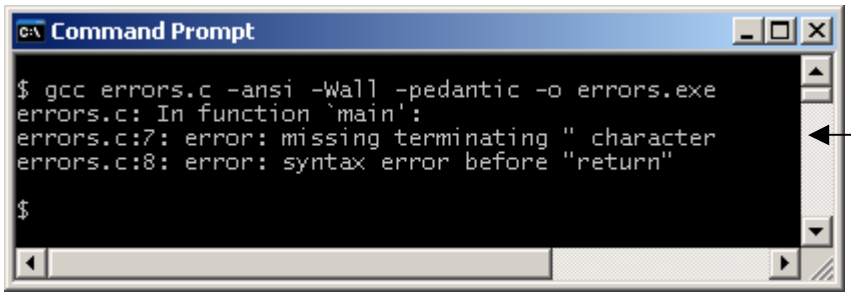
Answers

Terry Marris November 2010

Exercise 1.1

2.

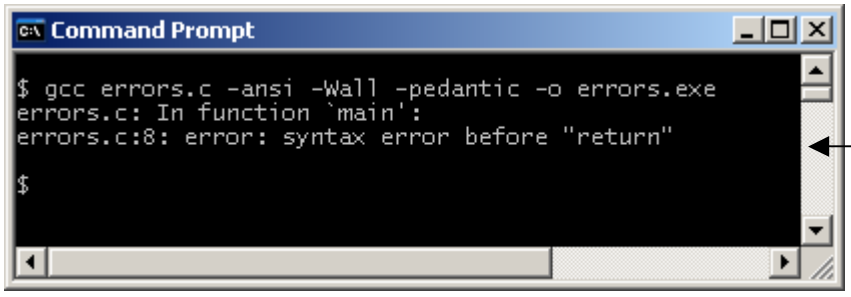
```
/* errors.c: missing quotes in printf() statement */

#include <stdio.h>

int main()
{
  printf("hello world\n);
  return 0;
}
```



error on line 7

Errors often have a knock-on effect. An error on one line causes an error message for the following line. Makes sense to fix the earliest error first, then re-compile.

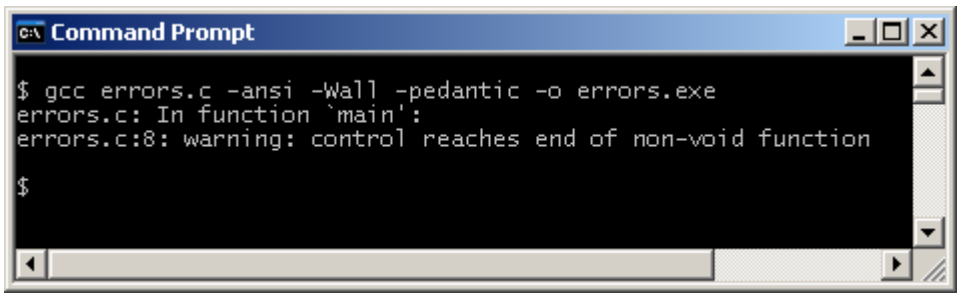
```
/* errors.c: missing semi-colon at end of printf() */

#include <stdio.h>

int main()
{
  printf("hello world\n")
  return 0;
}
```



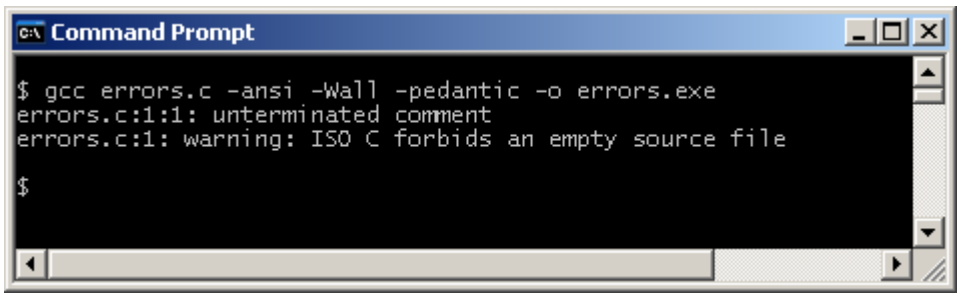
error on line 8

```
/* errors.c: missing return statement at end of program */

#include <stdio.h>

int main()
{
  printf("hello world\n");
}
```

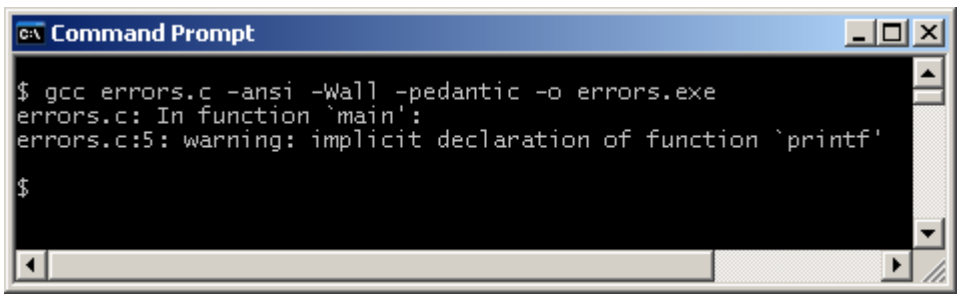
2




```
/* errors.c: missing comment terminator at end of this line

#include <stdio.h>

int main()
{
  printf("hello world\n");
  return 0;
}
```




```
/* errors.c: missing #include <stdio.h> */

int main()
{
  printf("hello world\n");
  return 0;
}
```

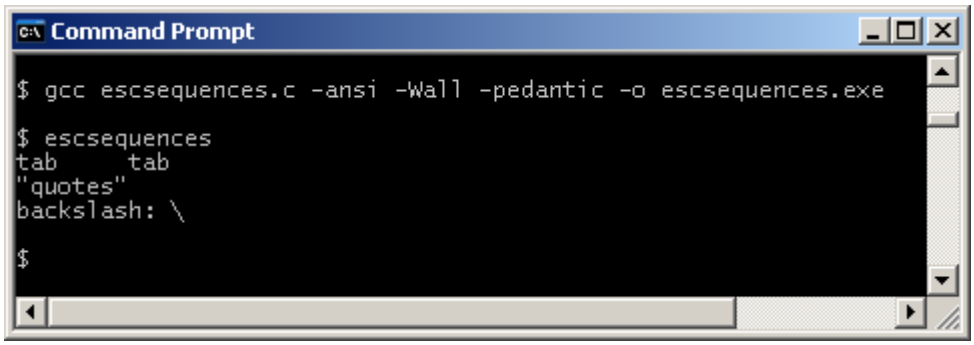


3.

```
/* rooster.c: displays lines from a song */

#include <stdio.h>

int main()
{
  printf("I had a little red rooster\n");
  printf("Too lazy to crow 'fore day\n");
  printf("I had a little red rooster\n");
  printf("Too lazy to crow 'fore day\n");
  printf("He kept ev'rything in the barnyard\n");
  printf("Eager settin' a-ready to lay\n");
  printf("Willie Dixon\n");
  return 0;
}
```

4.

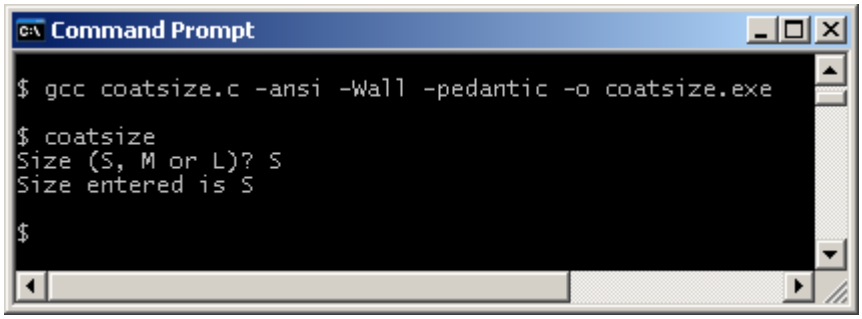
```
/* escsequences.c: demonstrates some escape sequences */

#include <stdio.h>

int main()
{
  printf("tab\ttab\n");
  printf("\"quotes\" \n");
  printf("backslash: \\ \n");
  return 0;
}
```



Exercise 2.1

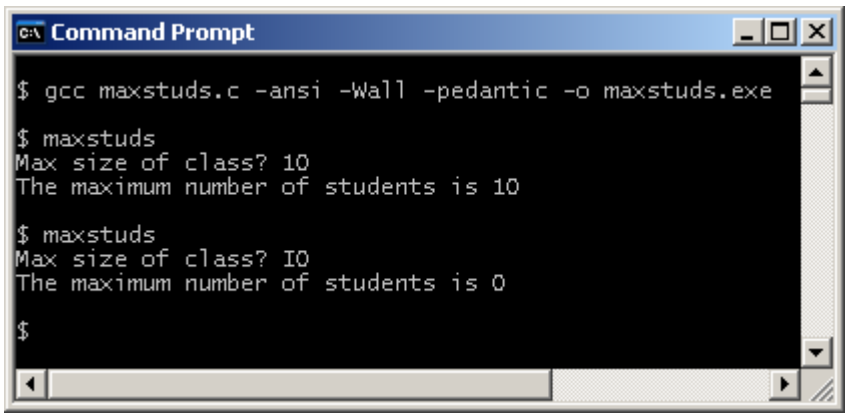
1.

```
/* coatsize.c: inputs and outputs S, M or L */

#include <stdio.h>

int main()
{
  char string[BUFSIZ];

  printf("Size (S, M or L)? ");
  gets(string);
  printf("Size entered is %c\n", string[0]);
  return 0;
}
```



2.

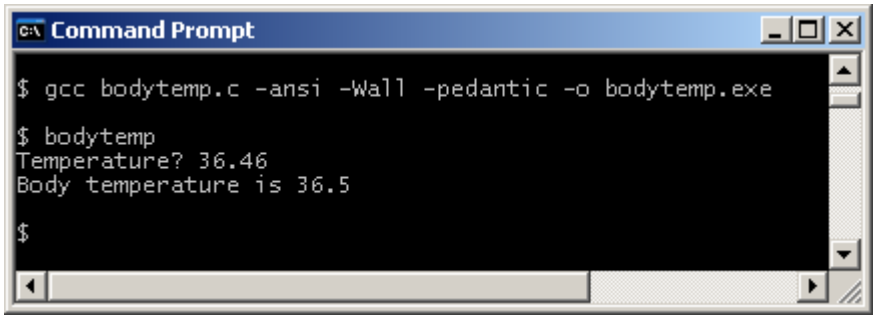
```
/* maxstuds.c: inputs and displays the max number of students */
#include <stdio.h>
#include <stdlib.h>

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

  printf("Max size of class? ");
  gets(string);
  max = atoi(string);
  printf("The maximum number of students is %d\n", max);
  return 0;
}
```



atoi() returns zero if it cannot translate the value supplied. Zero could be a valid class size, if, for example the class was cancelled. Perhaps better if a warning or an error message was displayed. Later we see how this may be done.

3.

```
/* bodytemp.c: inputs body temperature, outputs temperature to 1 dp */

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

int main()
{
  char string[BUFSIZ];
  double temperature;

  printf("Temperature? ");
  gets(string);
  temperature = atof(string);
  printf("Body temperature is %0.1f\n", temperature);
  return 0;
}
```



Exercise 3.1

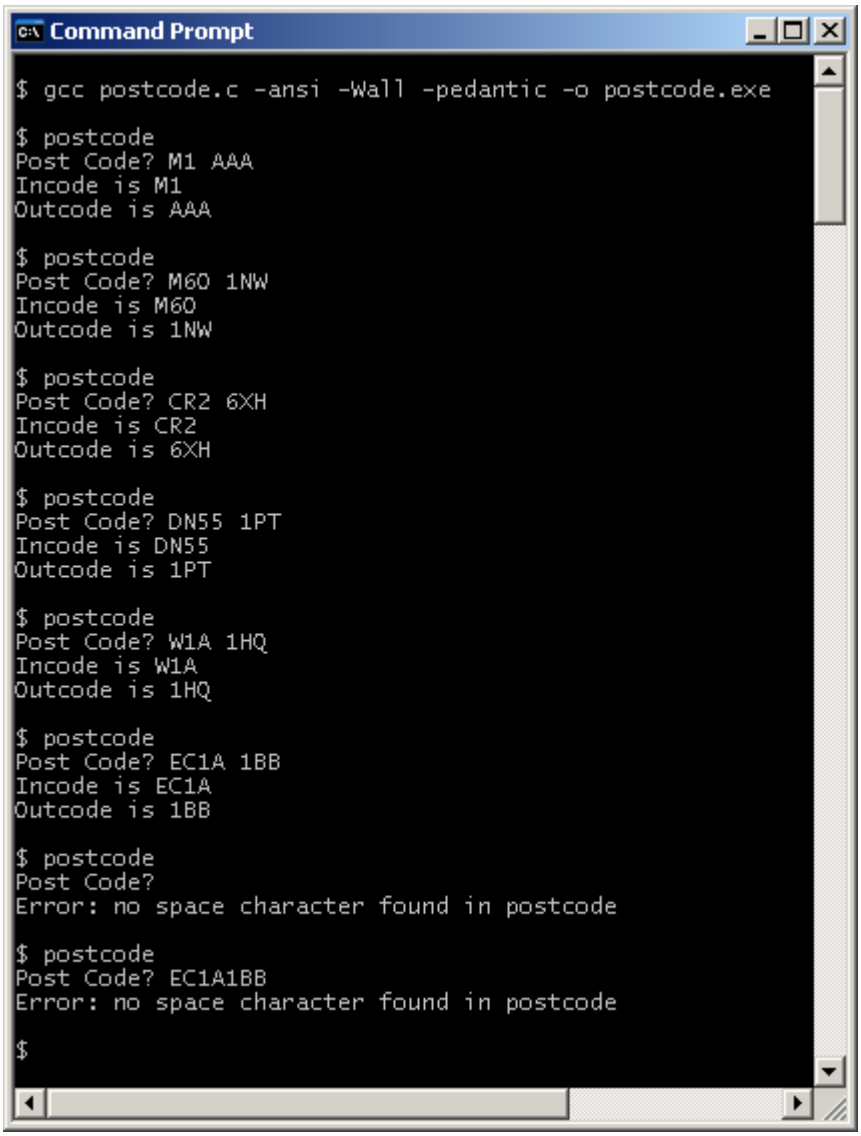
3. *strtok()* returns *NULL* if its string cannot be split into tokens. *NULL* cannot be copied into a string. So we include some error checking.

