Programming with C

Terry Marris November 2010

5 The Integer Number Type

Previously we looked at the *double* number type and some *math.h* functions. Now we look at arithmetic with numbers of type *integer*.

5.1 The Arithmetic Operators

Numbers of type integer include the natural counting numbers, 1, 2, 3, ..., zero, and negative whole numbers such as -1, -2, -3, ...

Addition, subtraction and multiplication work as expected (provided the numbers involved are not too large - see the chapter on validation). But division requires a little thought.

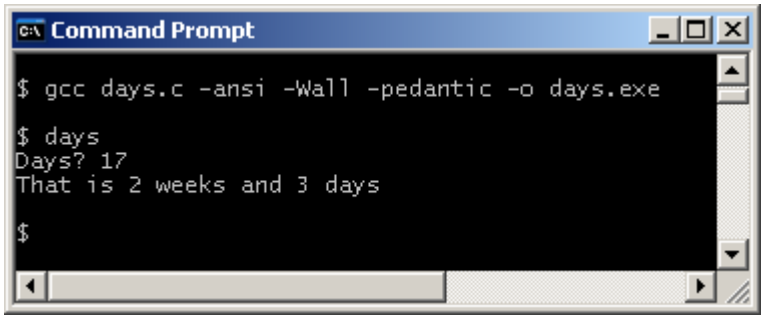
You might be tempted to think 7 / 3 = 2.33333. But 2.3333 is not an integer. For the solution recall your early school days. You might have written

7 ÷ 3 = 2 remainder 1 (because 7 = 3 x 2 + 1)

In C, 7 / 3 is 2, and 7 % 3 is 1.

The division operator, /, gives the whole number quotient after division. The remainder operator, %, gives the whole number remainder after division. *m % 5* gives a value from zero up to four inclusive, *0..4*. *m % n* gives a value from *0..n-1*.

```
/* days.c: converts days to weeks and days */

#include <stdio.h>
#include <stdlib.h>

int main()
{
  int days, weeks;
  char string[BUFSIZ];

  printf("Days? ");
  gets(string);
  days = atoi(string);
  weeks = days / 7;
  days = days % 7;
  printf("That is %d weeks and %d days\n", weeks, days);
  return 0;
}
```



Look at *days = days % 7*. The original value of *days* is divided by *7* and the remainder becomes the new value of *days*. The assignment operator, =, copies from right to left.

Of course, division by zero is not allowed. So expressions such as 5 / 0 and 5 % 0 are errors.

Problem: if you are using negative integers, the result of using the division and remainder operators depends on the compiler you are using. With the GNU C compiler, / always rounds towards zero, other compilers may behave differently. So it might be safer to use the standard library *div()* function

5.2 The div() Function

The *div()* function returns not an integer, but an item of type *div_t*. So we declare

```
div_t result;
```

To give *result* its value we write

```
result = div(days, 7);
```

Then

```
weeks = result.quot;
days = result.rem;
```

div(numerator, denominator) divides *numerator* by *denominator* and places the quotient in *quot* and the remainder in *rem*. The quotient is positive if both numerator and denominator have the same sign; negative if the signs are different. The remainder is the same sign as the numerator.

The program *divfunc.c* shown below shows how the *div()* function may be used.

```
/* divfunc.c: converts days to weeks and days */

#include <stdio.h>
#include <stdlib.h>

int main()
{
  div_t result;
  int days, weeks;
  char string[BUFSIZ];

  printf("Days? ");
  gets(string);
  days = atoi(string);
  result = div(days, 7);
  weeks = result.quot;
  days = result.rem;
  printf("That is %d weeks and %d days\n", weeks, days);
  return 0;
}
```

div_t is a structure. Structures are discussed in more detail in a later chapter. *div_t* and *div()* are defined in *stdlib.h*.

