



COMP 1045

Programming Fundamentals for Engineers

Assignment 1 – Programming with C

Contents

Introduction

Graduate Qualities

Assignment Overview

Practical Requirements

Stages

Submission Details

Extensions and Late Submissions

Academic Misconduct

Marking Criteria

Sample Output

INTRODUCTION

This document describes the first assignment for Programming Fundamentals for Engineers.

The assignment is intended to provide you with the opportunity to put into practice what you have learnt in the course by applying your knowledge and skills to the implementation of a game called **Dice Poker**.

This assignment is an **individual task** that will require an **individual submission**.

If you are an **internal student**, you will be required to submit your work via learnonline **before Monday 6 May (week 8), 3.00pm (internal students)**. **You will also be required to present your work to your practical supervisor during your practical session held in week 8 of the study period. Important:** You must attend the practical session that you have been attending all study period in order to have your assignment marked.

If you are an **external student**, you are only required to submit an electronic copy of your program via learnonline **before Friday 10 May (week 8), 11:30pm (external students only)**. External students are **not** required to demonstrate in person.

This document is a kind of specification of the required end product that will be generated by implementing the assignment. Like many specifications, it is written in English and hence will contain some imperfectly specified parts. Please make sure you seek clarification if you are not clear on any aspect of this assignment.

GRADUATE QUALITIES

By undertaking this assessment, you will progress in developing the qualities of a University of South Australia graduate.

The Graduate qualities being assessed by this assignment are:

- The ability to demonstrate and apply a body of knowledge (GQ1) gained from the lectures, practicals and readings. This is demonstrated in your ability to apply problem solving and programming theory to a practical situation.
- The use of communication skills (GQ6) by producing code that has been properly formatted; and writing adequate, concise and clear comments.

You will also progress in developing the following graduate qualities:

- The development of skills required for lifelong learning (GQ2), by searching for information and learning to use and understand the resources provided (C standard library, lectures, practical exercises, etc); in order to complete a programming exercise.
- The ability to effectively problem solve (GQ3) using C to complete the programming problem. Effective problem solving is demonstrated by the ability to understand what is required, utilise the relevant information from lectures, readings and practical work, write C code, and evaluate the effectiveness of the code by testing it.
- The ability to work autonomously (GQ4) in order to complete the task.
- The application of international standards (GQ7) by making sure your solution conforms to the standards presented in the C Programming Practice slides (available on the course website).

ASSIGNMENT OVERVIEW

You are required to write a C program that allows a player to play a **game of Dice Poker** against the computer. The program will allow a player to play as many games of dice poker as they wish. Once the player chooses to stop playing, the program will report the game summary to the screen.

Dice Poker





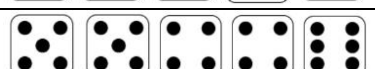


Poker is a popular card game where the goal is to beat the dealer (and other players) by having the hand which out ranks all other hands. <https://en.wikipedia.org/wiki/Poker> Typically, each player is dealt five playing cards called a hand. Each hand is then compared against the other players in order to determine the winning hand, that is, the hand that out ranks the other hands. Each hand has a rank and is ranked according to the rules of poker: https://en.wikipedia.org/wiki/List_of_poker_hands. This card game translates well into a dice game. Instead of a deck of 52 cards, you will use five, six sided dice in order to play the game.

Although there are many variations of the rules and game play of dice poker, we will be adhering to the following rules and game play for the assignment.

You are required to write a C program that allows a player to play a **game of dice poker** against the computer (i.e. the dealer). The program will allow a player to play as many games of dice poker as they wish. Once the player chooses to stop playing, the program will report the game summary to the screen.

Dice Poker Game Play and Rules:

- To begin, the player and the dealer (i.e. the computer) are dealt a hand, that is, they each roll five dice.
- The player and the dealer's hands are displayed to the screen.
- The hands are then ranked on a scale from 0 to 6 according to the following ranking rules:

Rank Number	Rank Name	Description	Example
Rank 6	Five of a kind	All five dice have the same face value.	
Rank 5	Four of a kind	Four dice have the same face value with one die having a different value.	
Rank 4	Full house	Three of a kind and a pair.	
Rank 3	Three of a kind	Three dice have the same face value and the other two have different values.	
Rank 2	Two pairs	Two pairs with one die having a different value.	
Rank 1	One pair	Two dice have the same value and the others have different values.	
Rank 0	Nothing special	All five dice have different values.	