We use *strchr()* to check whether there is a space character in the postcode.

```
if (strchr(postcode, ' ') == NULL) {
  printf("Error: no space character found in postcode\n");
  exit(EXIT_FAILURE);
}
```

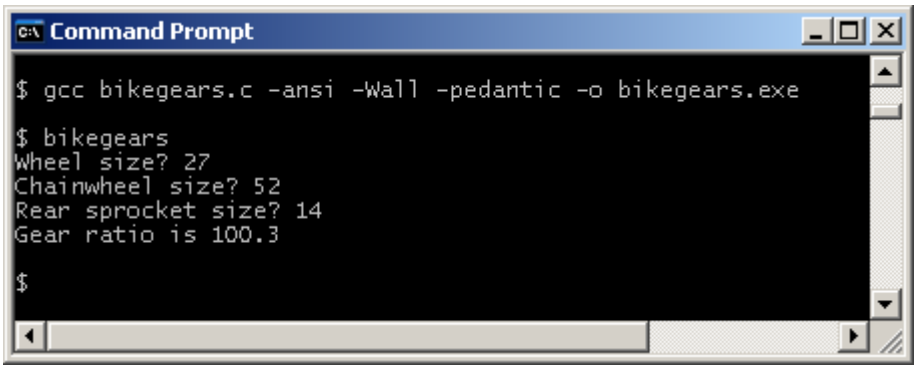
strchr() looks for a character, the space character in this case, in a string e.g. *postcode*. It returns *NULL* if the character cannot be found. *exit()* immediately ends program execution, and *EXIT_FAILURE* provides the reason. *exit()* and *EXIT_FAILURE* are both defined in *stdlib.h*.

```
/* postcode.c: splits a postcode into its parts */
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>

int main()
{
  char string[BUFSIZ];
  char postcode[9];  /* a postcode has 8 chars max + null */
  char incode[5];
  char outcode[4];

  printf("Post Code? ");
  gets(string);
  strncpy(postcode, string, 8);
  postcode[8] = '\0';
  if (strchr(postcode, ' ') == NULL) {
    printf("Error: no space character found in postcode\n");
    exit(EXIT_FAILURE);
  }
  strcpy(incode, strtok(postcode, " "));
  strcpy(outcode, strtok(NULL, " "));
  printf("Incode is %s\n", incode);
  printf("Outcode is %s\n", outcode);
  return 0;
}
```

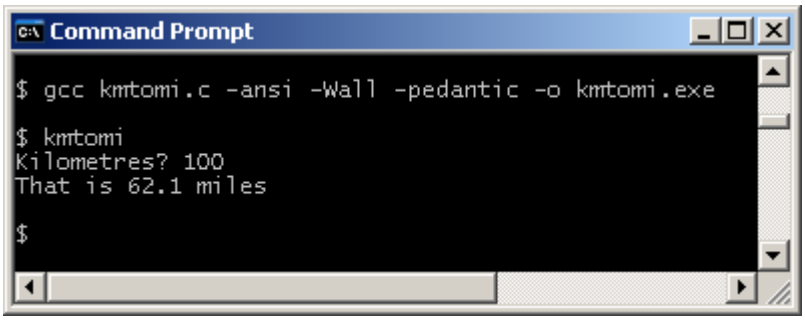


Exercise 4.1

1.
```
/* bikegears.c: calculates gear ratio */

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

int main()
{
  char string[BUFSIZ];
  double gearRatio, wheelSize, chainWheel, rearSprocket;

  printf("Wheel size? ");
  gets(string);
  wheelSize = atof(string);
  printf("Chainwheel size? ");
  gets(string);
  chainWheel = atof(string);
  printf("Rear sprocket size? ");
  gets(string);
  rearSprocket = atof(string);
  gearRatio = wheelSize * chainWheel / rearSprocket;
  printf("Gear ratio is %0.1f\n", gearRatio);
  return 0;
}
```

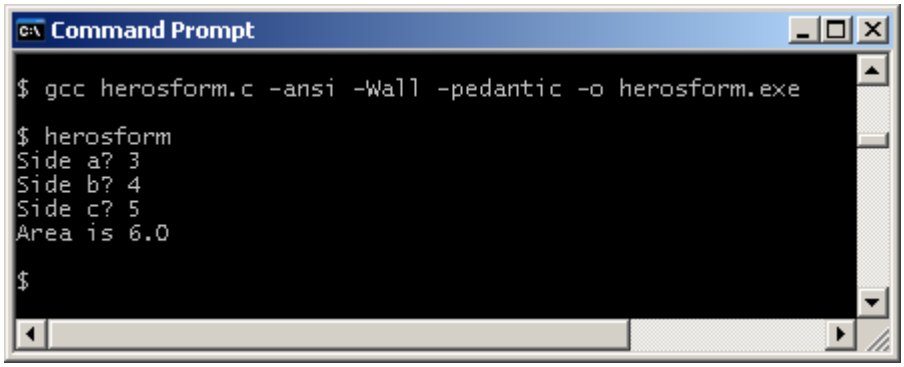


2.
```
/* kmtomi.c: converts kilometres to miles */
#include <stdio.h>
#include <stdlib.h>

int main()
{
  const double kmpermile = 0.621;  /* 1 km = 0.0621 miles */
  double km, miles;
  char string[BUFSIZ];

  printf("Kilometres? ");
  gets(string);
  km = atof(string);
  miles = km * kmpermile;
  printf("That is %0.1f miles\n", miles);
  return 0;
}
```



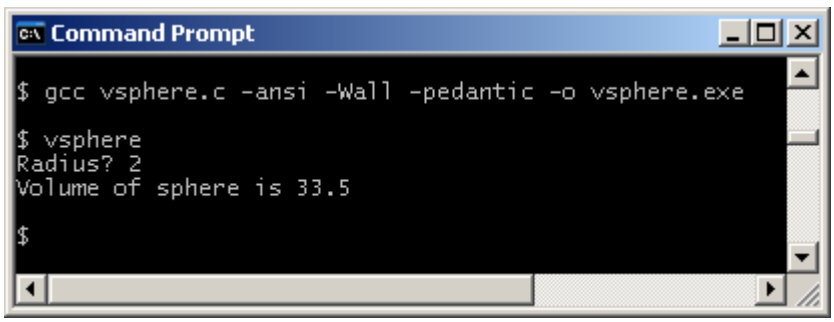
3.

```
/* herosform.c: calculates are of triangle using Hero's formula */

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

int main()
{
  char string[BUFSIZ];
  double a, b, c, s, A;

  printf("Side a? ");
  gets(string);
  a = atof(string);
  printf("Side b? ");
  gets(string);
  b = atof(string);
  printf("Side c? ");
  gets(string);
  c = atof(string);
  s = (a + b + c) / 2.0;
  A = sqrt(s * (s - a) * (s - b) * (s - c));
  printf("Area is %0.1f\n", A);
  return 0;
}
```



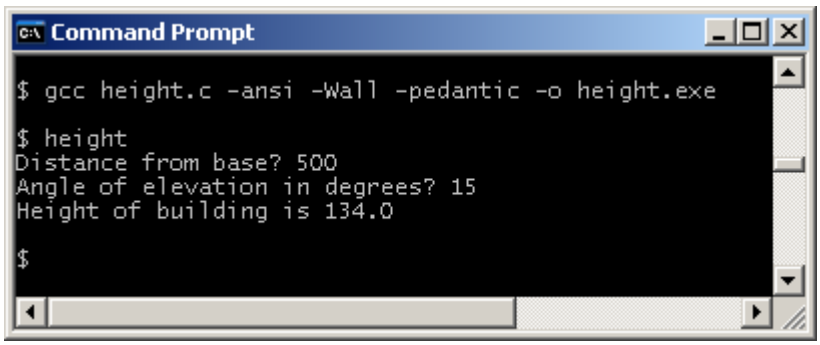
4.

```
/* vsphere.c: calulates the volume of a sphere */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  const double pi = 3.1416;
  double radius, volume;
  char string[BUFSIZ];

  printf("Radius? ");
  gets(string);
  radius = atof(string);
  volume = 4 * pi * pow(radius, 3) / 3;
  printf("Volume of sphere is %0.1f\n", volume);
  return 0;
}
```



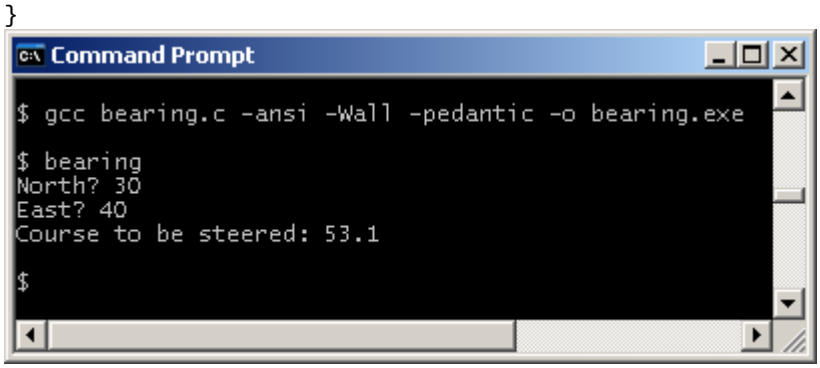
5.

```
/* height.c: calculates the height of a building */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  const double pi = 3.1416;
  double distance, height, radians, degrees;
  char string[BUFSIZ];

  printf("Distance from base? ");
  gets(string);
  distance = atof(string);
  printf("Angle of elevation in degrees? ");
  gets(string);
  degrees = atof(string);
  radians = degrees * pi / 180;
  height = distance * tan(radians);
  printf("Height of building is %0.1f\n", height);
  return 0;
}
```



6.

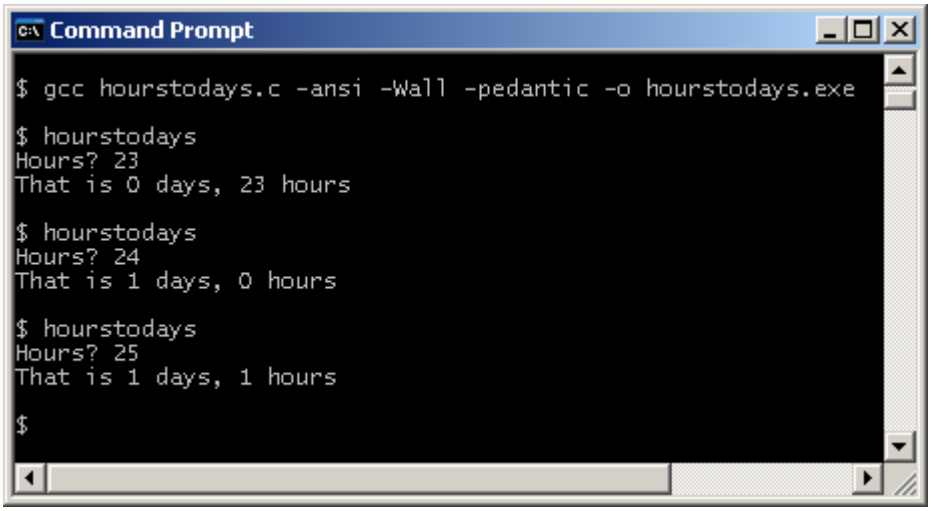
```
/* bearing.c: determines course to be steered */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  const double pi = 3.1416;
  double north, east, degrees, radians;
  char string[BUFSIZ];

  printf("North? ");
  gets(string);
  north = atof(string);
  printf("East? ");
  gets(string);
  east = atof(string);
  radians = atan(east / north);  /* north cannot be zero */
  degrees = radians * 180 / pi;
  printf("Course to be steered: %0.1f\n", degrees);
  return 0;
}
```



Exercise 5.1

1. **a** 11 / 5 = 2 **b** 13 % 7 = 6 **c** 8 % 4 = 0 **d** 5 / 11 = 0
e 41 / 6 = 6

