CP coursework

Introduction

This is an assessed coursework. Some parts of this coursework depends on the `descartes` graphics library. To obtain credit for your work, you will need to electronically submit the most advanced version of your programs.

The deadline for completion and electronic submission of the CP coursework (via the `submit` command on DICE) is **4pm**, Friday 18th November (end week 9). Work submitted after the deadline will attract zero credit. If you have medical or other difficulties meeting the deadline, you may petition the Informatics Teaching Organisation

`https://www.inf.ed.ac.uk/cgi-bin/iss/contact.cgi`

in advance of the deadline (login with your DICE password). Course staff cannot grant extensions.

It is fine to generally discuss understanding of the coursework with your fellow students, or get help with debugging your code. It is also fine to ask Lecturers/demonstrators for general advice. However, you may *never* share code, nor copy code (manually or electronically) from others, in completing the task. *Your submission must be entirely your own work.*

Assessment

This coursework is the only *assessed coursework* for CP. It is worth 10% of your final grade for CP (the other 90% will come from the exam).

For CPMT students, this coursework is also worth 10% of the final mark; however, the exam contributes only 70% (and the final 20% comes from the project done with the Music Technology department in week 11).

The coursework is marked out of 100. Each part indicates how much it is worth.

Requirements Overall

- Do not add your own global variables to solutions.
- Remove any debugging printf statements before submitting your code.

Notes

- Read through this document *before* you reach the keyboard, and work out in advance what you need to do.
- If you are genuinely stuck, seek help immediately from your tutor or one of the course lecturers.

Electronic submission

There are five program files to be submitted: `bakery.c`, `uniform.c`, `weighted.c`, `square.c`, and `shade.c`. The specific form of the command for submitting these files is (from a Terminal window, logged-in on DICE):

```
submit cp 1 bakery.c uniform.c weighted.c square.c shade.c
```

You must submit **source code** (the .c files), **not** executable.

You can submit as many times as you like before the deadline. If you submit a file with the same name, it will overwrite the previous version. If you submit after the deadline, we won't be able to tell what you had at the time of the deadline.

Text Programs

Task 1: Bakery [20 marks]

You work at a bakery that makes cookies and bread. These are baked on regular intervals. For example, cookies might be baked every 3 hours and bread might be baked every 5 hours. Management wants to you to make a program that prints a baking schedule for a set number of hours. The bakery runs continuously: do not worry about days, just count hours (so hour 100 is valid).

Specifically, implement a function

```
void PrintSchedule(int cookie, int bread, int hours)
```

that prints a schedule with **exactly one line for each hour**:

- `Cookie` if cookies are baked that hour.
- `Bread` if bread is baked that hour.
- If neither cookies nor bread is baked, print the number of the hour.
- If both cookies and bread are made in the same hour, print `CookieBread`, our house specialty.

Assume that we just made a batch of delicious CookieBread at hour 0. Your schedule starts at the next hour, number 1.

As an example, `PrintSchedule(2, 3, 10)` prints

```
1
Cookie
Bread
Cookie
5
CookieBread
7
Cookie
Bread
Cookie
```

Notice how there are exactly 10 lines (the number of hours passed to the function), every second line has us bake a cookie, and every third line produces bread. When we do not bake anything, it prints the number of the hour. Remember that the hour count starts at 1. Your output should look exactly like the above example, with no additional spaces.

Use the following main function, which will get the cookie and bread rates from the user, as well as the number of hours.

```
int main() {
  int cookie, bread, hours;
  scanf("%d %d %d", &cookie, &bread, &hours);
  PrintSchedule(cookie, bread, hours);
  return EXIT_SUCCESS;
}
```

Name the source code of your program `bakery.c` and run `submit cp 1 bakery.c` to submit it.

Box of Chocolates

Life was like a box of chocolates. You never know what you're gonna get.