- The rankings are displayed to the screen as text, i.e. Full house, Three of a kind, etc.
- The player with the highest ranked hand wins the game! If both players have equal ranked hands, the game is a draw.
- Once the player no longer wishes to play against the dealer (computer), the game summary is displayed to the screen.

You do not have to write the code that displays the die face values to the screen, a function that does that for you has been provided. The function is called `display_hand`. You are required to use this as part of this assignment, however, **please do not modify the `display_hand` function.**

PRACTICAL REQUIREMENTS

It is recommended that you develop this assignment in the suggested stages.

It is expected that your solution will include the use of:

- Your solution in a file called `yourEmailId_dice_poker.c`. For example: `bonjy007_dice_poker.c`
- Appropriate and well constructed `while/for` loops. (Marks will be lost if you use `break`, `exit`, `return` or similar statements in order to exit from within loops).
- Appropriate and well constructed `if`, `else if`, `else` statements (as necessary).
- The supplied `display_hand(int hand[], int max_dice)` function. This is provided for you – **please DO NOT modify this function**.
- The use of the `rand()` function in order to simulate the roll of a six sided die.
- Five functions (please refer to **stage 10** for description of functions):
 - `void display_details(void);`
 - `int roll_die(void);`
 - `void deal_hand(int hand[], int max_dice);`
 - `int rank_hand(int hand[], int max_dice);`
 - `void display_rank(int rank);`
- Array of integers to represent the player's hand, i.e. `int player_hand[5];`
- Array of integers to represent the dealer's (computer's) hand, i.e. `int dealer_hand[5];`
- Output that **strictly** adheres to the assignment specifications. If you are not sure about these details, you should check with the 'Sample Output' provided at the end of this document or post a message to the discussion forum.
- No global variables.
- Appropriate functions – 1 idea per function.
- Use of `#define` for symbolic constants (if necessary). No magic numbers.
- Good programming practice:
 - Consistent commenting and indentation. You are to provide comments to describe: your details, program description, **all** variable definitions, **all** function prototypes and **all** function definitions and every significant section of code.
 - Meaningful variable names (no single letter identifier names).
 - Consistent code indentation and layout.

NOTE: **Do not** use `break`, `exit` or `continue` statements for program control in your solution – doing so will result in a significant mark deduction.

PLEASE NOTE: You are reminded that you should ensure that all input and output conform to the specifications listed here; if you are not sure about these details you should check with the sample output provided at the end of this document or post a message to the discussion forum in order to seek clarification.

The layout and presentation of your output should **EXACTLY** match the sample output provided (refer to the end of this document).

STAGES

It is recommended that you develop this assignment in the suggested stages. Many problems in later stages are due to errors in early stages. **Make sure you have finished and thoroughly tested each stage before continuing.**

The following stages of development are recommended:

Stage 0 (preparation)

Start by setting up the project correctly. Please refer to the instructions on the course website if you are unsure of how to do this.

You will need the file that contains the `display_hand(int hand[], int max_dice)` function for this assignment. This has been provided for you. Please download this file from the course website (Assessment tab). The file is called `yourEmailId_dice_poker.c`. Change `yourEmailId` to your actual email id.

You will need to add the file (called `yourEmailId_dice_poker.c`) to your project (Select the Project menu, then Add existing item...). Before adding it to your project, ensure that you copy the file into the project folder.

Test to ensure that this is working correctly by entering the following within the main function of your `yourEmailId_game.c` file. For example:

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* Function display_dice to display the face value of dice to the screen.
You are provided with this function (see below) - please do NOT modify it. : ) */
void display_hand(int hand[], int max_dice);

int main() {
    int player_hand[5] = {2, 3, 4, 5, 6};

    printf("\n\nProgramming assignment - this will be deleted in stage 1!\n\n");

    /* Display player's hand to the screen. */
    printf("\n\nPlayer's hand:");
    display_hand(player_hand, 5);

    return 0;
}
```

Run the `yourEmailId_dice_poker.c` file. If this is working correctly, you should now see the following output when you run your program:

```
Programming assignment - this will be deleted in stage 1!
```

```
Player's hand:
```

Die 1	Die 2	Die 3	Die 4	Die 5
[2]	[3]	[4]	[5]	[6]
*	*	* *	* *	* *
	*		*	* *
*	*	* *	* *	* *

Note, this is for developmental purposes only, and you will need to modify and correctly position the above code. Make sure the program compiles and runs correctly. Once you have that working, back up your project.

Note: When developing software, you should always have fixed points in your development where you know your software is bug free and runs correctly.

Stage 1

Create an array to store the player's hand, i.e. the five dice values. For example:

```
int player_hand[5] = {0};
```

Add code to simulate the rolling of five dice, i.e. deal the player's hand. That is, you should generate five random numbers, each in the range of 1 through 6 (inclusive) and assign each number to an array element. Use the `rand()` function (i.e. `1 + rand() % 6`) to simulate the roll of a die. Hint: Use a loop in order to assign each random number (die roll) to an array element.

Display the player's hand (i.e. the randomly generated dice roll) to the screen, e.g.:

```
:
/* Place code to randomly generate the roll of five dice here... */
:
:
/* Display player's hand to the screen. */
printf("\n\nPlayer's hand:");
display_hand(player_hand, 5);
```

Sample output (this will look different given we are generating random values here):

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[5]	[2]	[1]	[1]	[6]
* *	*			* *
*		*	*	* *
* *	*			* *

You may like to implement the `void deal_hand(int hand[], int max_dice)` function in this stage or you may prefer to wait until stage 10 (see stage 10 for function description).

Make sure the program runs correctly. Once you have that working, back up your program. *Note: When developing software, you should always have fixed points in your development where you know your software is bug free and runs correctly.*

Stage 2

Add code to count how many times each die face value was rolled. Hint: Use an array in order to store this information.

To define an array in order to count how many times each die face value was rolled:

```
int die_count[7] = {0};
```

die count						
0	0	0	0	0	0	0
0	1	2	3	4	5	6

Given that die face values are 1 – 6 inclusive, we create an array with seven elements but ignore the zero element of the array. This will make it easier to increment the appropriate array element. That is, `die_count[1]` should contain the number of 1s that were rolled, `die_count[2]` the number of 2s rolled, etc.

For example: In the example provided in stage 1, the player's hand is assigned the following dice values: 5, 2, 1, 1, 6. In light of this, the `die_count` should be as follows:

die_count						
0	2	1	0	0	1	1
0	1	2	3	4	5	6

To access and update the appropriate array element (using the value of the die as an index):

```
int die_value = player_hand[0];
die_count[die_value] = die_count[die_value] + 1;
```

For example: If `die_value` is assigned the value 3.

die_count						
0	0	0	1	0	0	0
0	1	2	3	4	5	6

Hint: Use a loop in order to count how many times each die face value was rolled. In the example above, this would mean that `player_hand[0]` would then be `player_hand[index]` if placed within a loop.

Stage 3

Determine the rank of the player's hand (i.e. an integer value between 0 and 6 as described in the section 'Dice Poker Game Play and Rules'). Hint: Use the `die_count` array that holds how many times each die face value was rolled in order to determine the rank of the hand.

You may like to implement the `int rank_hand(int hand[], int max_dice)` function in this stage or you may prefer to wait until stage 10 (see stage 10 for function description).

Stage 4

Given the rank value ascertained in stage 3, display the corresponding rank name to the screen (see section 'Dice Poker Game Play and Rules' for rank names).

You may like to implement the `void display_rank(int rank)` function in this stage or you may prefer to wait until stage 10 (see stage 10 for function description).

Stage 5

Add code to simulate the dealer's hand (i.e. computer). That is, repeat steps 1 – 4 in order to do this for the dealer.

Stage 6

Add code to determine whether the player wins, loses or draws with the dealer (computer). Display the appropriate message to the screen, i.e.:

- `** Player wins! **`
- `** Dealer wins! **`
- `** Draw! **`

Stage 7

Now... it's time to allow the player to play more than one game. Let's add a loop which loops until the user enters 'n' (to stop playing the game). Think about where this code should go – what needs to be repeated, etc.

Stage 8

Add code to keep track of how many games were played, won, lost and drawn. Display this to the screen (as seen in the sample output at the end of this handout).

Stage 9

Add code to validate **all** user input:

- Would you like to play dice poker [y|n]?

Sample output:

```
Would you like to play dice poker [y|n]? p
Please enter either 'y' or 'n'.
```

```
Would you like to play dice poker [y|n]? y
```

- Play again [y|n]?

Sample output:

```
Play again [y|n]? z
Please enter either 'y' or 'n'.
```

```
Play again [y|n]? n
```

Stage 10

Modify your code to include and make use of the following **five** functions:

- `void display_details(void);`
- `int roll_die(void);`
- `void deal_hand(int hand[], int max_dice);`
- `int rank_hand(int hand[], int max_dice);`
- `void display_rank(int rank);`

1. Write a function called `display_details()` that will display your details to the screen. The function takes no parameters and does not return a value. The function simply displays your details to the screen. Your function should produce the following output (with your details).

Output:

```
File      : wayby001_dice_poker.c
Author    : Batman
Stud ID   : 0123456X
Email ID  : wayby001
This is my own work as defined by the
University's Academic Misconduct Policy.
```

2. Write a function called `roll_die()` that simulates the rolling of one die. The function will generate a random number in the range 1 – 6 (the face value on the die are 1 – 6 inclusive). The function takes no parameters and returns the random number generated.

For example:

```
die = roll_die();
```

3. Write a function called `deal_hand()` that takes an array of integers (i.e. player/dealer hand) and the size of the array as parameters. The function generates five random numbers, each in the range of 1 through 6 (inclusive) and assigns each number to an array element, i.e. deals the player/dealer hand. The function must call the `roll_die()` function to simulate the roll of a die. Hint: Use a loop in order to assign each random number (die roll) to an array element. The function does not return a value. Remember that arrays are passed call-by-reference to functions, this means that any changes you make to the array parameter within the function modifies the original array passed in as an argument.
4. Write a function called `rank_hand()` that takes an array of integers (i.e. player/dealer hand) and the size of the array as parameters. The function determines the rank of the hand based on the combination of dice values stored in the array. The function returns an integer value between 0 and 6 (inclusive) depending on the combination of dice values as described in the section 'Dice Poker Game Play and Rules'.
5. Write a function called `display_rank()` that takes an integer value representing the rank (number) of the hand as a parameter. The function displays the corresponding rank name to the screen based on the rank number being passed in as a parameter. See section 'Dice Poker Game Play and Rules' for rank names, e.g. If the rank number passed in as a parameter is 5, the function should display 'Four of a kind' to the screen; if the rank number is 4, the function should display 'Full house' to the screen, etc. The function does not return a value.

Reminder:

Defining a function does not execute the function – you will need to call the function(s) from the appropriate place(s) from within the main function in the program (some functions, more than once).

Stage 11 – THIS IS IMPORTANT!

Finally, check the **sample output** (see section titled 'Sample Output') and if necessary, modify your code so that:

- The output produced by your program **EXACTLY** matches the sample output provided.
- Your program **EXACTLY** behaves as described in these specs **and** the sample output provided.

Stage 12

Now that you know your code is working – why not add code that will allow more randomness between runs.

To see more randomness between runs you can use the seed which is available as part of the C library. This seed value gives the random number generator a new starting point. You can input any integer you like to this function as a seed. Start your program by calling the time function (which returns the time of day in seconds) and passing the value it returns to function `srand()` - this will seed the random number generator for you. **You only need to do this once at the start of your program.**

For a different sequence each run:

```
#include <time.h>

srand(time(NULL));
```

SUBMISSION DETAILS

You are required to do the following in order to submit your work and have it marked:

- Internal students:
 - If you are an internal student, you are required to submit an electronic copy of your program via learnonline **before Monday 6 May (week 8), 3:00pm (internal students)**. No further changes can be made to your assignment once submitted.
 - **Internal students are also required to demonstrate your assignment to your practical supervisor during your week 8 practical class for marking. The supervisor will mark your work using the marking criteria included in this document. You MUST attend the practical session that you have been attending all study period in order to have your assignment marked.**

Assignments submitted to learnonline, but not demonstrated during your allocated practical session, will NOT be marked. Likewise, assignments that have been demonstrated during the practical session, but have not been submitted via learnonline, will NOT be marked. Assignments are submitted to learnonline in order to check for plagiarism.

- External students:
 - If you are an external student, you are only required to submit an electronic copy of your program via learnonline **before Friday 10 May (week 8), 11:30pm (external students)**. External students are **not** required to demonstrate in person.

All students (internal and external) must follow the submission instructions below:

The solution file should be called `yourEmailId_dice_poker.c`. For example: `bonjy007_dice_poker.c`. Ensure that your file is named correctly (as per instructions outlined in this document).

Ensure that the following file is included in your submission:

- `yourEmailId_dice_poker.c`

All files that you submit must include the following comments.