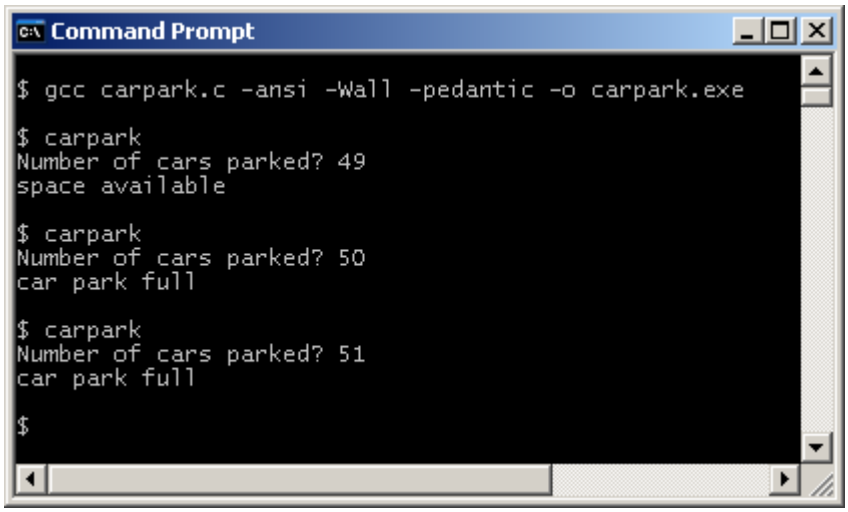
2.
```
/* hourstodays.c: converts hours to days and hours */

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

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

  printf("Hours? ");
  gets(string);
  hours = atoi(string);
  days = hours / 24;
  hours = hours % 24;
  printf("That is %d days, %d hours\n", days, hours);
  return 0;
}
```



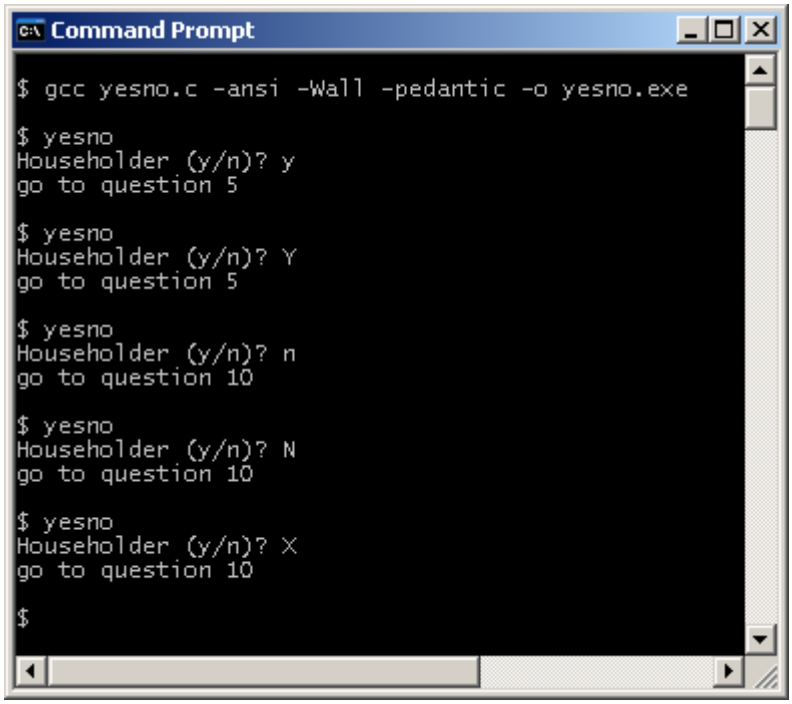
3.
```
/* loto.c: generates 6 random numbers in 1..49 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
  int number, seed;

  printf("Prints 6 random numbers in 1..49\n");
  seed = time(NULL);
  srand(seed);
  number = rand() % 49 + 1;
  printf("%d, ", number);
  number = rand() % 49 + 1;
  printf("%d, ", number);
  number = rand() % 49 + 1;
  printf("%d, ", number);
  number = rand() % 49 + 1;
  printf("%d, ", number);
  number = rand() % 49 + 1;
  printf("%d, ", number);
  number = rand() % 49 + 1;
  printf("%d ", number);
  return 0;
}
```

Exercise 6.1

1.
```
/* carpark.c: displays the state of a car park */

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

int main()
{
  const int maxspaces = 50;
  int cars;
  char string[BUFSIZ];

  printf("Number of cars parked? ");
  gets(string);
  cars = atoi(string);
  if (cars < maxspaces)
    printf("space available\n");
  else
    printf("car park full\n");
  return 0;
}
```



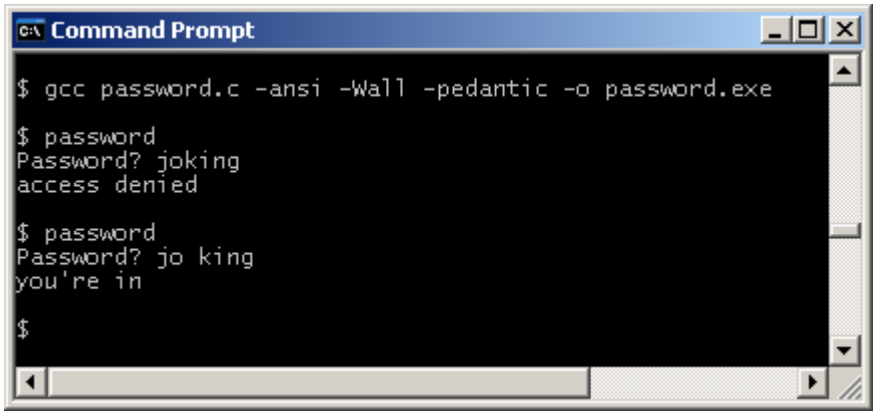
2.
```
/* yesno.c: processes a yes/no question */
#include <stdio.h>
#include <ctype.h>
int main()
{
  char string[BUFSIZ];
  char yesno;
  printf("Householder (y/n)? ");
  gets(string);
  yesno = string[0];
  yesno = tolower(yesno);
  if (yesno == 'y')
    printf("go to question 5\n");
  else
    printf("go to question 10\n");
  return 0;
}
```



Notice that any answer other than y or Y is assumed to be no.

3.

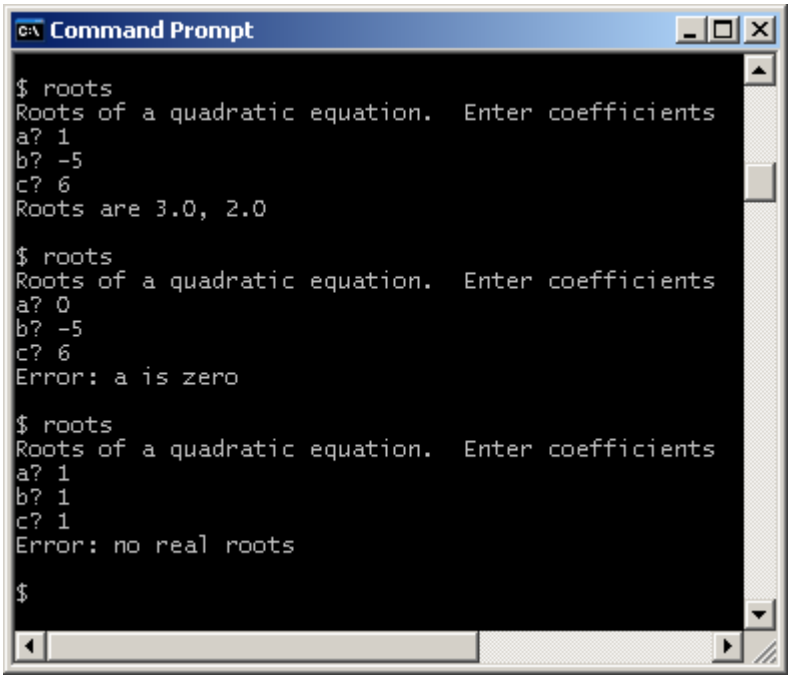
```
/* password.c: compares password entered with internal password */

#include <stdio.h>
#include <string.h>

int main()
{
  const char password[] = "jo king";
  char string[BUFSIZ];

  printf("Password? ");
  gets(string);
  if (strcmp(string, password) == 0)
    printf("you're in\n");
  else
    printf("access denied\n");
  return 0;
}
```



4.

```
/* roots.c: determines the roots of a quadratic equation */

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

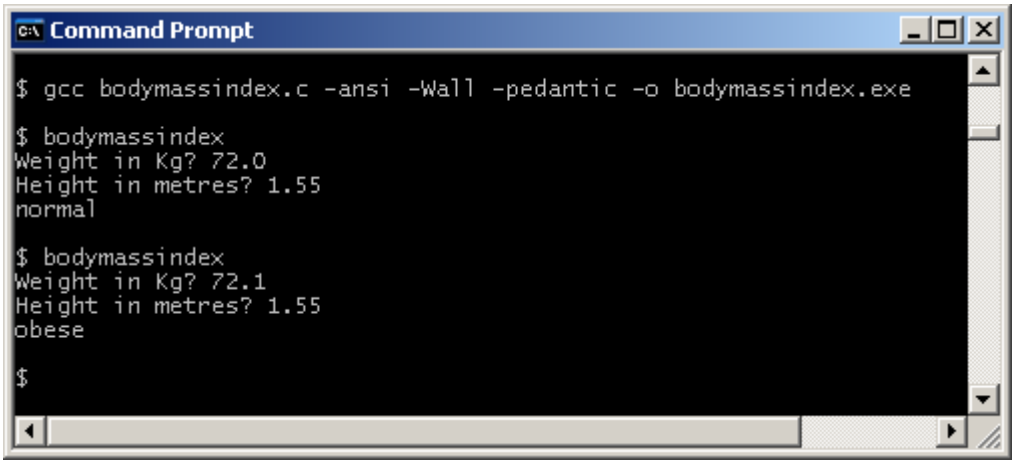
```
int main()
{
  double a, b, c, x1, x2;
  char string[BUFSIZ];

  printf("Roots of a quadratic equation.  Enter coefficients\n");
  printf("a? ");
  gets(string);
  a = atof(string);
  printf("b? ");
  gets(string);
  b = atof(string);
  printf("c? ");
  gets(string);
  c = atof(string);

  if (a == 0) {
    printf("Error: a is zero\n");
    exit(EXIT_FAILURE);          /* ends program run immediately */
  }                              /* exit(), EXIT_FAILURE defined in */
                                 /* stdlib.h */
  if (b * b < 4 * a * c) {
    printf("Error: no real roots\n");
    exit(EXIT_FAILURE);
  }

  x1 = (-b + sqrt(b * b - 4 * a * c)) / 2 * a;
  x2 = (-b - sqrt(b * b - 4 * a * c)) / 2 * a;

  printf("Roots are %0.1f, %0.1f\n", x1, x2);
  return 0;
}
```



5.

```
/* bodymassindex.c: determines bmi from weight and height */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  const double epsilon = 0.000005;
  const double limit = 30.0;
  double weight, height, bmi;
  char string[BUFSIZ];
```

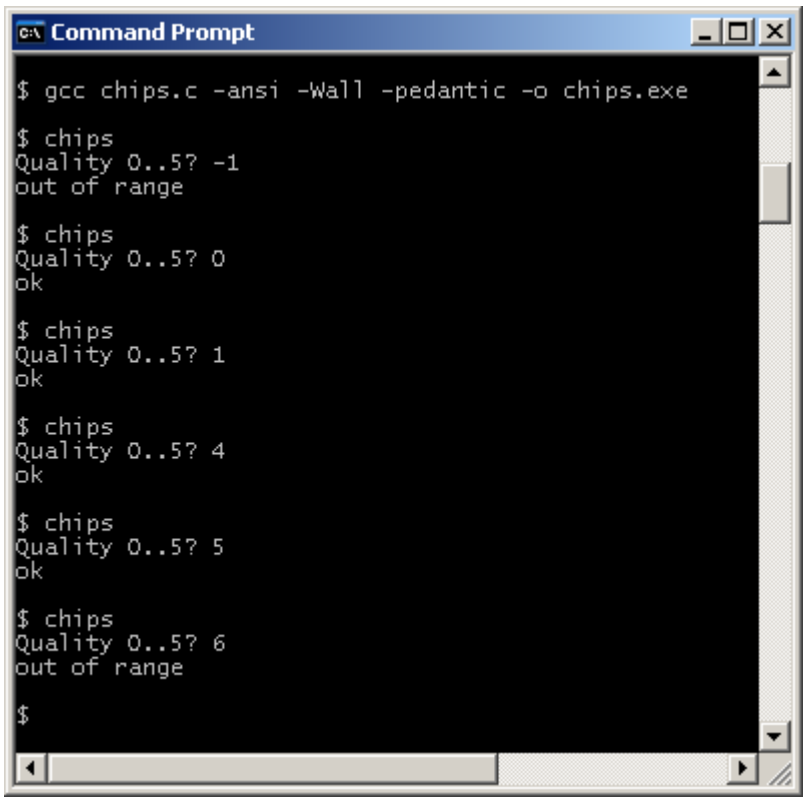
```
  printf("Weight in Kg? ");
  gets(string);
  weight = atof(string);
  printf("Height in metres? ");
  gets(string);
  height = atof(string);

  bmi = weight / pow(height, 2);
  if (bmi >= limit + epsilon)
    printf("obese\n");
  else
    printf("normal\n");
  return 0;
}
```



Exercise 7.1

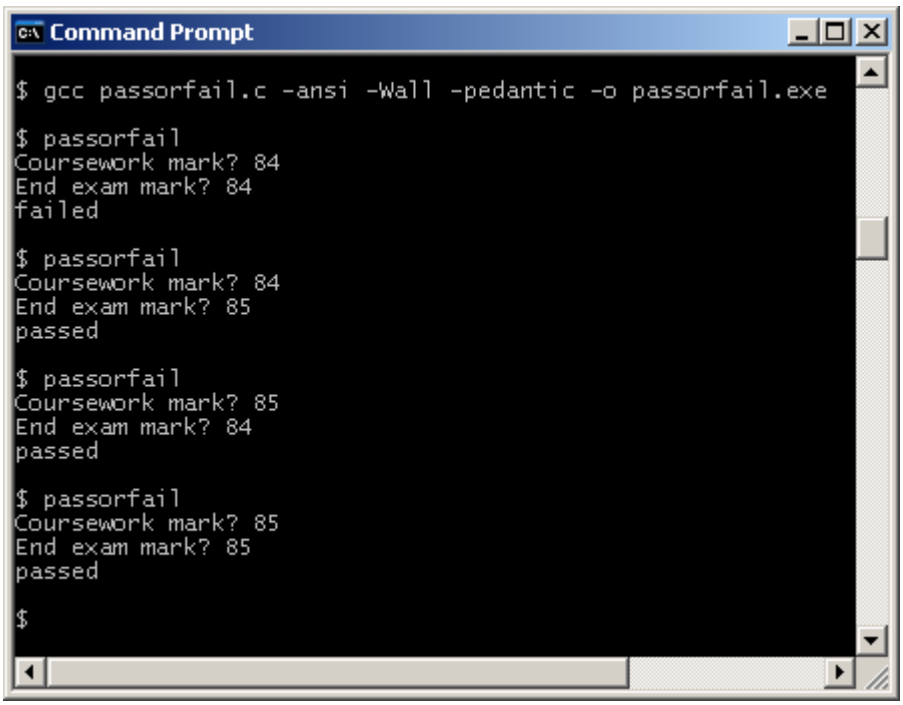
1.

