CAB202 Tutorial 2
Functions

Functions are the building blocks of complete programs. Much like how you wouldn't build a house from the roof down, programs are built by first creating functions, and then making use of them in a working program. Functions are used to represent a repeatable process (or algorithm) that adapts its behaviour based on the input arguments provided, and returns a result once completed. In the last session, you learnt about using conditional statements and loops to control the execution of code at runtime. By the completion of this tutorial, you will be able to use functions to represent repeatable segments of code, and know how to break complex problems down into smaller functions. This tutorial is worth 3% of your final mark for this subject.

Booleans in C

Booleans (variables which can either have the value of `true` or `false`) are not a native type in the C programming language. There are many different ways to introduce `boolean` like variables into C. For this unit, we will take advantage of the fact we are compiling against **`gnu99`** standards, and use the **`stdbool.h`** library. This library allows you to use Booleans (through the **`bool`** identifier) like they were a native type. The snippet below shows how to include them and use them in basic operation:

```
#include <stdbool.h>

int main() {
     bool my_boolean = true;
     if (my_boolean) {
           // Will do this
     } else if (!my_boolean) {
           // Will not do this
     }
}
```

Using "cab202_timers" from the ZDK

The ZDK includes a series of functions to provide timer-like operation. To use them, you need to include the appropriate header file, and call your desired function (in this tutorial we will only be using the **`timer_pause()`** function). For example, to pause for 5 seconds:

```
#include "cab202_timers.h"

int main() {
     // Pause for 5 seconds
     timer_pause(5000);
}
```

Non-Assessable Exercises

NOTE: These exercises are not assessable, and the tutor can help you with them. Assessable exercises are in the following section. Do NOT attempt to complete these exercises by reading pixels from the screen.

1. Draw the starting position of 5 racers in a game

The template provided in **`question_1.c`** on Blackboard is for a racing game where 5 players start on the left side of the screen and have to race to the right side of the screen. Before the race can begin, each player must be drawn in the leftmost column of the screen. Each player's character symbol is their player number. Players 1-5 must start in rows **4**, **8**, **12**, **16**, and **20** respectively

Write a function that draws a player at their start position with the following signature:

```
void draw_racer_at_start(int player_number);
```

If the variable **`player_number`** supplied is anything other than an **`int`** in the range 1 to 5, nothing is drawn. Only drawing should happen in this function (i.e. `show_screen()` should **not** be called)!

Challenge exercises:

- *Replace your solution with an analytical version (you should have nothing more than an* **`if`** *statement, with only one case, and a* ***single*** *draw call)!*
- *Extend your solution to work on any screen size (**Hint:** convert the values specified above to ratios of the screen height).*

2. Detect if a character is drawn within a box

The template provided in **`question_2.c`** on Blackboard is for a game where a player is trying to bomb (represented by an '**x**' character) a home base (represented by '**H**' characters). The edges of the home base go from 25% to 75% of both the screen width and screen height. In the logic of this game, there needs to be a function that evaluates whether the last bomb hit the base.

The demo code provided for this exercise has a game which repeatedly drops bombs at random locations until **`has_bomb_hit()`** returns `true`. At the moment, this function always returns `false`. To make the game work correctly, provide an implementation that returns `true` when the bomb coordinates hit a part of the base, and `false` otherwise.

Challenge exercises:

- *Modify the code so that the previous bomb is removed from the screen if it did not hit the base.*
- *Extend the game so that a bomb is dropped every time the space bar is pressed. Then update the behaviour so that the player only has 5 bombs. If all of the bombs are used, and none hit the base, instead present an alternate message telling the player that they lost.*

3. Detect a bullet hit in a space invaders style game

The template provided in **`question_2.c`** on Blackboard is for a space invaders game, where the bullet fires across the screen until it collides with an object. In this example consider a bullet (represented by a '**-**' character) that starts in the middle of the leftmost column (i.e. column is 0 and row is 50% of screen height) and shoots towards the right. At some point in its trajectory there will be a rock (represented by a '**#**' character).

The demo code provided for this exercise draws a rock at a random position and calls function **`shoot_rock()`**. Complete this function so that the bullet continues to the right until the rock is adjacent to the bullet (i.e. directly to the left of the rock).

