3.6 – Page 55)

```
/*
 * Using the math library to compute exponents (no error checking).
 */
#include <stdio.h>
#include <math.h>

int main()
{
  double x;                                /* user-supplied base */
  double y;                                /* user-supplied exponent */

  while (scanf("%lf %lf", &x, &y) == 2)
    printf("%g^%g = %g\n", x, y, pow(x, y));
  return 0;
}
```

Sample I/O – points

```
0 0 3 4
The distance from (0,0) to (3,4) is 5
3 4 0 0
The distance from (3,4) to (0,0) is 5
-4 -5.8 10.3 18.6
The distance from (-4,-5.8) to (10.3,18.6) is 28.2816
```

points.c

(Not In Text)

```
/*
 * Program to read in points and print the distance between them.
 */
#include <stdio.h>
#include <math.h>

main()
{
  double distance(double x1, double y1, double x2, double y2);

  double start_x;
  double start_y;
  double finish_x;
  double finish_y;

  while (scanf("%lf %lf %lf %lf",
         &start_x, &start_y, &finish_x, &finish_y) == 4)
    printf("The distance from (%g,%g) to (%g,%g) is %g\n",
           start_x, start_y, finish_x, finish_y,
           distance(start_x, start_y, finish_x, finish_y));
}

/*
 * Determine the distaance between two points.
 */
double distance(double x1, double y1, double x2, double y2)
{
  double xdiff;               /* difference in X coordinates */
  double ydiff;               /* difference in Y coordinates */

  xdiff = x1 - x2;
  ydiff = y1 - y2;
  return sqrt(xdiff * xdiff + ydiff * ydiff);
}
```

ASSIGNING A LARGE LONG TO AN INT

← **`long`** → $-1,691,154,500$

1001101100110011	0000011110001100

← **`int`** → 1980

Sample Output – conex

```
First answer: -7936
Second answer: 1980
```


conex.c

(Figure 3.7 – Page 57)

```
/*
 * Example of automatic conversions: multiplying two longs together
 * and storing them in an int.
 */
#include <stdio.h>

int main()
{
  int answer;
  long i;
  long j;

  i = 10000;
  j = 10000;
  answer = i * j;
  printf("First answer: %i\n", answer);
  answer = -1691154500L;
  printf("Second answer: %i\n", answer);
  return 0;
}
```

WHAT CONVERSIONS ACTUALLY DO

From	To	Representation Change
`short`	`int`	none
`float`	`double`	pad mantissa with 0s
`signed int`	`signed long`	sign-extended high word
`unsigned int`	`unsigned long`	zero-fill high word
`long`	`int`	truncate high word
`signed long`	`unsigned long`	none
`signed int`	`unsigned int`	none
`unsigned int`	`signed int`	none

round.c

(Figure 3.8 – Page 58)

```
/*
 * The round and trunc functions and a minimal test program using them.
 *   round - round a value up or down (for positive values only).
 *   trunc - get rid of fractional part of value.
 */
#include <stdio.h>

int main()
{
  int    round(double);
  double trunc(int);
  double x;

  printf("Enter a series of positive real numbers, followed by EOF\n");
  while (scanf("%lf", &x) == 1)
    printf("value=%f,rounded=%i,truncated=%f\n", x, round(x), trunc(x));
  return 0;
}

int round(double value)
{
  return value + 0.5;
}

double trunc(int value)
{
  return value;
}
```

Sample I/O – avg

```
97 76 85 91 98
Average of 5 values is 89.400000.
```

avg.c

(Figure 3.9 – Page 60)

```
/*
 * Compute average of its input values.
 */
#include <stdio.h>

int main()
{
  int    next;                       /* next input value */
  long   sum;                        /* running total */
  int    n;                          /* number of input values */
  int    result;                     /* did we read another value? */
  double avg;                        /* average of input values */

  sum = 0;
  n = 0;
  while ((result = scanf("%i", &next)) == 1)
  {
    sum = sum + next;
    n = n + 1;
  }
  if (result != EOF)
    printf("Warning: bad input after reading %i values\n", n);
  if (n == 0)
    avg = 0.0;
  else
    avg = (double) sum / n;
  printf("Average of %i values is %f.\n", n, avg);

  return 0;
}
```