```
/* chips.c: rates the quality of chips */

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

int main()
{
  const int lower = 0;
  const int upper = 5;
  int rating;
  char string[BUFSIZ];

  printf("Quality 0..5? ");
  gets(string);
  rating = atoi(string);
  if (rating >= lower && rating <= upper)
    printf("ok\n");
  else
    printf("out of range\n");
  return 0;
}
```



2.

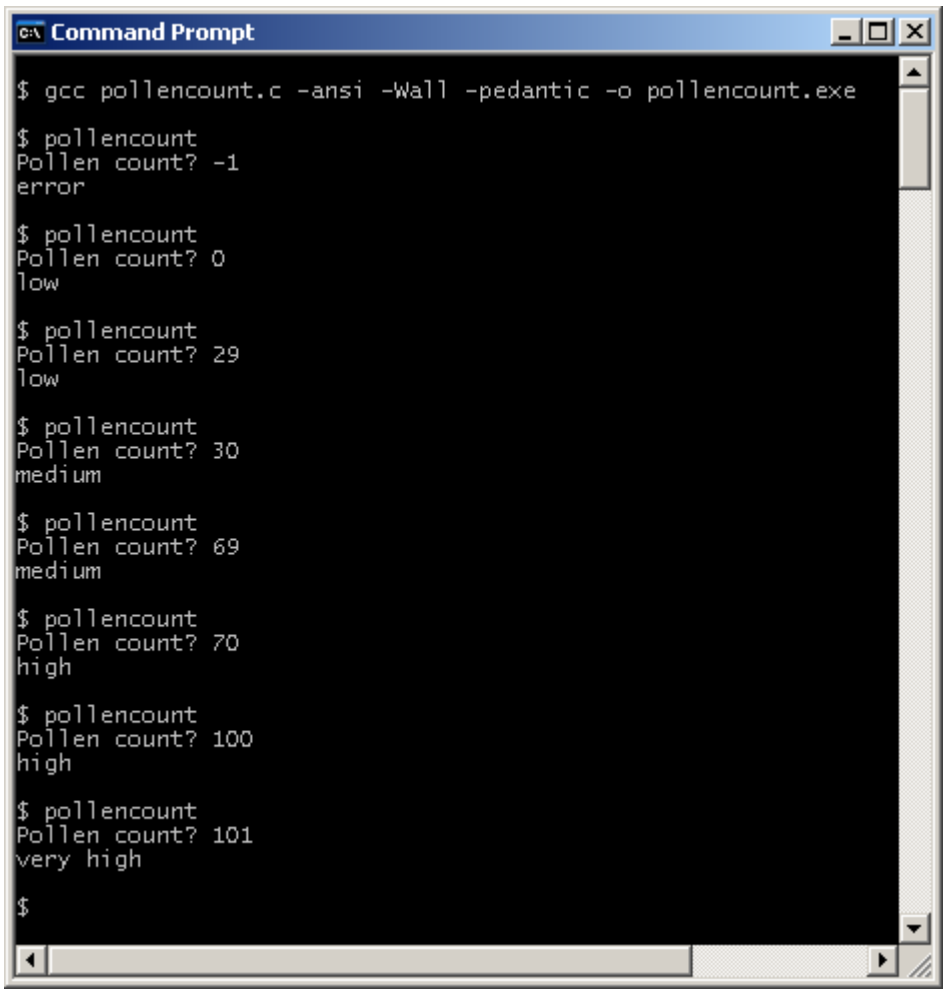
```
/* passorfail.c: shows whether a student has passed */

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

int main()
{
  const int courseworkPassMark = 85;
  const int examPassMark = 85;
  int course, exam;
  char string[BUFSIZ];

  printf("Coursework mark? ");
  gets(string);
  course = atoi(string);
  printf("End exam mark? ");
  gets(string);
  exam = atoi(string);

  if (course >= courseworkPassMark || exam >= examPassMark)
    printf("passed\n");
  else
    printf("failed\n");
  return 0;
}
```



3.

```
/* pollencount.c: outputs the severity of the pollen count */

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

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

  printf("Pollen count? ");
  gets(string);
  count = atoi(string);

  if (count < 0)
    printf("error\n");
  else if (count < 30)
    printf("low\n");
  else if (count < 70)
    printf("medium\n");
  else if (count <= 100)
    printf("high\n");
  else
    printf("very high\n");
  return 0;
}
```

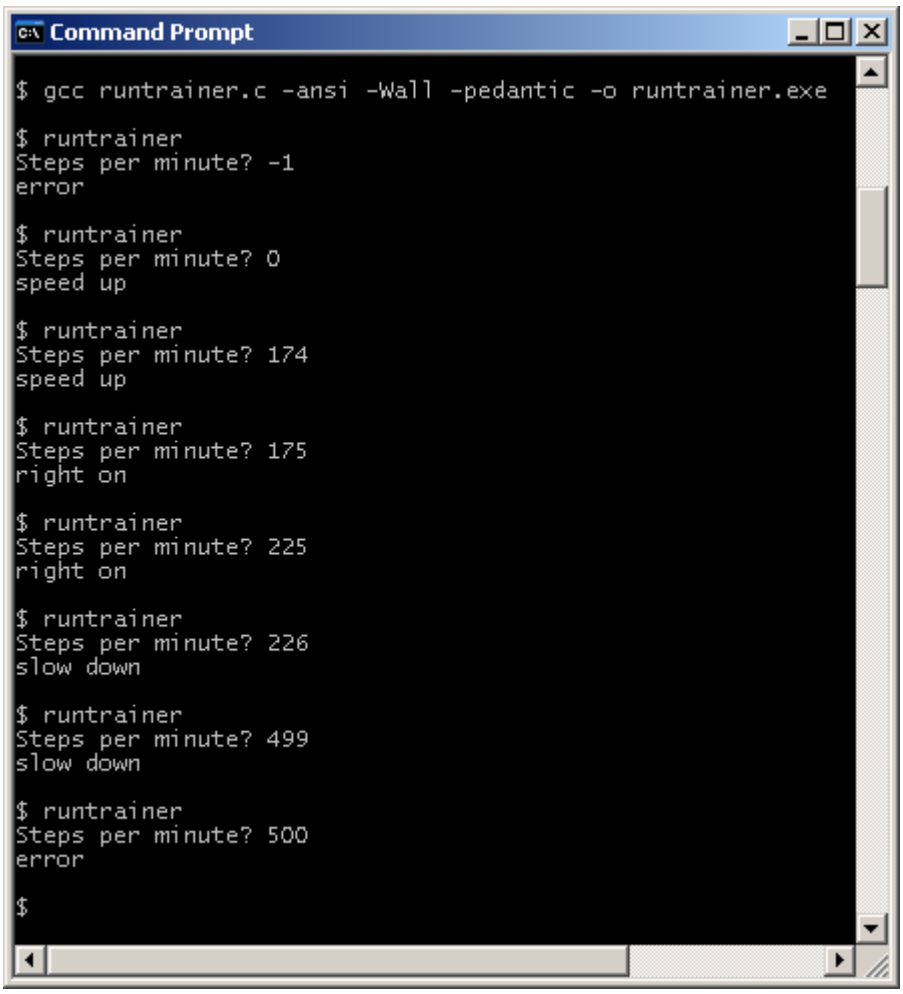
You could have written this using && and || - that is ok provided you carefully tested all the boundaries and got right answers.



4.

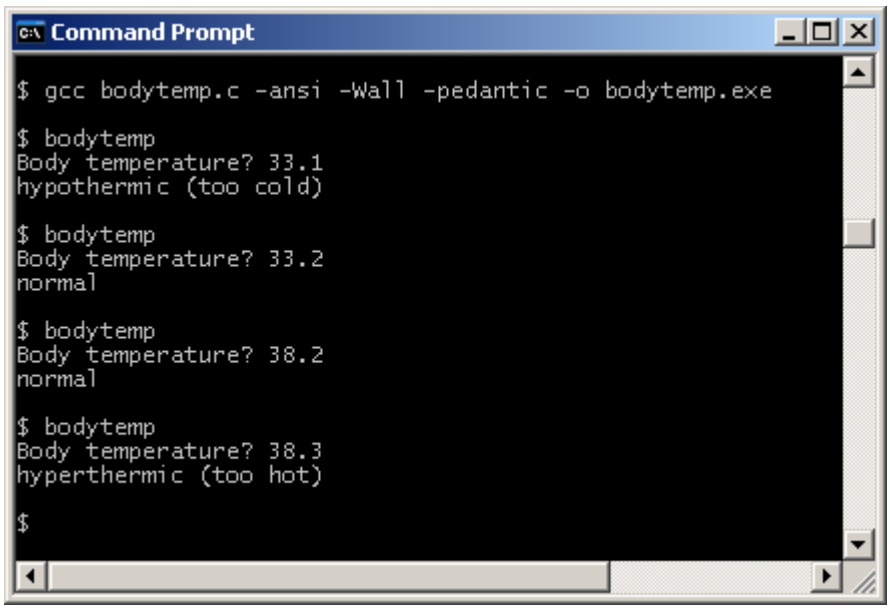
```
/* runtrainer.c: advises runner on ideal steps per minute */

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

int main()
{
  const int lower = 175;
  const int upper = 225;
  const int max = 500;
  int steps;
  char string[BUFSIZ];

  printf("Steps per minute? ");
  gets(string);
  steps = atoi(string);
  if (steps < 0)
    printf("error\n");
  else if (steps < lower)
    printf("speed up\n");
  else if (steps <= upper)
    printf("right on\n");
  else if (steps < max)
    printf("slow down\n");
  else
    printf("error\n");
  return 0;
}
```

Again, you could have used && and || where appropriate.



5.

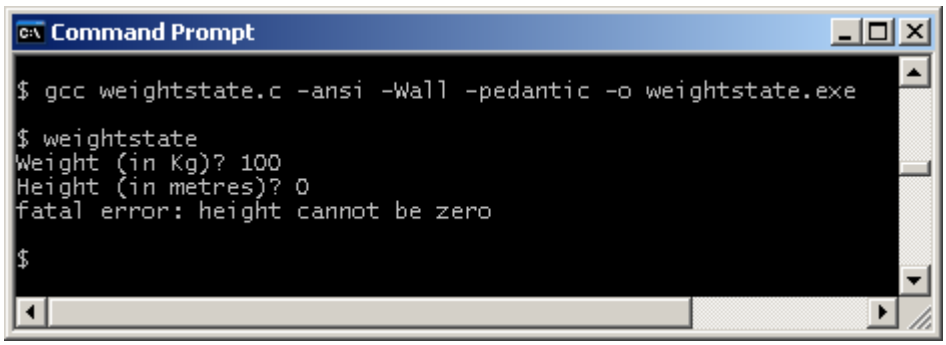
```
/* bodytemp.c: determines whether a patient is
   hypothermic or hyperthermic */

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

int main()
{
  const double lower = 33.2;
  const double upper = 38.2;
  double temp;
  char string[BUFSIZ];

  printf("Body temperature? ");
  gets(string);
  temp = atof(string);

  if (temp < lower)
    printf("hypothermic (too cold)\n");
  else if (temp > upper)
    printf("hyperthermic (too hot)\n");
  else
    printf("normal\n");
  return 0;
}
```



6.