5.3 Increment and Decrement Operators

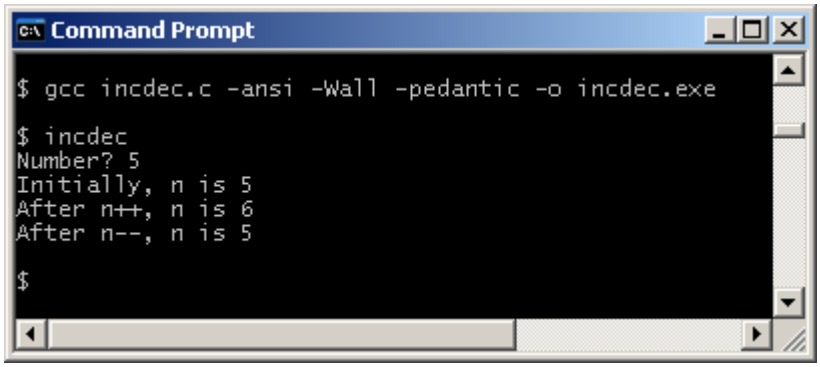
Adding 1 to an integer variable, and subtracting 1, is a common event in programming. C has a notation for doing just this.

If *n* is an integer variable, then *n*++ add 1 to its value, *n*-- subtracts one from its value.

n++ means *n = n + 1*. *n*-- means *n = n - 1*.

++ is known as the increment operator. -- is known as the decrement operator.

```
/* incdec.c: illustrates the increment and decrement operators */

#include <stdio.h>
#include <stdlib.h>

int main()
{
  char string[BUFSIZ];
  int n;

  printf("Number? ");
  gets(string);
  n = atoi(string);
  printf("Initially, n is %d\n", n);
  n++;
  printf("After n++, n is %d\n", n);
  n--;
  printf("After n--, n is %d\n", n);
  return 0;
}
```

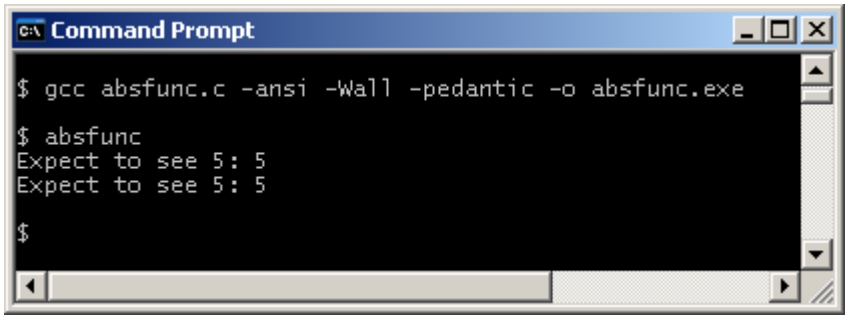


We use the increment and decrement operators extensively in the later chapter on iterations.

5.3 The abs() Function

abs(5) gives *5* and *abs(-5)* also gives *5*. *abs(n)* returns the absolute value of *n*. *abs()* is defined in *stdlib.h*.

```
/* absfunc.c: checks out the abs() function */

#include <stdio.h>
#include <stdlib.h>

int main()
{
  int n;

  n = abs(5);
  printf("Expect to see 5: %d\n", n);
  n = abs(-5);
  printf("Expect to see 5: %d\n", n);
  return 0;
}
```



abs(), and its floating point equivalent, *fabs()* is used in statistics, in the calculation of standard deviation for example. And *fabs()* could be used to ensure that the square root of only positive numbers is taken. Numbers of type *double* are floating point numbers. *fabs()* is defined in *math.h*.

5.4 Random Numbers

Random numbers are used in games, statistics and in simulations.

rand() generates a pseudo random integer number, pseudo because the sequence of numbers generated is always the same - unless you provide a starting point or seed that is different each time you run the program.

srand() allows you to provide a seed. One way of doing so is to get the time from your computer system with *time(NULL)*.

```
seed = time(NULL);
srand(seed);
```

