
Computer Programming: Skills & Concepts (INF-1-CP1)

Intro to Practical 1; more on `scanf`

8th October, 2009



Summary of Lecture 6

- General form of `if`-statement.
- Developing `quadratic.c` via nested `if`-statements.
- Boolean operators.
- Loops - the `while` and `for` statements.
- Fibonacci numbers.
- Prime numbers.

This Lecture

- The descartes graphics routines.
- Example: Square-drawing example using descartes routines.
- Discussion on Practical 1.
- `scanf` and erroneous input.

descartes.c

descartes.c is a set of small *functions* or *routines* which perform basic graphics tasks through a primitive graphics drawing tool.

- What is a *function* (in programming)?
It is an encapsulated and named section of code, which takes a number of parameters (or certain declared *types*), performs a sequence of C-statements, and returns a value of a declared *type*.

descartes.h

descartes.h contains the *type* declarations for the (non-native) *structured data types* and *functions* of descartes.c. But does NOT contain the *code* ...

```
/* A point is specified by its x- and y-coordinates. */  
typedef struct {int x, y;} point_t;
```

```
/* A line segment is specified by its endpoints. */  
typedef struct {point_t initial, final;} lineSeg_t;
```

```
/* Waits until the user clicks the left mouse button, then returns  
 * the point that the user is indicating. If the middle mouse button  
 * is clicked then the value returned is (-1, -1). */
```

```
point_t GetPoint(void);
```

```
/* Creates a point with given coordinates. */
point_t Point(int a, int b);

/* Returns the x-coordinate of the point given as argument. */
int XCoord(point_t p);

/* Returns the y-coordinate of the point given as argument. */
int YCoord(point_t p);

/* Creates a line segment with given endpoints. */
lineSeg_t LineSeg(point_t p1, point_t p2);

/* Returns one endpoint of a line segment... */
point_t InitialPoint(lineSeg_t l);
```

```
/* ... returns the other endpoint. */  
point_t FinalPoint(lineSeg_t l);  
  
/* Returns the length of a line segment. */  
float Length(lineSeg_t l);  
  
/* Draws a line segment. */  
void DrawLineSeg(lineSeg_t l);  
  
/* Opens and initialises the graphics window */  
void OpenGraphics(void);  
  
/* Closes the graphics window - actually waits for a right-mouse-click */  
void CloseGraphics(void);
```

Practical 1

- Part A (generalized Imperial to Metric distance converter) does not use the graphics tool.
- For Parts B-D, you should use the pre-programmed implementations of the functions of `descartes.h`. The code for these is in `descartes.c`.
- `/group/teaching/cp1/Proj1/` contains *completed* versions of `descartes.h` and `descartes.c`, and *mostly blank* versions of the files `convert.c`, `segment.c`, `rectangle.c` and `polygon.c`:
 - Do not edit `descartes.h` or `descartes.c`!!
 - Your C programs for Parts A, B, C, D should be written into `convert.c`, `segment.c`, `rectangle.c` and `polygon.c` respectively.

Part B: `segment.c`

Write a program which reads two points in the plane (specified as clicks on the graphics window), draws the line connecting these points, and calculates the distance between them.

Discuss: Which functions from `descartes.h` will be useful?

Part C: rectangle.c

Write a program which reads in two points from the plane (given as clicks on the graphics window), and then:

- (i) draws the implied rectangle,
- (ii) computes the length of its diagonal,
- (iii) classifies the shape of the rectangle as *almost square*, *wide* or *tall*.

Discuss: Which functions from `descartes.h` will be useful?

Part D: polygon.c

Write a program which reads in a sequence of points from the plane (given as clicks on the graphics window), and computes the perimeter of the polygon defined by those points.

Discuss: Which functions from `descartes.h` will be useful?

descartes example: Drawing a Square

Write a program which uses the descartes functions to load the graphics window, read one point (specified by a click) from this window, and draw a square of side-length 100 which has this point as its North-West corner.

Which descartes functions will we need? Discuss.
What variables will we define?