```
/* weightstate.c: calculates bmi, displays state */

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

int main()
{
  const double epsilon = 0.001;
  double difference, weight, height, bmi;
  char string[BUFSIZ];

  printf("Weight (in Kg)? ");
  gets(string);
  weight = atof(string);
  printf("Height (in metres)? ");
  gets(string);
  height = atof(string);

  difference = abs(height - 0.0);
  if (difference < epsilon) {
    printf("fatal error: height cannot be zero\n");
    exit(EXIT_FAILURE);
  }

  bmi = weight / pow(height, 2);
  if (bmi < 15.0)
    printf("starvation\n");
  else if (bmi < 18.6)
    printf("underweight\n");
  else if (bmi < 29.9)
    printf("normal\n");
  else if (bmi < 40.0 + epsilon)
    printf("obese\n");
  else
    printf("morbidly obese\n");
  return 0;
}
```

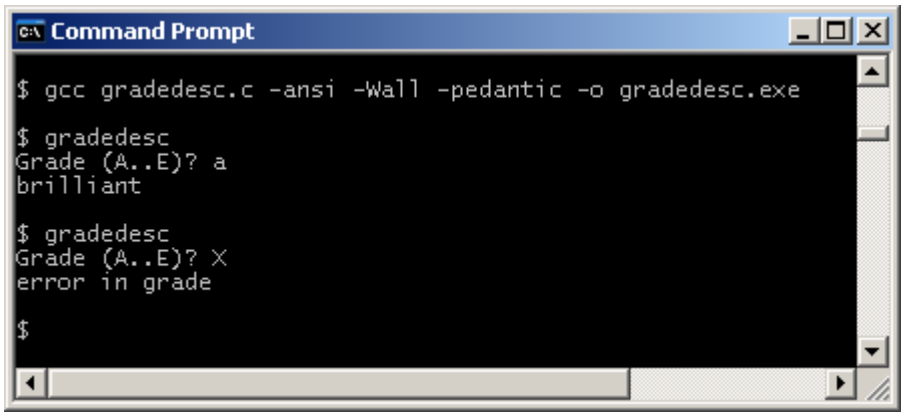
Notice we have not rigidly applied the rule for determining equality for values of type double because the application is not safety critical. However, we have applied the equality test for zero height since you cannot divide by zero.



Exercise 8.1

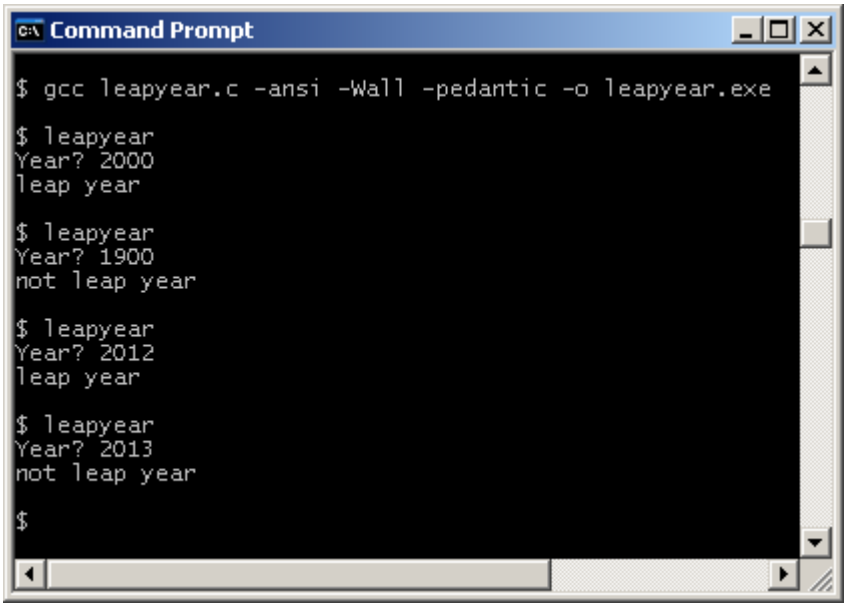
1.

```
/* gradedesc.c: translates a grade into a description */

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

int main()
{
  char string[BUFSIZ];

  printf("Grade (A..E)? ");
  gets(string);
  string[0] = toupper(string[0]);
  switch (string[0]) {
  case 'A':
    printf("brilliant\n");
    break;
  case 'B':
    printf("very good\n");
    break;
  case 'C':
    printf("good\n");
    break;
  case 'D':
    printf("fair\n");
    break;
  case 'E':
    printf("appalling\n");
    break;
  default:
    printf("error in grade\n");
    exit(EXIT_FAILURE);
  }
  return 0;
}
```



2.

```
/* leapyear.c: determines whether a year is a leap year */

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

int main()
{
  int isLeapYear, year;
  char string[BUFSIZ];

  printf("Year? ");
  gets(string);
  year = atoi(string);

  isLeapYear = (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
  isLeapYear ? printf("leap year\n") : printf("not leap year\n");

  return 0;
}
```



Since || has a lower precedence than than &&, you would think that the brackets in

```
isLeapYear = (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
```

are unecessary. But my compiler politely suggested I include them so ... I did.

Exercise 9.1

1. a.

	(num%2==0)?	print	num	i	i<4?
initially			1	0	true
enter loop	false		2	1	true
enter loop	true	2	3	2	true
enter loop	false		4	3	true
enter loop	true	4	5	4	false
exit loop					

b.

	sum	n	i	i<3?	print
initially	0	1	0	true	
enter loop	1	2	1	true	
enter loop	5	3	2	true	
enter loop	14	4	3	false	
exit loop					14

2.

```
/* avge.c: finds the arithmetic mean of the first six ints > 0 */

#include <stdio.h>

int main()
{
  int sum = 0;
  double avge = 0.0;
  int num = 1;
  int count;

  count = 0;
  while (count < 6) {
    sum = sum + num;
    num++;
    count++;
  }
  avge = (count != 0) ? (double)sum / count : 0.0;
  printf("Average = %0.1f\n", avge);
  return 0;
}
```

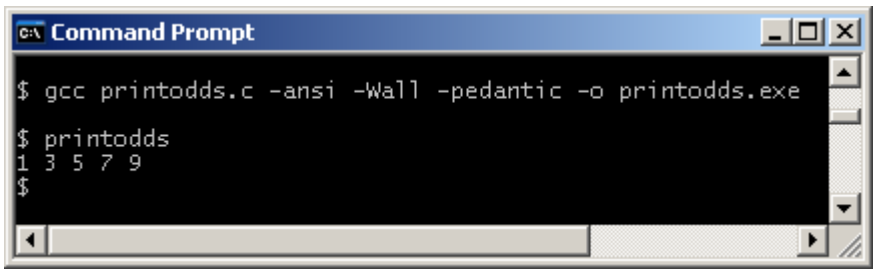
In ordinary arithmetic: 1 + 2 + 3 + 4 + 5 + 6 = 21. 21 / 6 = 3.5. Two important points:

```
  avge = (count != 0) ? (double)sum / count : 0.0;
```

In caluclating the average we divide by a count of the numbers. This count cannot be zero (since you cannot divide by zero) and so we have included a guard to prevent this happening. Secondly, to ensure that the division operator, /, works as it should with numbers of type double, we have included the cast operator, *(double)*. This temporarily promotes the the expression that follows into a value of type *double*.

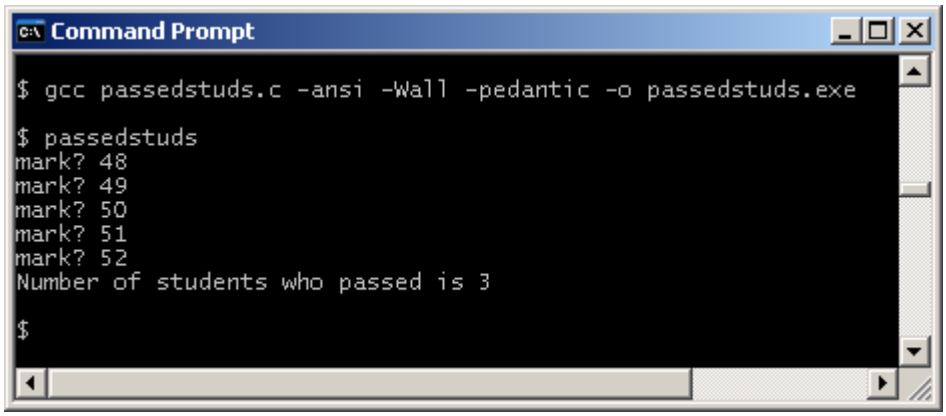
3.

```
/* printodds.c: prints the first five odd integers */

#include <stdio.h>

int main()
{
  int n = 1;
  int i;

  i = 0;
  while (i < 5) {
    if (n % 2 != 0)
      printf("%d ", n);
    n = n + 2;
    i++;
  }
  return 0;
}
```



4.

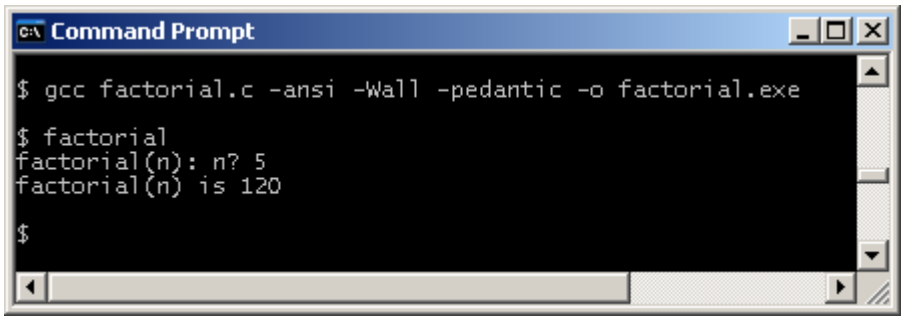
```
/* passedstuds.c: counts the students who passed */

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

int main()
{
  const int passMark = 50;
  int mark;
  int count = 0;
  int nStuds;
  char string[BUFSIZ];

  nStuds = 0;
  while (nStuds < 5) {
    printf("mark? ");
    gets(string);
    mark = atoi(string);
    if (mark >= passMark)
      count++;
    nStuds++;
  }
  printf("Number of students who passed is %d\n", count);
  return 0;
}
```



5.

```
/* factorial.c: computes factorial(n), n > 0 */

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

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

  printf("factorial(n): ");
  printf("n? ");
  gets(string);
  n = atoi(string);
  if (n < 1) {
    printf("n must be more than zero\n");
    exit(EXIT_FAILURE);
  }
```

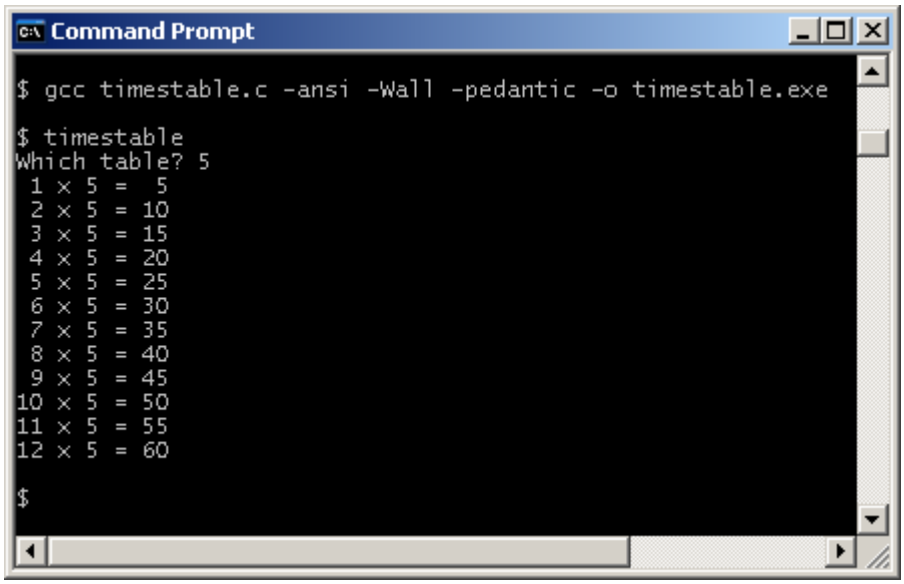
```
  nfactorial = 1;
  i = 1;
  while (i <= n) {
    nfactorial = i * nfactorial;
    i++;
  }

  printf("factorial(n) is %d\n", nfactorial);
  return 0;
}
```



Exercise 10.1

1.

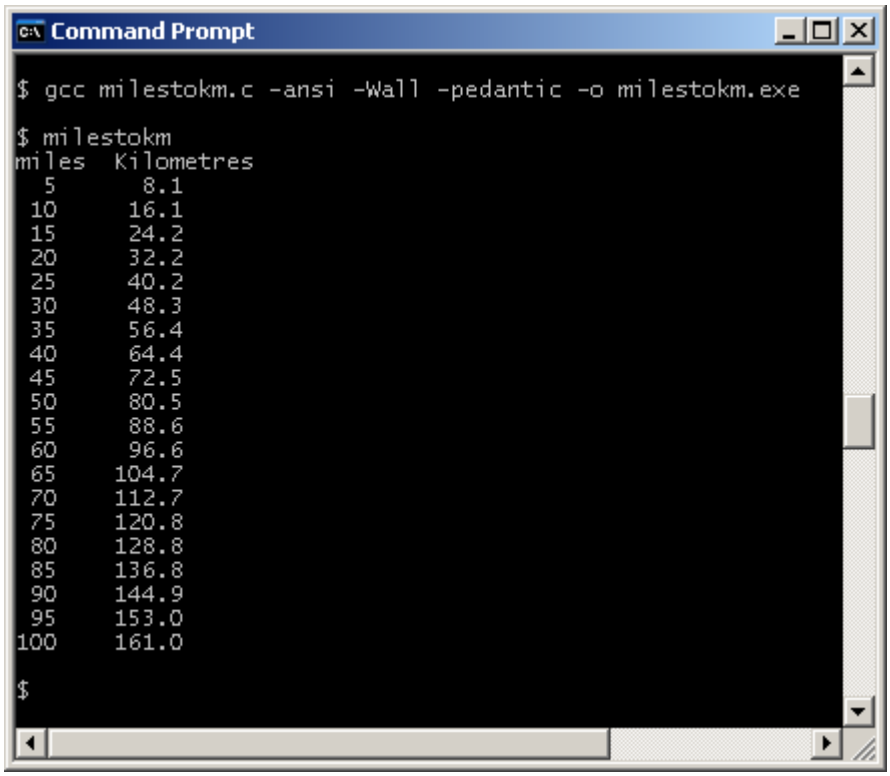
```
/* timetable.c: prints a multiplication table */