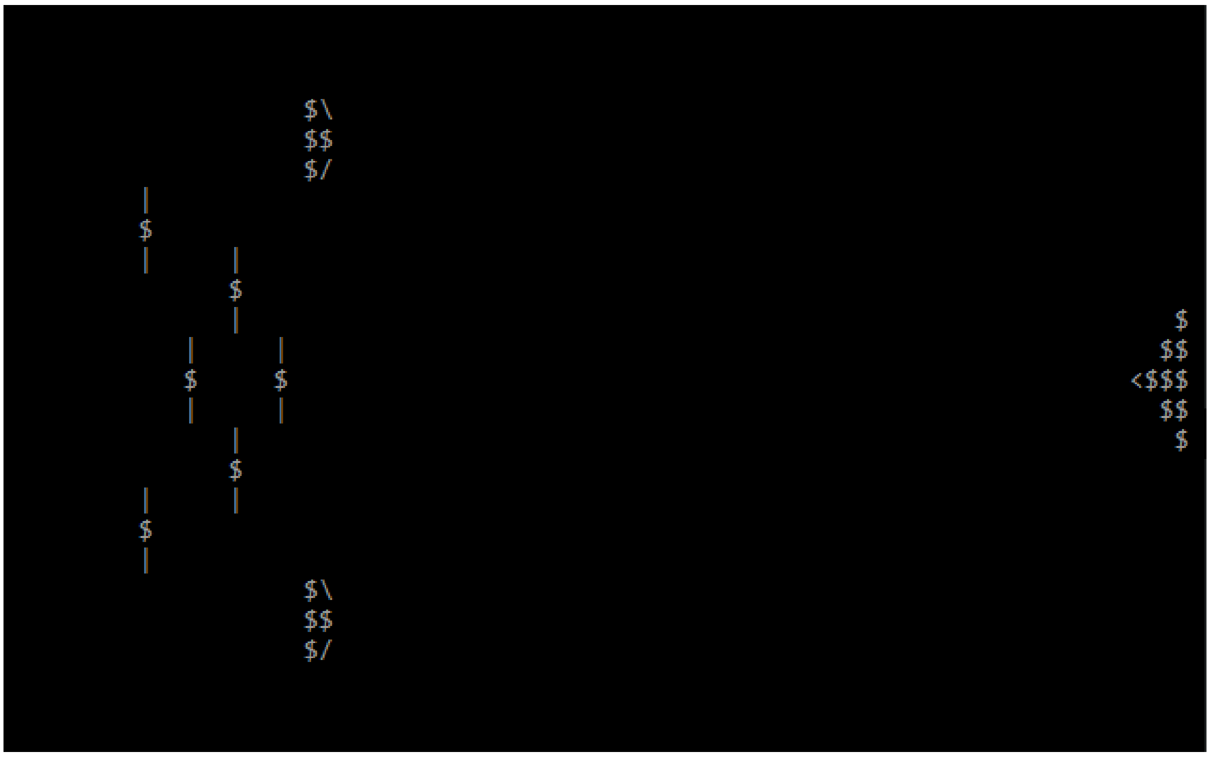
4. Investigate the scope of a variable in the previous solution

In the template used above for question 3, the variable named **`rock_x`** was used in a number of different places throughout the `*.c` file. Answer the following questions:

- How many times is the variable declared? Are you allowed to declare variables multiple times? What allows multiple declarations of the variable in this code?
- How come changing the value of the variables in within the **`draw_rock()`** function doesn't affect its value in **`main()`**? Make modifications to the code to verify this behaviour.
- Is it valid to declare variables outside functions? What happens if a variable is declared at the top of the source file after the `#include` directives? Why is declaring variables there often advised against?

5. Create 'space invaders' functions to work on top of 'cab202_graphics'

Imagine you are creating a space invaders game. The game will involve drawing the player's ship, and different types of enemies numerous times throughout the duration of a single game. This is a scenario where the use of functions, and possibly creating a library, are crucial programming decisions. Complete the following exercises in order, to slowly build up a more complete and reusable solution to draw the graphics for a space invaders game (the shape of the each of the ships is shown in the image below):



```
                     $\
                     $$
                     $/
         |
         $
         |     |
               $
               |                                                                $
            |     |                                                            $$
            $     $                                                          <$$$
            |     |                                                            $$
               |                                                                $
               $
         |     |
         $
         |
                     $\
                     $$
                     $/
```

1. Start with a **`question_5.c`** source file. Declare and implement the following 3 functions (the **x** and **y** arguments should correspond to the position of each ship's 'nose'):

```
draw_enemy_1(int x, int y);
```

```
draw_enemy_2(int x, int y);
draw_player(int x, int y);
```

2. Implement a `main` function that draws 6 type 1 enemies, 2 type 2 enemies, and 1 player (like the figure above). Compile and run your code to verify everything works properly.

Challenge exercises:

- *Combine your code with the solution from question 3. When the space bar is pressed, a bullet should shoot from the player's ship. Implement basic collision detection with the enemy ships directly in the path of the bullet.*
- *Create an **`invaders`** library (move your function declarations and implementations from above into **`invaders.h`** and **`invaders.c`** respectively). Your **`game.c`** file should now only have the `main` function. Add the necessary `#include` directive.*
- *Write a `Makefile`, or use a `gcc` compile command, to create **`libinvaders.a`** (look at the `Makefile` from the **ZDK**, or **`dummy_library`** for ideas).*

Assessed Exercises (AMS)

Complete the assessed exercises via the AMS (available at http://bio.mquter.qut.edu.au/CAB202).