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

5th October, 2010

Summary of Lecture 6

- `float` and `double`.
- The `marathon.c` program.
- Solving quadratic equations.
- General form of `if`-statement.
- Developing `quadratic.c` via nested `if`-statements.
- Boolean operators.

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);
```


descartes.h cont'd

```
/* 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);
```


descartes.h cont'd

```
/* 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 point where the user clicked. */
   ........       /* Draw 4 line segs - 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. */
```


square.c cont'd

```
   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/srgp
-L/usr/X11R6/lib -Wall
LIBS      = -lm -lX11 -lsrgp

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