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

int main()
{
  char string[BUFSIZ];
  const int max = 12;
  int n, i;

  printf("Which table? ");
  gets(string);
  n = atoi(string);
  for (i = 1; i <= max; i++)
    printf("%2d x %d = %2d\n", i, n, (i * n));
  return 0;
}
```



Notice that the *2* in *%2d* specifies a field width. Numbers are printed right justified in this width.

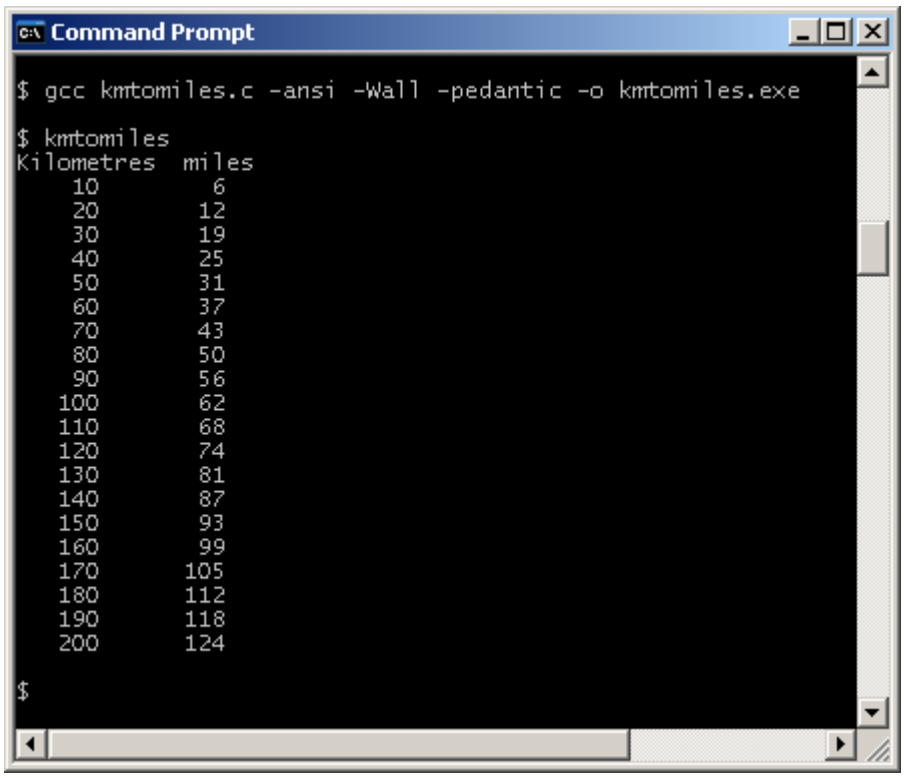
2.

```
/* milestokm.c: prints miles to Kilometres table */

#include <stdio.h>

int main()
{
  const double km = 1.61;  /* Km to a mile */
  int miles;

  printf("miles  Kilometres\n");
  for (miles = 5; miles <= 100; miles = miles + 5)
    printf("%3d %8.1f\n", miles, (miles * km));

  return 0;
}
```



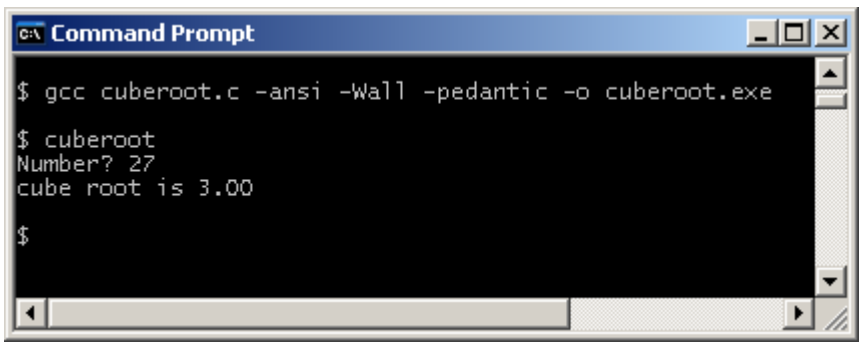
3.

```
/* kmtomiles.c: prints Kilometres to miles table */

#include <stdio.h>

int main()
{
  const double miles = 0.62;  /* miles to a Km */
  double km;

  printf("Kilometres  miles\n");
  for (km = 10; km <= 200; km = km + 10)
    printf("%6.0f %8.0f\n", km, (km * miles));
  return 0;
}
```



4.

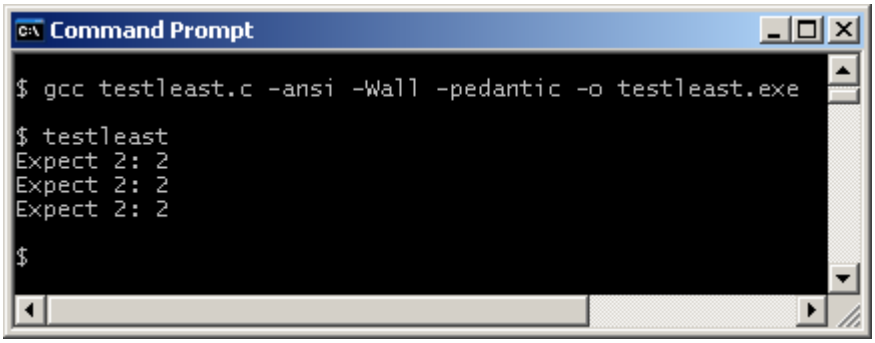
```
/* cuberoot.c: calculates cube root */

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

int main()
{
  double epsilon = 0.0001;
  double num, prev, curr; /* number, previous guess, current guess */
  char string[BUFSIZ];

  printf("Number? ");
  gets(string);
  num = atof(string);

  if (num < epsilon) {
    printf("number cannot bge less than zero\n");
    exit(EXIT_FAILURE);
  }

  prev = num;
  for (curr = num; abs(prev * prev * prev - num) > epsilon;
              prev = curr, curr = (2*prev + num /(curr*curr)) / 3);
    ;
  printf("cube root is %0.2f\n", curr);
  return 0;
}
```



Exercise 11.1

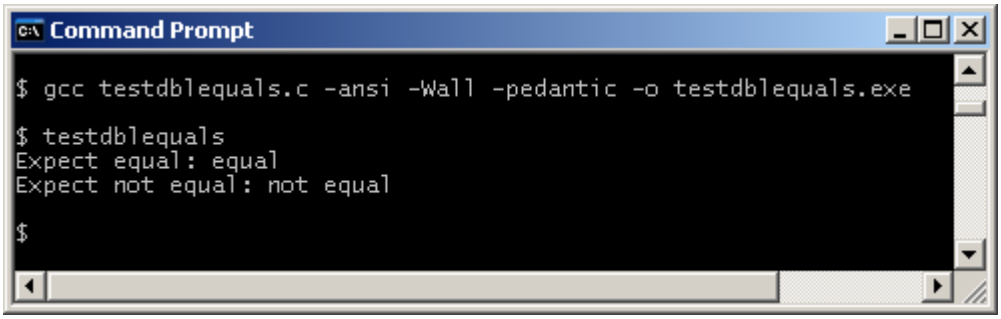
1.

```
/* testleast.c: tests the least() function */

#include <stdio.h>

/* least: returns the least of two integers */
int least(int a, int b)
{
  if (a < b)
    return a;
  else
    return b;
}

int main()
{
  printf("Expect 2: %d\n", least(2, 3));
  printf("Expect 2: %d\n", least(3, 2));
  printf("Expect 2: %d\n", least(2, 2));
  return 0;
}
```



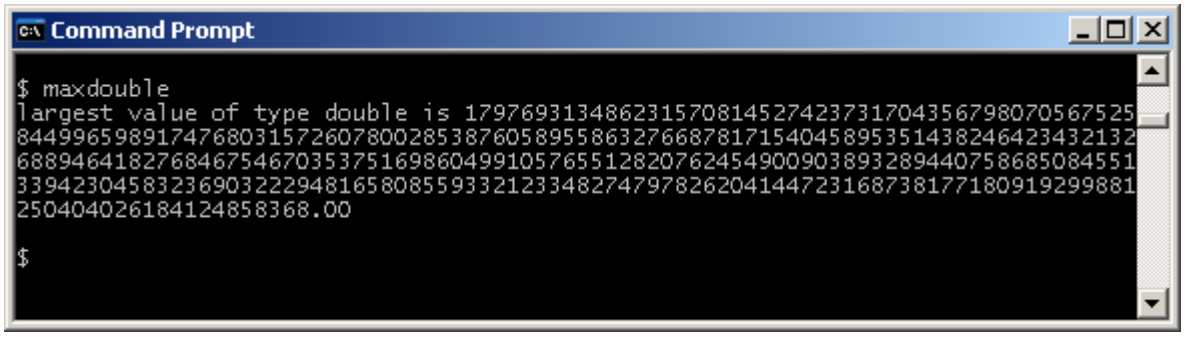
2.

```
/* testdblequals.c: tests dblEquals() */

#include <stdio.h>
#include <math.h>

/* dblEquals: returns 1 (true) if a and b are close enough to equal,
    (as determined by tolerance) otherwise returns 0 (false) */
int dblEquals(double a, double b, double tolerance)
{
  double difference = fabs(a - b);
  return difference < tolerance;
}

int main()
{
  printf("Expect equal: ");
  dblEquals(2.0,2.09,0.1)? printf("equal\n"): printf("not equal\n");
  printf("Expect not equal: ");
  dblEquals(2.0,2.11,0.1)? printf("equal\n"): printf("not equal\n");
  return 0;
}
```



3.

```
/* maxdouble.c: prints the largest value of type double */

#include <stdio.h>
#include <float.h>

int main()
{
  printf("largest value of type double is %0.2f\n", DBL_MAX);
  return 0;
}
```


Command Prompt
$ maxdouble
largest value of type double is 17976931348623157081452742373170435679807056752584499659891747680315726078002853876058955863276687817154045895351438246423432132688946418276846754670353751698604991057655128207624549009038932894407586850845513394230458323690322294816580855933212334827479782620414472316873817718091929988125040402618412485836 8.00
$
```

That is 309 digits (I think!).

**4.**

```
/* testgetdbl.c: tests getDouble() */

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

/* getString: reads a string from the keyboard, returns its length */
int getString(char string[], int maxLength)
{
  char c;
  int i = 0;

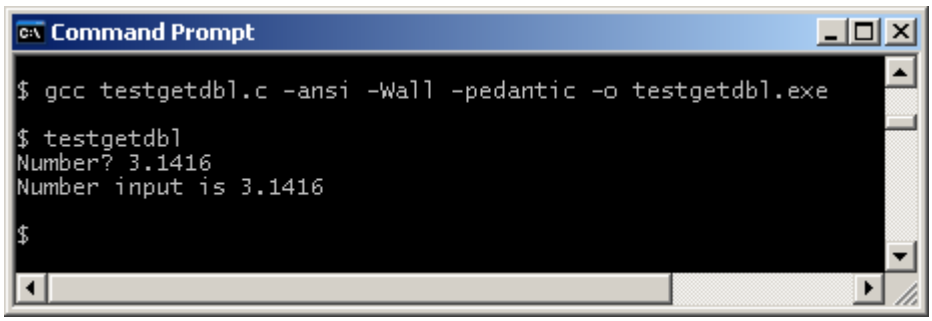
  while ((c = getchar()) != '\n') {
    if (i < maxLength - 1) {
      string[i] = c;
      i++;
    }
  }
  string[i] = '\0';
  return i;
}

/* getDouble: returns the value of type double entered at the keyboard */
double getDouble()
{
  char string[308];

  getString(string, 308);
  return atof(string);
}

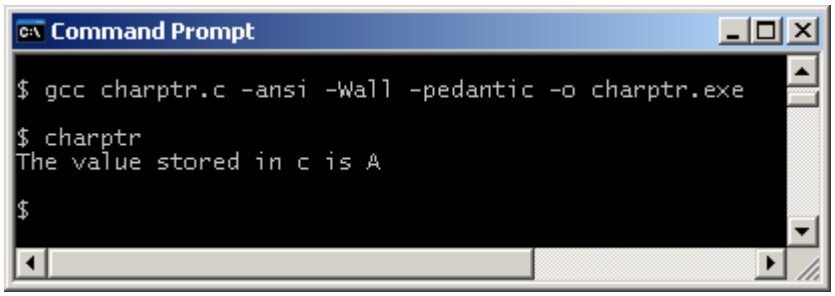
int main()
{
  double n;

  printf("Number? ");
  n = getDouble();
  printf("Number input is %0.4f\n", n);
  return 0;
}
```
```



Exercise 13.1

2.
```
/* charptr.c: shows use of pointers with char */