Our factory makes five different chocolates: Caramel, Milk, Sweet, Semisweet, and Dark. We make boxes with random assortments of chocolates in them. You are tasked with writing a program to randomly decide the contents of a box of chocolates.

To get a random number, make sure your program has `#include <stdlib.h>` near the top then use the `drand48()` function. This returns a `double` in the range $[0, 1)$. That means it can return 0, 0.462940, 0.99999 etc. but never exactly 1. To get different random numbers each time we run the program, we'll *seed* the random number generator from the time. Here's how:

```
#include <stdlib.h>
#include <stdio.h>
/* Necessary to get the time from the system */
#include <time.h>

int main() {
  /* Seed the random number generator */
  srand48(time(NULL));
  /* Prints a random value satisfying 0 <= value < 1 */
  printf("%lf\n", drand48());
  return EXIT_SUCCESS;
}
```

Task 2: Uniformly Popular [15 marks]

Assume that the five types of chocolate are equally popular. Write a program that randomly selects one of the names given earlier (Caramel, Milk, Sweet, Semisweet, or Dark) and prints the name on a single line.

Test your code by running it a number of times and ensuring it is capable of printing each type.

Name the source code of your program `uniform.c` and run `submit cp 1 uniform.c` to submit it.

Task 3: Weighted Popularity [25 marks]

Some types of chocolate are more popular than others. Write a program that prompts the user for the popularity of each type then picks chocolates in proportion to their popularity. Specifically, your program's input is 5 doubles, one per line, that indicate the popularity of each chocolate in order. Here is an example input:

```
0.01
0.5
900
3.2
300
```

In this case, Sweet is incredibly popular, Dark is one third as popular, Caramel is unpopular, and the rest are in between. Your program should print the name of exactly one chocolate variety on a single line. The variety should be chosen with probability proportionate to its popularity.

Here's how to do it. Sum up all the popularities. Multiply `drand48()` by this sum, storing the result in a value. Go through the varieties of chocolate one by one, subtracting the popularity from the value. When the value becomes negative, stop and print the name of the chocolate whose popularity made the value go negative. There are other ways to do this correctly, which we will also accept.

Name the source code of your program `weighted.c` and run `submit cp 1 weighted.c` to submit it.

Graphics Programs with `descartes`

These tasks have you work with the `descartes` drawing program. Recall that the points in the graphics window are those with co-ordinates from 0 to 499.

Preparation

This portion has associated files and templates.

To copy the files, go to the course webpage and click on `cwk.tar`. Or go directly to

`http://www.inf.ed.ac.uk/teaching/courses/cp/cwk16/cwk.tar`

Your browser should bring up a window listing the different files within the tar directory. Save each of these *into the particular directory* that you have created for the coursework. You will not be able to compile your code unless you have `descartes.h` and `descartes.o` in your working directory.

If you are not yet 100% confident in working with the `descartes` library, please return to the Labsheet for Lab 4 and make sure you can carry out all tasks in that Labsheet.

Remember to use the functions `OpenGraphics`, `CloseGraphics`, `GetPoint`, and `DrawLineSeg`. See Lab 4 for more on how to use these functions.

Compiling

Just like Lab 4, you compile with `-lSDL`. For example, `square.c` is compiled with

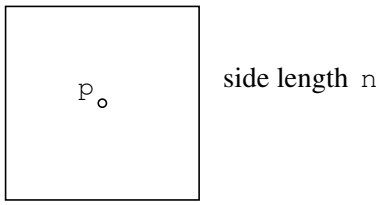
```
gcc -Wall square.c descartes.o -lSDL -lm
```

in the terminal window. The `-lSDL` option is to link to the `SDL library`, on which `descartes` depends.

Compiling is solely for your own testing. We want you to submit only the source code (`square.c` in this example).