Sample I/O – convert

```
175 2
175 in base 10 is 10101111 in base 2
175 3
175 in base 10 is 20111 in base 3
175 4
175 in base 10 is 2233 in base 4
175 5
175 in base 10 is 1200 in base 5
175 6
175 in base 10 is 451 in base 6
175 7
175 in base 10 is 340 in base 7
175 8
175 in base 10 is 257 in base 8
175 9
175 in base 10 is 214 in base 9
```

convert.c

(Figure 3.10 – Page 63)

```
/*
 * Convert base 10 values into values in a specified base between 2
 * and 10.  If the base is invalid, the program uses a default base.
 */
#include <stdio.h>

#define MINBASE   2                   /* smallest destination base */
#define MAXBASE  10                   /* largest destination base */

int main()
{
  void display_value_in_base(long value, int base);
  long value;                         /* next input value */
  int  newbase;                       /* base to convert it to */
  int  n;                             /* number of values read in */
  int  status;                        /* did program succeed? */

  while ((n = scanf("%li %i", &value, &newbase)) == 2)
  {
    if (value < 0)
    {
      printf("Error: %li is negative (ignoring -).\n", value);
      value = -value;
    }
    if (newbase < MINBASE)
    {
      printf("Error: %i invalid base (using %i).\n", newbase, MINBASE);
      newbase = MINBASE;
    }
    if (newbase > MAXBASE)
    {
      printf("Error: %i invalid base (using %i).\n", newbase, MAXBASE);
      newbase = MAXBASE;
    }
    printf("%li in base 10 is ", value);
    display_value_in_base(value, newbase);
    printf(" in base %i\n", newbase);
  }
```

```
  if (n != EOF)
  {
    printf("Error: Expected pair of integer values.\n");
    status = 1;
  }
  else
    status = 0;
  return status;
}
```

convfunc.c

(Figure 3.11 – Page 64)

```
/*
 * 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  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 is 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)
    {
      printf("%i", v / divisor);
      v = v % divisor;
      divisor = divisor / b;
    }
  }
}
```

REMINDERS

- When efficiency isn't crucial, make all integer variables **`long`**s.
- Make sure your formatting codes correspond to the types of the values that you're trying to read or write.
- Unless efficiency is a major concern, make all of your floating point variables **`double`**s.
- Avoid overflows during signed integer arithmetic by carefully choosing your data types.
- Avoid converting from a longer type to a shorter type.

CHAPTER SUMMARY

- C provides several different types of whole numbers: **`int`**s, **`short`**s, and **`long`**s, in both **`signed`** and **`unsigned`** variations. We use **`short`**s to save space, **`int`**s to save time, and **`long`**s to prevent portability problems.

- C allows several different types of integer constants: decimal (begins with a digit other than zero), octal (begins with a 0), and hex (begins with 0x), **`long`** (trailed by **`l`** or **`L`**) and **`unsigned`** (trailed by **`u`** or **`U`**).

- C provides several different types of real numbers: **`float`**s, **`double`**s, and **`long double`**s. We use **`float`**s to save space, **`double`**s to maximize precision, and **`long double`**s to take advantage of the extra range and precision provided by some machines.

- C provides only the arithmetic operators **`+`**, **`–`**, **`*`**, **`/`**, and **`%`**. Operators common to other languages are often found as functions in the math library.

CHAPTER SUMMARY

- C performs automatic conversions whenever we mix types in arithmetic expressions, assignments, or function calls (when we've provided a prototype).

- We can use the cast operator (a type surrounded by parentheses) to request specific conversions.