#include <stdio.h>

int main()
{
  char c;
  char *ptrToC;

  c = 'A';
  ptrToC = &c;
  printf("The value stored in c is %c\n", *ptrToC);
  return 0;
}
```



Exercise 14.1

1.
```
/* stringToLowerCase: returns lower case version of string parameter */
char *stringToLowerCase(const char *string)
{
  char *lcString = malloc(strlen(string) + 1); /* +1 for '\0' */
  int i;

  for (i = 0; string[i] != '\0'; i++)
    lcString[i] = string[i];
  lcString[i] = '\0';

  for (i = 0; lcString[i] != '\0'; i++)
    lcString[i] = tolower(lcString[i]);
  return lcString;
}
```

A duplicate copy of string is made, then each element in the duplicated string is converted to lower case - if poasible - then the amended duplicated string is returned.

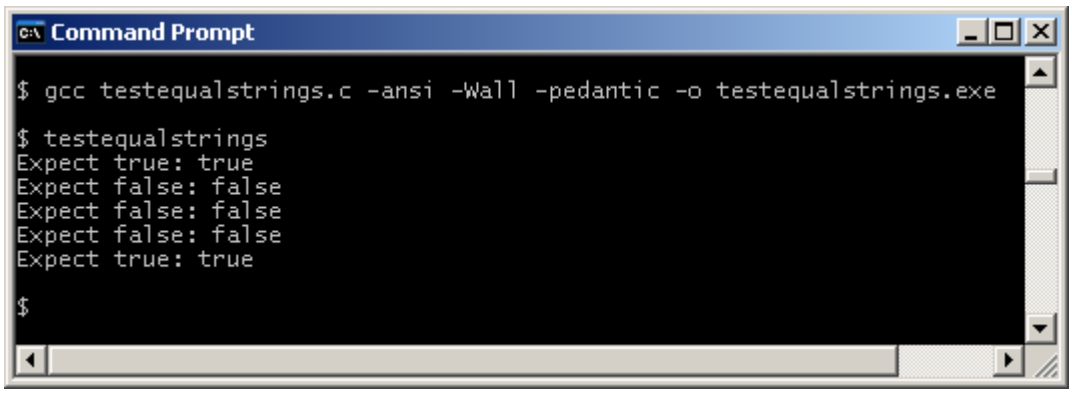
2.
```
/* stringToUpperCase: returns upper case version of string parameter */
char *stringToUpperCase(const char *string)
{
  char *ucString = malloc(strlen(string) + 1); /* +1 for '\0' */
  int i;

  for (i = 0; string[i] != '\0'; i++)
    ucString[i] = string[i];
  ucString[i] = '\0';
  for (i = 0; ucString[i] != '\0'; i++)
    ucString[i] = toupper(ucString[i]);
  return ucString;
}
```

Makes a duplicate copy of strig, then goes through the duplicate converting each char to lower case - if it can.

3.
```
/* getStr: returns a string from the keyboard */
char *getStr(int size)
{
  char *string = malloc(size + 1);
  char c;
  int i = 0;

  while ((c = getchar()) != '\n') {
    if (i < size - 1) {
      string[i] = c;
      i++;
    }
  }
  string[i] = '\0';
  return string;
}
```

Creates a string of the given size, reads chars from the keyboard until the given size is nearly reached, or the newline char is entered, whichever happens first, assigning each read into string. Appends null and returns the completed string.

4.
```
/* testequalstrings: tests equalStrings() */

#include <stdio.h>

/* equalStrings: returns 1 if str1 and str2 have identical elements,
   otherwise returns 0 */
int equalStrings(const char *str1, const char* str2)
{
  int i;

  for (i = 0; str1[i] != '\0'; i++)
    if (str1[i] != str2[i])
      return 0;
  return str1[i] == str2[i];
}

int main()
{
  printf("Expect true: ");
  equalStrings("jo king","jo king")? printf("true\n"): printf("false\n");
  printf("Expect false: ");
  equalStrings("jo king","Jo King")? printf("true\n"): printf("false\n");
  printf("Expect false: ");
  equalStrings("jo king","jo kin")? printf("true\n"): printf("false\n");
  printf("Expect false: ");
  equalStrings("jo kin","jo king")? printf("true\n"): printf("false\n");
  printf("Expect true: ");
  equalStrings("","")? printf("true\n"): printf("false\n");

  return 0;
}
```



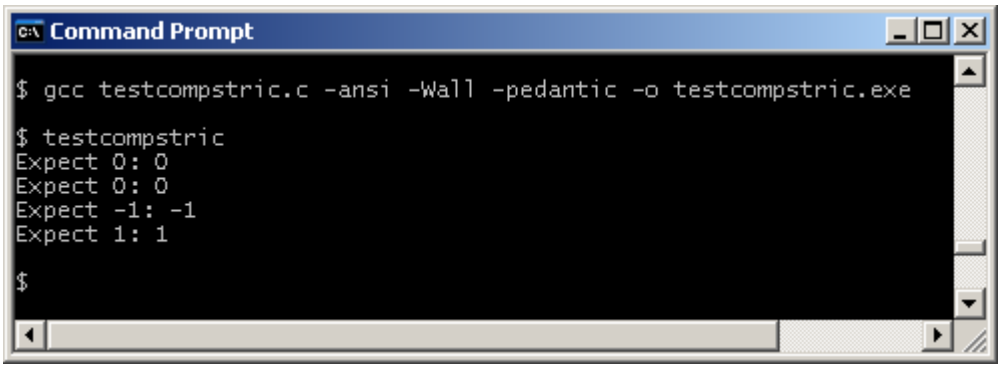
Compares each string char by char, returns immediately if two chars don't match. When finished looping, returns true (1) if the two last characters compared are identical, false (0) otherwise.

5.
```
/* equalStringsIgnoreCase: returns 1 if str1 and str2 have identical
elements, otherwise returns 0 irrespective of case */
int equalStrings(const char *str1, const char* str2)
{
  int i;

  char *s1 = stringToLowerCase(str1);
  char *s2 = stringToLowerCase(str2);

  for (i = 0; s1[i] != '\0'; i++)
    if (s1[i] != s2[i])
      return 0;
  return s1[i] == s2[i];
}
```

Converts each string to lower case. Then same process as for equalStrings().

6.
```
/* testcompstric.c: tests compareStringsIgnoreCase() */

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

/* stringToLowerCase: returns lower case version of string parameter */
char *stringToLowerCase(const char *string)
{
  char *lcString = malloc(strlen(string) + 1); /* +1 for '\0' */
  int i;

  for (i = 0; string[i] != '\0'; i++)
    lcString[i] = string[i];
  lcString[i] = '\0';

  for (i = 0; lcString[i] != '\0'; i++)
    lcString[i] = tolower(lcString[i]);

  return lcString;
}
```

```
/* compareStringsIgnoreCase: returns -1 if str1 < str2,
   0 if str1 == str2, +1 if str1 > str2 in alphabetical order */
int compareStringsIgnoreCase(const char *str1, const char *str2)
{
  int i;

  char *s1 = stringToLowerCase(str1);
  char *s2 = stringToLowerCase(str2);

  for (i = 0; s1[i] == s2[i]; i++)
    if (s1[i] == '\0')
      return 0;
  if (s1[i] < s2[i])
    return -1;
  else
    return 1;
}

int main()
{
  printf("Expect 0: %d\n",compareStringsIgnoreCase("Jo King","JO KING"));
  printf("Expect 0: %d\n",compareStringsIgnoreCase("JO KING","Jo King"));
  printf("Expect -1: %d\n",compareStringsIgnoreCase("JO KIN", "Jo King"));
  printf("Expect 1: %d\n",compareStringsIgnoreCase("JO KING", "Jo Kin"));

  return 0;
}
```



Converts both string paramters to lower case. Loops while equal chars are found in both arrays, returns 0 if the end of string is reached. On finishing looping, returns -1 or 1 dependingon whether the last pair of characters retrieved are in order or not.

Exercise 15.1

1.

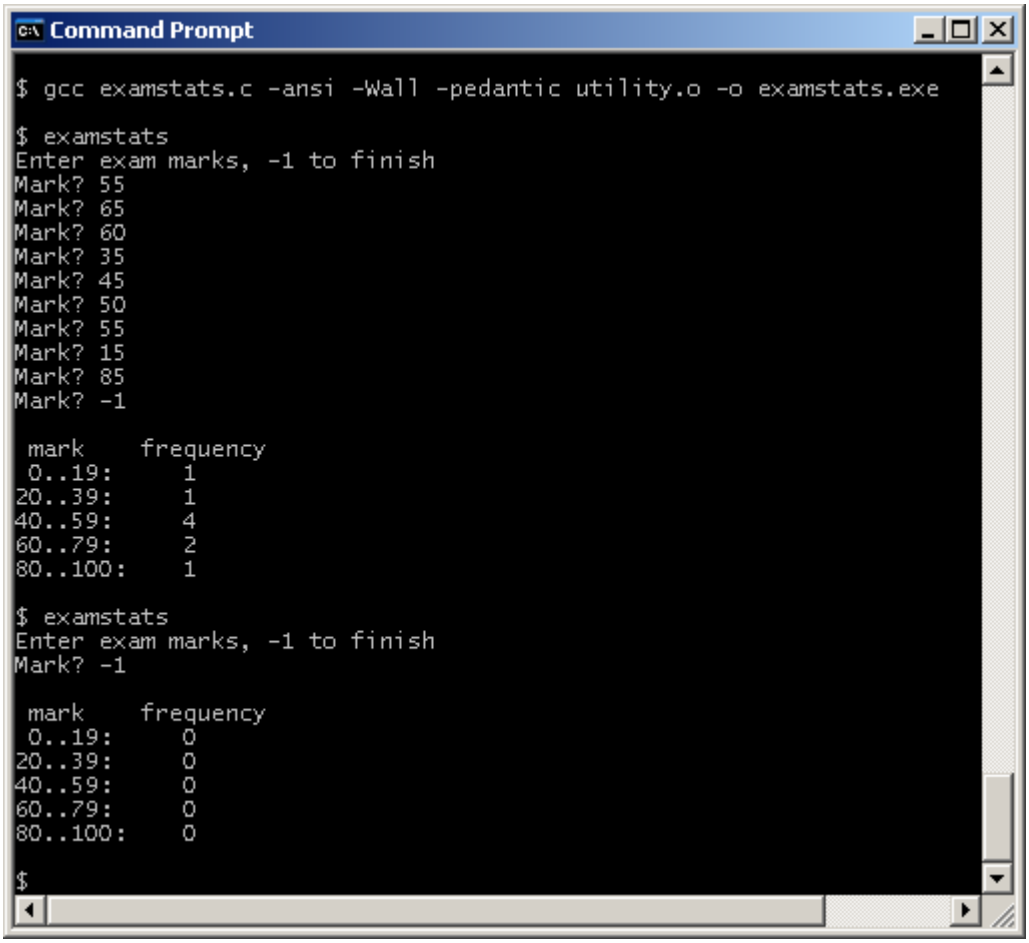
```
/* projstats.c: inputs project marks, outputs statistics */

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

/* getMarks: stores up to size-1 project marks */
int *getMarks(int size)
{
  int *marks = malloc(sizeof(int) * size);
  int i;

  for (i = 0; i < size; i++) {
    printf("Mark (-1 to end)? ");
    marks[i] = getInt();
    if (marks[i] < 0)
      break;
  }
  marks[i] = -1;
  return marks;
}
```

```
/* getMin: returns the least mark in marks */
int getMin(const int *marks, int size)
{
  int min = marks[0];
  int i;

  for (i = 0; marks[i] >= 0 && i < size; i++)
    if (marks[i] < min)
      min = marks[i];
  return min;
}

/* getMax: returns the largest mark in marks */
int getMax(const int *marks, int size)
{
  int max = marks[0];
  int i;

  for (i = 0; marks[i] >= 0 && i < size; i++)
    if (marks[i] > max)
      max = marks[i];
  return max;
}

/* getNStuds: returns the number of students */
int getNStuds(const int *marks, int size)
{
  int i;

  for (i = 0; marks[i] >= 0 && i < size; i++)
    ;
  return i;
}

/* getAverage:P returns the average mark */
double getAverage(const int *marks, int size)
{
  int numMarks = getNStuds(marks, size);
  int sumMarks = 0;
  int i;

  for (i = 0; marks[i] >= 0 && i < size; i++)
    sumMarks = sumMarks + marks[i];
  if (numMarks == 0)
    return 0.0;
  else
    return sumMarks / numMarks;
}

/* printStats: prints min, max and nStuds */
int printStats(int min, int max, int nStuds, double average)
{
  return printf("Min: %d, max: %d, students: %d, average: %0.1f\n",
           min, max, nStuds, average);
}

int main()
{
  enum { size = 16 };
  int *marks;
  int min, max, nStuds;
  double average;

  marks = getMarks(size);
  min = getMin(marks, size);
  max = getMax(marks, size);
  nStuds = getNStuds(marks, size);
  average = getAverage(marks, size);
  printStats(min, max, nStuds, average);
  return 0;
}
```

2.

