COMP1511 18s2

Week-12 Laboratory Exercises

Topics

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

In particular, re-read (and/or watch the lecture video for) the lecture slides on the topics:

- "Recursion, Linked List with Recursion"

We will only assess lab Exercises 01, 02 and 04 to derive marks for this lab. However, you will learn a lot by attempting and solving the Exercise 03.

Getting Started

Create a new directory for this lab called `lab12` by typing:

```
$ mkdir lab12
```

Change to this directory by typing:

```
$ cd lab12
```

Exercise-01: Factorial - recursive function (individual)

Factorial is a common example of a recursive function.

Factorial is defined on the set of natural numbers, which we can approximate with `long long`, as

$$\begin{aligned} 0! &= 1 \\ n! &= n \times (n-1)! \end{aligned}$$

(The $0!$ is important!)

For this exercise, we'll implement a simple recursive factorial, which has this prototype:

```
long long factorial (long long n);
```

A recursive function should have a *base case*, and a *recursive case*; in the recursive case, the function should call itself, with different arguments.

Beware! Lots of resources will suggest you should just `return`. Our style guide forbids multiple returns. Consider how you define your base and recursive cases to avoid multiple returns.

Autotesting and submission instructions will be available later.

To run some simple automated tests:

```
$ 1511 autotest factorial
```

Submit your work with the *give* command, like so:

```
$ give cs1511 wk12_factorial
```

Exercise-02: A Divisive Issue (pair)

this activity was derived from *the Structure and Interpretation of Computer Programs*, by Abelson, Sussman, and Sussman; if you're interested in a very different take on computer science and on programming, you will enjoy this book!

The greatest common divisor, or GCD, of two integers a and b is the largest integer that divides both a and b with no remainder; for example, the GCD of 16 and 28 is 4 (as $16 = 4 \times 4$, and $28 = 4 \times 7$).

One way to calculate the GCD would be to totally factor both numbers, and find common factors, but there's a much faster and easier way to do it.

If r is the remainder when I divide a by b, then the common divisors of a and b are precisely the same as the common divisors of b and r, so the following identity holds:

$$\gcd(a, b) = \gcd(b, r)$$

If we start with any two positive integers, and applied this identity repeatedly, r will eventually become zero, and the other number in the pair is the greatest common divisor.

This is an amazing method known as Euclid's Algorithm, and is probably the oldest known non-trivial algorithm; it was first described in Euclid's *Elements* in around 300 BC.

For this exercise, you should create a file `gcd_rec.c`, and implement a function

```
int gcdRec (int a, int b);
```

which implements Euclid's algorithm as described above.

Autotesting and submission instructions will be available later.

To run some simple automated tests:

```
$ 1511 autotest gcd_rec
```

Submit your work with the *give* command, like so:

```
$ give  cs1511  wk12_gcd_rec
```

Exercise-03: Draw Bot (individual)

For this activity, you'll be creating a basic player for the Final Card-Down. If you have difficulties getting started or understanding any specific point(s), please discuss with your tutor.

We'll start out with the most basic player possible: one that simply draws a card, and then ends their turn, called Draw Bot.

Remember that it's always valid at the start of the game to draw a card (whether or not you have a valid card that you could play).

Write your first player for the Final Card-Down in a file called `drawBot.c`.

Base your code on the provided `drawBot.c` stub code:

Download drawBot.c,

However, Draw Bot isn't a very interesting player, since it only ever draws cards, and never plays cards, it doesn't stand a chance of winning the game.So, in this activity, you'll be trying to make Draw Bot a little bit more interesting. We have provided you with the following six stubs for helper functions that will help Draw Bot play the Final Card-Down more successfully. By the way, you may need additional helper functions also.

```
static int findMatchingCardColor (Game game, color color);
```

The `findMatchingCardColor` function finds a card in the player's hand that matches the specified color, if such a card exists. It returns the card index, or NOT_FOUND if no matching card was found.

```
static int doCardsMatch (Card first, Card second);
```

The `doCardsMatch` function compares two cards to determine whether they match on at least one of color, suit, or value. It returns TRUE if they match on any of the above features, and FALSE if they do not match on any of the above features.

```
static int canDrawCard (Game game);
```

The `canDrawCard` function determines whether the player can currently draw a card. It returns TRUE if they can, and FALSE if they can't.

(If they can't draw a card, they should probably end their turn?)

Once you have completed these functions, you might want to put together a better version of Draw Bot, which can do more than just drawing cards.

Don't forget about the `isValidMove` function, it's a handy way to work out before you play a move whether or not it will be valid (and you should only ever be making valid moves). You can also use `isValidMove` function to help you work out where you are at in the game.

Carefully read comments in the file `drawBot.c` for more information.

Do not include a main function in your file.

Autotesting and submission instructions will be available later.

To run some simple automated tests:

```
$ 1511 autotest drawBot
```

Submit your work with the *give* command, like so:

```
$ give cs1511 wk12_drawBot
```

Or, if you are working from home, upload the relevant file(s) to the `wk12_drawBot` activity on Give Online.

Exercise-04: Set ADT (pair)

Here's another ADT: the Set ADT. The Set ADT, like the Stack and Queue ADTs, model a concept you're likely to encounter elsewhere; a set is a cool mathematical construct, which stores a collection of unique values of the same type.

Download `Set.h`,

The Set ADT defines these methods in its interface:

- ```
  Set newSet (void);
  ```

  Create a new `Set`.

- ```
  void destroySet (Set);
  ```

  Release all resources associated with a `Set`.

- ```
  void setAdd (Set, item);
  ```

  Add an `item` to the `Set`. If the `item` already exists in the set, it does nothing.

- ```
  void setRemove (Set, item);
  ```

  Remove an `item` from the `Set`. If the `item` does not exist in the set, it does nothing.

- ```
  int setSize (Set);
  ```

  How many items are in the `Set`? (hint: store a counter in your `struct _set`)

- ```
  bool setContains (Set, item);
  ```

Does the `Set` contain this `item`? Returns `true` or `false`.

- ```
  Set setUnion (Set, Set);
  ```

  Take the union of two sets ($a \cup b$), and return the resulting set. The union of two sets is the set containing all the unique `item`s of both sets.

- ```
  Set setIntersection (Set a, Set b);
  ```

  Take the intersection of two sets ($a \cap b$), and return the resulting set. The intersection of two sets is the set containing all the `item`s that are common to both sets.

- ```
  bool setSubset (Set a, Set b);
  ```

  Is `a` a subset of `b` ($a \subseteq b$)? That is, does `a` contain all the `item`s that `b` cotains?

- ```
  bool setEqual (Set a, Set b);
  ```

  Returns `true` if `a` is equal to `b`, or `false` otherwise. If $a \subseteq b$ and $b \subseteq a$, then $a \equiv b$; Or, to put it another way, if `a` contains all the `item`s in `b`, and `b` contains all the `item`s in `a`, `a` and `b` are equal.

Create a file called `Set.c`; in it, you should implement these functions. You probably should use a linked list to store the different `item`s.

> Autotesting and submission instructions will be available later.

To run some simple automated tests:

```
$ 1511 autotest setADT
```

Submit your work with the *give* command, like so:

```
$ give cs1511 wk12_setADT
```

Or, if you are working from home, upload the relevant file(s) to the `wk12_setADT` activity on Give Online.