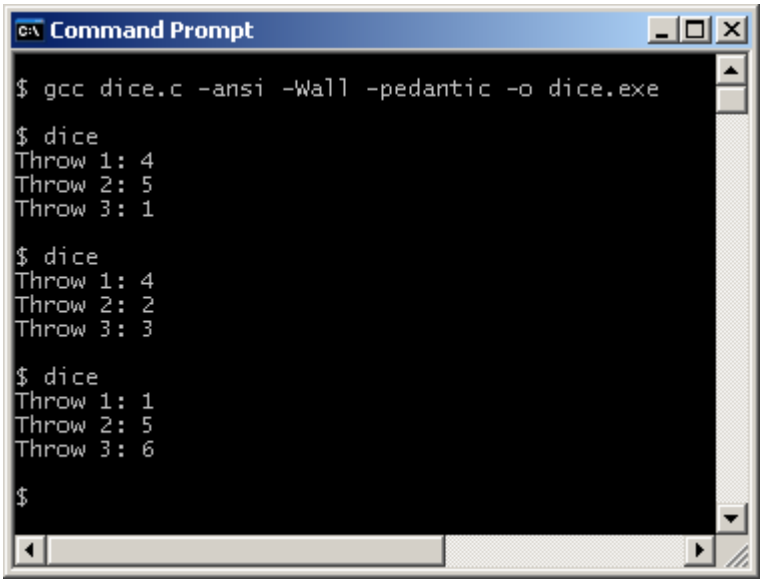
time() is defined in *time.h*. *srand()* and *rand()* are defined in *stdlib.h*.

We want a random number in the range 1..6. *rand() % 6* gives a number in the range 0..5. So *rand() %6 + 1* will give a number in the range 1..6.

The program shown below generates three random numbers in the range 1..6.

```
/* dice.c: generates random numbers in 1..6 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
  int die, seed;

  seed = time(NULL);
  srand(seed);

  die = rand();
  die = die %6 + 1;
  printf("Throw 1: %d\n", die);

  die = rand();
  die = die %6 + 1;
  printf("Throw 2: %d\n", die);

  die = rand();
  die = die %6 + 1;
  printf("Throw 3: %d\n", die);

  return 0;
}
```

We run the program three times:



5.5 Cast Operators

Look at

```
double a;
int b = 7;
int c = 2;

a = b / c;
```

This leaves *a* with the value 3.0.

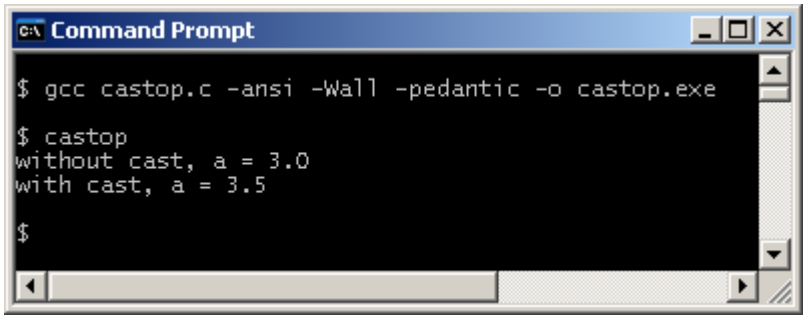
But

```
double a;
int b = 7;
int c = 2;

a = (double)b / c;
```

leaves *a* with the value 3.5

(double) is an example of a *cast operator*. It temporarily promotes the expression that follows to a value of type *double*. And so the rules of arithmetic that apply to values of type double also apply. A cast operator names a type in brackets.

```
/* castop.c: shows use of a cast operator */

#include <stdio.h>

int main()
{
  double a;
  int b = 7;
  int c = 2;

  a = b / c;
  printf("without cast, a = %0.1f\n", a);
  a = (double)b / c;
  printf("with cast, a = %0.1f\n", a);
  return 0;
}
```



Exercise 5.1

1. Evaluate
   - **a.** 11 / 5
   - **b.** 13 % 7
   - **c.** 8 % 4
   - **d.** 5 / 11
   - **e.** 41 / 6

2. Write and test a program that inputs a number of hours and outputs the corresponding number of days and hours. Test your program three times: once with hours input = 23, once with hours input = 24, once with hours input = 25.

3. Write and test a program that will generate 6 random numbers in the range 1..49.

We have seen how to perform arithmetic with numbers of type *integer*, what the *abs()* functions does, and how to generate random numbers in a chosen range.

Next we see look at selections.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988