```
/* projstats.c: inputs project marks, outputs statistics */

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

/* getMarks: stores up to size-1 project marks */
int *getMarks(int size)
{
  int *marks = malloc(sizeof(int) * size);
  int i;

  for (i = 0; i < size; i++) {
    printf("Mark (-1 to end)? ");
    marks[i] = getInt();
    if (marks[i] < 0)
      break;
  }
  marks[i] = -1;
  return marks;
}

/* getMin: returns the least mark in marks */
int getMin(const int *marks, int size)
{
  int min = marks[0];
  int i;

  for (i = 0; marks[i] >= 0 && i < size; i++)
    if (marks[i] < min)
      min = marks[i];
  return min;
}

/* getMax: returns the largest mark in marks */
int getMax(const int *marks, int size)
{
  int max = marks[0];
  int i;

  for (i = 0; marks[i] >= 0 && i < size; i++)
    if (marks[i] > max)
      max = marks[i];
  return max;
}

/* getNStuds: returns the number of students */
int getNStuds(const int *marks, int size)
{
  int i;

  for (i = 0; marks[i] >= 0 && i < size; i++)
    ;
  return i;
}

/* getAverage:P returns the average mark */
double getAverage(const int *marks, int size)
{
  int numMarks = getNStuds(marks, size);
  int sumMarks = 0;
  int i;

  for (i = 0; marks[i] >= 0 && i < size; i++)
    sumMarks = sumMarks + marks[i];
  if (numMarks == 0)
    return 0.0;
  else
    return sumMarks / numMarks;
}
```

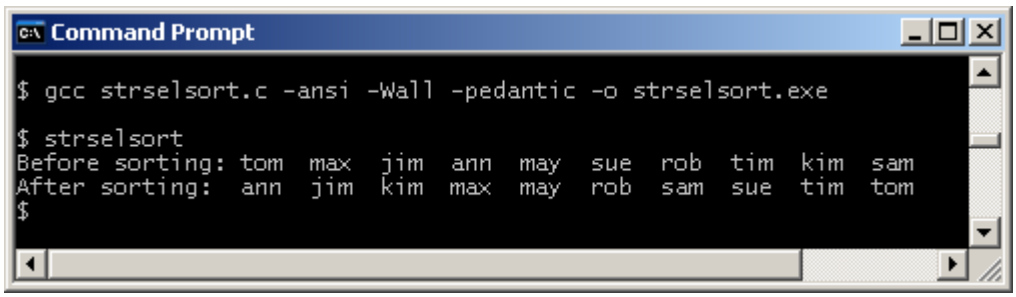
```
/* printStats: prints min, max and nStuds */
int printStats(int min, int max, int nStuds, double average)
{
  return printf("Min: %d, max: %d, students: %d, average: %0.1f\n",
                min, max, nStuds, average);
}

int main()
{
  enum { size = 16 };
  int *marks;
  int min, max, nStuds;
  double average;

  marks = getMarks(size);
  min = getMin(marks, size);
  max = getMax(marks, size);
  nStuds = getNStuds(marks, size);
  average = getAverage(marks, size);
  printStats(min, max, nStuds, average);

  return 0;
}
```



Exercise 16.1

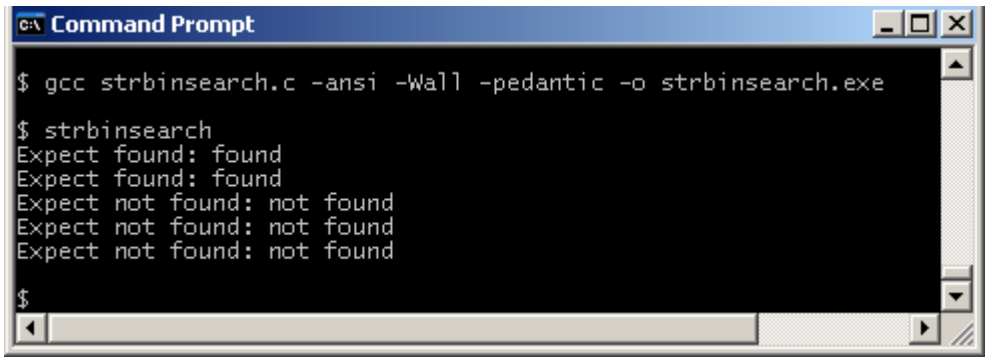
1.

```
/* strselsort.c: sorts array of strings into lexical order */

#include <stdio.h>
#include <string.h>

/* indexOfLargest: returns index of largest value in
   array[0]..array[lastInded] */
int indexOfLargest(char *array[], int lastIndex)
{
  int index = 1;
  int i;

  for (i = 0; i <= lastIndex; i++)
    if (strcmp(array[i], array[index]) > 0)
      index = i;
  return index;
}

/* strSwap: exchanges the values at index i and j in array */
int strSwap(char *array[], int i, int j)
{
  char *temp = array[i];   /* we don't use strcpy() here since */
  array[i] = array[j];     /* we are dealing with arrays of    */
  array[j] = temp;         /* pointers.  It is the pointers we */
  return 0;                /* copying                          */
}

/* strSelectionSort: sorts array into ascending order */
int strSelectionSort(char *array[], int arraySize)
{
  int index;
  int i;

  for (i = arraySize - 1; i > 0; i--) {
    index = indexOfLargest(array, i);
    strSwap(array, i, index);
  }
  return 0;
}

/* strPrintArray: displays contetns of array */
int strPrintArray(char *array[], int arraySize)
{
  int i;

  for (i = 0; i < arraySize; i++)
    printf("%s  ", array[i]);
  return 0;
}

int main()
{
  enum { size = 10 };
  char *array[size] = { "tom", "max", "jim", "ann", "may",
                        "sue", "rob", "tim", "kim", "sam" };

  printf("Before sorting: ");
  strPrintArray(array, size);
  strSelectionSort(array, size);
  printf("\n");
  printf("After sorting:  ");
  strPrintArray(array, size);

  return 1;
}
```



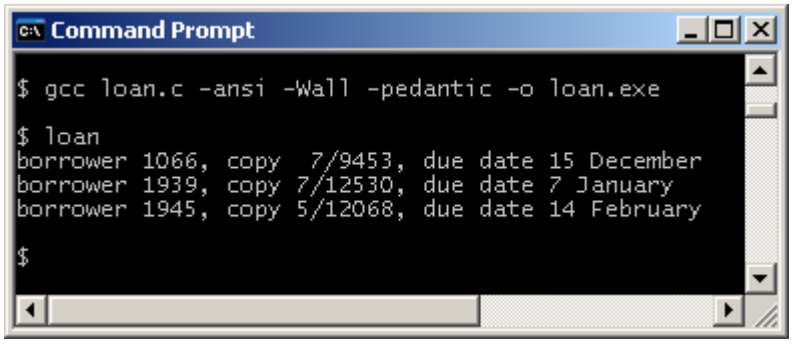
2.

```
/* strbinsearch.c: looks for an item in a sorted array of strings */

#include <stdio.h>
#include <string.h>

/* strBinarySearch: returns index where target is found in array,
   -1 otherwise.  Array must be sorted */
int strBinarySearch(char *array[], int arraySize, char *target)
{
  int low, high, middle;

  low = 0;
  high = arraySize - 1;
  while (low <= high) {
    middle = (low + high) / 2;
    if (strcmp(target, array[middle]) < 0)
      high = middle - 1;
    else if (strcmp(target, array[middle]) > 0)
      low = middle + 1;
    else
      return middle;
  }
  return -1;
}

int main()
{
  enum { size = 10 };
  char *array[size] = { "ann", "jim", "kim", "max", "may",
                        "rob", "sam", "sue", "tim", "tom" };

  printf("Expect found: ");
  strBinarySearch(array, size, "ann") < 0?
       printf("not found\n") : printf("found\n");
  printf("Expect found: ");
  strBinarySearch(array, size, "tom") < 0?
       printf("not found\n") : printf("found\n");
  printf("Expect not found: ");
  strBinarySearch(array, size, "pam") < 0?
       printf("not found\n") : printf("found\n");
  printf("Expect not found: ");
  strBinarySearch(array, size, "alf") < 0?
       printf("not found\n") : printf("found\n");
  printf("Expect not found: ");
  strBinarySearch(array, size, "vim") < 0?
       printf("not found\n") : printf("found\n");
  return 0;
}
```



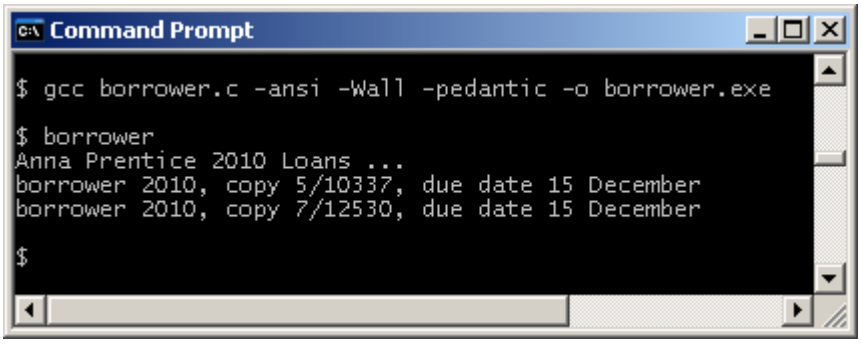
Exercise 17.1

1.
```
/* loan.c: deals with a single library loan */

#include <stdio.h>
#include <string.h>

typedef struct tagLoan {
  char borrowerNum[10];
  char accessionNum[10];
  char dueDate[20];
} Loan;

/* newLoan: creates a new loan */
Loan newLoan(char *borrower, char *accession, char *due)
{
  Loan loan;
  strcpy(loan.borrowerNum, borrower);
  strcpy(loan.accessionNum, accession);
  strcpy(loan.dueDate, due);
  return loan;
}

/* printLoan: displays a loan's attributes */
int printLoan(Loan loan)
{
  return printf("borrower %s, copy %s, due date %s\n",
         loan.borrowerNum, loan.accessionNum, loan.dueDate);
}

int main()
{
  Loan loan1 = newLoan("1066", "7/9453", "15 December");
  Loan loan2 = newLoan("1939", "7/12530", "7 January");
  Loan loan3 = newLoan("1945", "5/12068", "14 February");

  printLoan(loan1);
  printLoan(loan2);
  printLoan(loan3);
  return 0;
}
```



2.
```
/* borrower.c: records a borrower's loans */

#include <stdio.h>
#include <string.h>

typedef struct tagLoan {
  char borrowerNum[10];    /* duplicated in borrower */
  char accessionNum[10];
  char dueDate[20];        /* array of char is easiest solution */
} Loan;

typedef struct tagBorrower {
  char name[15];
  char number[10];
  Loan loan[5];            /* five loans maximum */
} Borrower;

/* newLoan: creates a new loan */
Loan newLoan(char *borrower, char *accession, char *due)
{
  Loan loan;
  strcpy(loan.borrowerNum, borrower);
  strcpy(loan.accessionNum, accession);
  strcpy(loan.dueDate, due);
  return loan;
}

/* printLoan: displays a loan's attributes */
int printLoan(Loan loan)
{
  return printf("borrower %s, copy %s, due date %s\n",
         loan.borrowerNum, loan.accessionNum, loan.dueDate);
}

/* newBorrower: creates a new borrower.
   A new borrower has no loans */
Borrower newBorrower(char *name, char *number)
{
  int size, i;
  Borrower borrower;

  strcpy(borrower.name, name);
  strcpy(borrower.number, number);

  size = sizeof borrower.loan / sizeof borrower.loan[0];
  for (i = 0; i < size; i++)
    borrower.loan[i] = newLoan("", "", "");
  return borrower;
}

/* addLoan: adds a loan to a borrower, but
   only if max loans has not been reached */
Borrower addLoan(Borrower borrower, Loan loan)
{
  int size, i;

  size = sizeof borrower.loan / sizeof borrower.loan[0];
  for (i = 0; i < size; i++) {
    if (strcmp(borrower.loan[i].accessionNum, "") == 0) {
      borrower.loan[i] = loan;
      break;
    }              /* no warning displayed if max loans has been reached */
  }
  return borrower;
}

/* printBorrower: displays a borrower's attributes and their loans */
int printBorrower(Borrower borrower)
{
  int size, i;
```

```
  printf("%s %s Loans ...\n", borrower.name, borrower.number);
  size = sizeof borrower.loan / sizeof borrower.loan[0];
  for (i = 0; i < size; i++)
    if (strcmp(borrower.loan[i].accessionNum, "") != 0)
      printLoan(borrower.loan[i]);
  return 0;         /* TODO: prints borrower number twice, needs fixing */
}

int main()
{
  Borrower borrower = newBorrower("Anna Prentice", "2010");
  Loan loan = newLoan("2010", "5/10337", "15 December");
  borrower = addLoan(borrower, loan);
  loan = newLoan("2010", "7/12530", "15 December");
  borrower = addLoan(borrower, loan);
  printBorrower(borrower);
  return 0;
}
```



3.

```
/* catalogue.c: creates a catalogue of book titles */

#include <stdio.h>
#include <string.h>

typedef struct tagTitle {
  char author[30];
  char title[30];
  int numberOfCopies;
} Title;

typedef struct tagCatalogue {
  char description[20];
  Title entry[10];  /* should have many more entries */
} Catalogue;

/* newTitle: creates a new title */
Title newTitle(char *author, char *bookTitle, int copies)
{
  Title title;
  strcpy(title.author, author);
  strcpy(title.title, bookTitle);
  title.numberOfCopies = copies;
  return title;
}

/* printTitle: prints title attributes */
int printTitle(Title title)
{
  return printf("%s %s %d\n",
       title.author, title.title, title.numberOfCopies);
}

/* newCatalogue: creates a new catalogue */
Catalogue newCatalogue(char *description)
{
  int size, i;
```

```
  Catalogue cat;
  strcpy(cat.description, description);
  size = sizeof cat.entry / sizeof cat.entry[0];
  for (i = 0; i < size; i++)
    cat.entry[i] = newTitle("", "", 0);
  return cat;
}

/* addTitle: adds title to catalogue.
   Assumes catalogue has space for the new title */
Catalogue addTitle(Catalogue cat, Title title)
{
  int size, i;
  size = sizeof cat.entry / sizeof cat.entry[0];
  for (i = 0; i < size; i++)
    if (strcmp(cat.entry[i].author, "") == 0) {
      cat.entry[i] = title;
      break;
    }
  return cat;
}

/* printCatalogue: displays titles in catalogue */
int printCatalogue(Catalogue cat)
{
  int size, i;

  printf("%s\n", cat.description);
  size = sizeof cat.entry / sizeof cat.entry[0];
  for (i = 0; i < size; i++)
    if (strcmp(cat.entry[i].author, "") != 0)
      printTitle(cat.entry[i]);
  return 0;
}

int main()
{
  Catalogue cat = newCatalogue("Loans Catalogue");
  Title title = newTitle(
       "Kernighan & Ritche", "The C Programming Language", 10);
  cat = addTitle(cat, title);
  title = newTitle("Mark Williams", "ANSI C A Lexical Guide", 5);
  cat = addTitle(cat, title);
  title = newTitle("Dromey R", "How to Solve it by Computer", 3);
  cat = addTitle(cat, title);
  printCatalogue(cat);
  return 0;
}
```