Task 4: Drawing a Centered Square [20 marks]

In this task we will consider the problem of drawing a square where the reference point is now the *centre* of the square.[1] We will generalise this simple drawing task to allow for varying side lengths. Also, we will require the function to test that the desired square will lie within the boundaries of the graphics window. Here is a diagram showing the relationship of the square to the centre point:



To determine whether the entire square will fit, you will need to use the fact that the dimensions of the graphics window are 0 to 499 (inclusive) for both the x- and y-dimensions.

`square.c` contains the following function prototype for the `square` function:

```
int square (point_t p, int n);
```

Implement this function so that it draws a square of side length `n` with its centre at `p` *if the desired square will fit entirely within the graphics window.* If the square is drawn, the function should return `TRUE` (which has been defined as the value 1 in `square.c`); otherwise it should return `FALSE`.

You must implement the code within the `square` function, and not inside `main`. You should *not* use any global variables, and your `square` function should *not* use `printf`.

Testing the Centered Square

Compile with the `descartes` library, linking to `SDL`, to test your implementation:

```
gcc -Wall square.c descartes.o -lSDL -lm
```

The `main` program provided in `square.c` allows the user to carry out tests by clicking on the graphics window. For more accurate testing, you may want to write a different `main` function to call the `square` function with explicitly defined points and side lengths. For example:

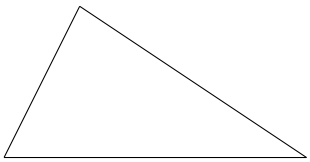
- Calling `square` with the point $(50, 70)$ and side-length 100 should return `TRUE`.
- Calling `square` with the point $(50, 70)$ and side-length 102 should return `FALSE`.
- Calling `square` with the point $(400, 200)$ and side-length 200 should return `FALSE`.
- Calling `square` with the point $(400, 200)$ and side-length 170 should return `TRUE`.

Name the source code of your program `square.c` and run `submit cp 1 square.c` to submit it.

[1] This is different to the case considered in the lecture slides for Lab 4 (in that case our clicked point was the NW corner).

Task 5: Shaded Triangle [20 marks]

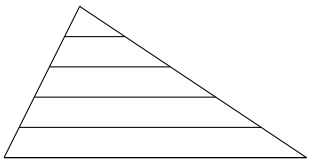
Write a program with `descartes` to draw a shaded triangle. Let's start by constraining the type of triangle. In this task, we'll assume that one leg of the triangle is horizontal. Here is an unshaded example.



More formally, the triangle consists of three points, but two of them have the same `y` coordinate.

Your program should solicit input from the user by calling `GetPoint()` three times. The third point will have whatever `y` coordinate the user happened to pick. Copy the second point's `y` coordinate and use it to overwrite the third point's `y` coordinate, thereby making the final leg horizontal. Ideally, your program will have a `triangle_t GetTriangle();` function that solicits this input and returns a custom `struct` called `triangle_t` of your own design. You can then write a `void DrawTriangle(triangle_t triangle);` function to draw a figure like the one above.

By shading, we mean drawing horizontal lines at regular intervals inside the triangle.



Write another function, `DrawShadedTriangle` that does much the same as `DrawTriangle`, except it shades the triangle with a specified number of horizontal lines. In the above figure, there are four additional horizonal lines.

As a hint, you should write a function

```
point_t PartWay(lineseg_t segment, double part);
```

which returns a point partway along a line segment. For example, if `part == 0.5`, then it would return the point exactly between the two ends of the line segment. Test this function first. Then intergrate it into your `DrawShadedTriangle` implementation.

Your final program will:

1. Ask the user how many shades they want with `scanf`.
2. Prompt for three points in the graphics window.
3. Draw the shaded triangle with the specified number of additional horizontal lines.

It is possible to solve this problem without using a `struct`, in which case partial marks may be awarded.

Name the source code of your program `shade.c` and run `submit cp 1 shade.c` to submit it.

Submission Reminder

Remember to submit all your files!
`submit cp 1 bakery.c uniform.c weighted.c square.c shade.c`
This can be done as often as you like.