Drawing a Square

Steps of our program:

- Start up the Graphics window.
- Read in a point from that window.
- Draw the 4 edges of the square.
- Close the graphics window.

square.c - outline

```
#include <stdlib.h>
#include <stdio.h>
#include "descartes.h"
int main(void)
{
    point_t p, q;    /* Two point variables, */
    lineSeg_t pq;   /* One line segment variable */
    int x, y;       /* Two integers. */

    OpenGraphics(); /* Load graphics window. */
    printf("Indicate NW corner by clicking left mouse button.\n");
    p = GetPoint(); /* p stores the point where the user clicked. */
    .....         /* Draw 4 line segs, using LineSeg(,), DrawLineSeg(,) */
    CloseGraphics();
    return EXIT_SUCCESS;
}
```

square.c

```
#include <stdlib.h>
#include <stdio.h>
#include "descartes.h"

/* Draws a square, of side 100, with given NW corner */

int main(void)
{
    point_t p, q;      /* Two points,          */
    lineSeg_t pq;     /* a line segment     */
    int x, y;         /* and two integers.  */
    OpenGraphics();

    printf("Indicate NW corner by clicking left mouse button.\n");
```

```
p = GetPoint();    /* p stores the point where the user clicked. */
x = XCoord(p);    /* We can take a point apart          */
y = YCoord(p);    /* into its two coordinates...          */
q = Point(x + 100, y); /* and then reassemble.          */
pq = LineSeg(p, q); /* Two points define a line segment. */
DrawLineSeg(pq);  /* Let's have a look at what we've got. */

p = q;            /* Start where we left off.*/
x = XCoord(p);
y = YCoord(p);

q = Point(x, y - 100);
pq = LineSeg(p, q);
DrawLineSeg(pq);

/* We can construct these shifted points more tersely... */
```

```
p = q;  
q = Point(XCoord(p) - 100, YCoord(p));  
DrawLineSeg(LineSeg(p, q));  
  
p = q;  
q = Point(XCoord(p), YCoord(p) + 100);  
DrawLineSeg(LineSeg(p, q));  
  
CloseGraphics();  
return EXIT_SUCCESS;  
}
```

Makefile

```
CC          = /usr/bin/gcc
FLAGS      = -g -ansi -I/usr/X11R6/include -I/usr/include/srgrp -L/usr/X11R6/lib
            -Wall
LIBS       = -lm -lX11 -lsrgrp

descartes.o:  descartes.c descartes.h
              $(CC) $(FLAGS) -c descartes.c $(LIBS)

square:       square.c descartes.o
              $(CC) $(FLAGS) -o square descartes.o square.c $(LIBS)
```

To apply this ... type `make square` at the command line.
... if compilation succeeds, the executable gets saved in `square`
... then type `./square` to run

scanf - erroneous input

This bit not relevant for Practical 1

...

EXCEPT for Part A

scanf - erroneous input

What if the user types a word, when an integer is required?

Apart from the *action* performed by `scanf` (reading, or attempting to read, the object of the specified type), `scanf` *returns* an integer, which is the *number of input items assigned*. This may be fewer than provided for, or even zero, in the event of a matching failure.

This returned value can be used to test for a successful read:

```
scanf("%d", &a) == 1
```

if and only if an integer was successfully read into `a`.

scanf - error-checking our input

Suppose we want to read in an integer to `x`:

We can *test* for success by saving the returned value of `scanf`:

```
read_succ = scanf("%d", &x);
if (read_succ == 1) {
    ....
}
else {
    ....
}
```

What about the `else` branch?

- Print an error message and terminate?
- Can give the user a second try.

scanf error-checking - first attempt

```
printf("Please input an integer: ");
read_succ = scanf("%d", &x);
if (read_succ == 1) {
    ....
}
else {
    /* read_succ must have been 0 */
    printf("That wasn't an integer! Try again: ");
    read_succ = scanf("%d", &x);
    ....
}
```

PROBLEM: Guaranteed to fail ...

WHY?

scanf error-checking - “skipping over”

`scanf("%*s");` - means “skip over” first item in read-buffer from standard input (the `s` is for ‘string’, the `*` for ‘don’t save’).

```
printf("Please input an integer: ");
read_succ = scanf("%d", &x);
if (read_succ == 1) {
    ....
}
else {
    /* read_succ must have been 0 */
    scanf("%*s"); /* scan the bad-input, don't try to save it */
    printf("That wasn't an integer! Try again: ");
    read_succ = scanf("%d", &x);
    ....
}
```

scanf error-checking - loops

```
printf("Please input an integer: ");
read_succ = scanf("%d", &x);
if (read_succ != 1) {    /* read_succ must have been 0 */
    while (read_succ != 1) {
        scanf("%*s");    /* scan the bad-input, don't try to save it */
        printf("That wasn't an integer! Try again: ");
        read_succ = scanf("%d", &x);
    }
}
....                    /* Now we definitely have an int; do the work */
```

Examples - with fibonacci.c, fibonacci1.c, fibonacci2.c, fibonacci3.c,