```
File      : wayby001_dice_poker.c
Author    : Batman
Stud ID   : 0123456X
Email ID  : wayby001
This is my own work as defined by the
University's Academic Misconduct Policy.
```

Assignments that do not contain these details may not be marked.

You must submit your program **before** the online submission due date. You will also be required to demonstrate that you have correctly submitted your work to learnonline. Work that has not been correctly submitted to learnonline will not be marked. No further changes can be made to your assignment once submitted.

It is expected that students will make copies of all assignments and be able to provide these if required.

EXTENSIONS AND LATE SUBMISSIONS

There will be **no** extensions/late submissions for this course without one of the following exceptions:

1. A medical certificate is provided that has the timing and duration of the illness and an opinion on how much the student's ability to perform has been compromised by the illness. **Please note** if this information is not provided the medical certificate WILL NOT BE ACCEPTED. Late assessment items will not be accepted unless a medical certificate is presented to the Course Coordinator. The certificate must be produced as soon as possible and must cover the dates during which the assessment was to be attempted. In the case where you have a valid medical certificate, the due date will be extended by the number of days stated on the certificate up to five working days.
2. A Learning and Teaching Unit councillor contacts the Course Coordinator on your behalf requesting an extension. Normally you would use this if you have events outside your control adversely affecting your course work.
3. Unexpected work commitments. In this case, you will need to attach a letter from your work supervisor with your application stating the impact on your ability to complete your assessment.
4. Military obligations with proof.

Applications for extensions must be lodged with the Course Coordinator before the due date of the assignment.

Note: Equipment failure, loss of data, 'Heavy work commitments' or late starting of the course are not sufficient grounds for an extension.

ACADEMIC MISCONDUCT

ACADEMIC MISCONDUCT

Students are reminded that they should be aware of the academic misconduct guidelines available from the University of South Australia website.

Deliberate academic misconduct such as plagiarism is subject to penalties. Information about Academic integrity can be found in Section 9 of the *Assessment policies and procedures manual* at:

<http://www.unisa.edu.au/policies/manual/>

MARKING CRITERIA

Other possible deductions:

- *Programming style:* Things to watch for are poor or no commenting, poor code layout, poor variable names, etc.
- *Submitted incorrectly:* -5 marks if assignment is submitted incorrectly (i.e. not adhering to the specs).



Programming Fundamentals for Engineers (COMP 1045).
C Programming Assignment - Weighting: 15% - Due: Week 8, 2019

NAME:	MAX MARK	MARK	COMMENT
<p>PRODUCES CORRECT RESULTS (OUTPUT)</p> <pre> File : wayby001_dice_poker.c Author : Batman Stud ID : 0123456X Email ID : wayby001 This is my own work as defined by the University's Academic Misconduct Policy. Would you like to play dice poker [y/n]? y Player's hand: Die 1 Die 2 Die 3 Die 4 Die 5 [6] [2] [6] [3] [6] ** * ** * ** ** * ** * ** ** ** * ** Dealer's hand: Die 1 Die 2 Die 3 Die 4 Die 5 [5] [4] [1] [5] [5] ** ** * ** ** * * * * ** ** ** ** ** -- Player has Three of a kind -- Dealer has Three of a kind ** Draw! ** Play again [y/n]? n Game Summary ===== You played 1 games: --> Games won: 0 --> Games lost: 0 --> Games drawn: 1 Thanks for playing! ----- No worries... another time perhaps... :) Five of a kind Four of a kind Full house Three of a kind Two pairs One pair Nothing special ** Player wins! ** ** Dealer wins! ** ** Draw! ** </pre>	20 marks		<div> <input type="checkbox"/> -2 No or incorrect general line spacing </div> <div> <input type="checkbox"/> -1 No details displayed to screen </div> <div> <input type="checkbox"/> -1 No or incorrect play prompt </div> <div> <input type="checkbox"/> -1 No or incorrect player's hnd display </div> <div> <input type="checkbox"/> -1 No or incorrect dice display </div> <div> <input type="checkbox"/> -1 No or incorrect dealer's hnd display </div> <div> <input type="checkbox"/> -1 No or incorrect dice display </div> <div> <input type="checkbox"/> -1 No or incorrect player hand msg </div> <div> <input type="checkbox"/> -1 No or incorrect dealer hand msg </div> <div> <input type="checkbox"/> -1 No or incorrect win/lose/draw msg </div> <div> <input type="checkbox"/> -1 No or incorrect play again prompt </div> <div> <input type="checkbox"/> -1 No or incorrect game summary layout and/or text </div> <div> <input type="checkbox"/> -1 No or incorrect summary values </div> <div> <input type="checkbox"/> -1 No or incorrect another time msg </div> <div> <i>(Max. -5 mark deduction for rank name)</i> <input type="checkbox"/> -1 No or incorrect five msg </div> <div> <input type="checkbox"/> -1 No or incorrect four msg </div> <div> <input type="checkbox"/> -1 No or incorrect full msg </div> <div> <input type="checkbox"/> -1 No or incorrect three msg </div> <div> <input type="checkbox"/> -1 No or incorrect two msg </div> <div> <input type="checkbox"/> -1 No or incorrect one msg </div> <div> <input type="checkbox"/> -1 No or incorrect nothing msg </div> <div> <input type="checkbox"/> -2 No or incorrect win/lose/draw msgs </div>

<p>ADHERES TO SPECIFICATIONS (CODE)</p> <p>Use of <code>1 + (rand() % 6)</code> or similar for die roll</p> <p>Appropriate if statements</p> <p>While loop for play again (<code>play == 'y'</code>) or similar</p> <p>Function <code>void display_details(void);</code></p> <p>Function <code>int roll_die(void);</code></p> <p>Function <code>void deal_hand(int hand[], int max_dice);</code></p> <p>Function <code>int rank_hand(int hand[], int max_dice);</code></p> <p>Function <code>void display_rank(int rank);</code></p> <p>Array of integers to represent player and dealer hands</p> <p>No global variables.</p> <p>No magic numbers.</p> <p>Use of the supplied <code>void display_hand(int hand[], int max_dice)</code> function.</p> <p>Validate user input – validate [y/n] choice only. Not required to check for numeric input or strings when expecting characters, etc. Please enter either 'y' or 'n'.</p> <p>Good loops (i.e. no <code>break</code>, <code>exit</code> or <code>return</code> statements (or similar) to exit loops).</p>			<p><input type="checkbox"/> -1 No or incorrect use of <code>rand()</code></p> <p><input type="checkbox"/> -1 No or incorrect if statements</p> <p><input type="checkbox"/> -2 No or incorrect play again while</p> <p><input type="checkbox"/> -1 Not to specs or -1 not implemented</p> <p><input type="checkbox"/> -1 Not to specs or -2 not implemented</p> <p><input type="checkbox"/> -1 Not to specs or -2 not implemented</p> <p><input type="checkbox"/> -1 Not to specs or -2 not implemented</p> <p><input type="checkbox"/> -1 Not to specs or -2 not implemented</p> <p><input type="checkbox"/> -1 Not to specs or -2 not implemented</p> <p><input type="checkbox"/> -2 For use of global variables</p> <p><input type="checkbox"/> -1 For use of magic numbers</p> <p><input type="checkbox"/> -2 No or incorrect use of function</p> <p><input type="checkbox"/> -2 No validation of user input</p> <p><input type="checkbox"/> -2 For using <code>break</code>, <code>exit</code> or <code>return</code> statements to exit loops</p>
<p>STYLE</p> <p>Comments (your details, program description, all variable definitions, functions, function prototypes and significant sections of code).</p> <p>Consistent code layout and indentation.</p> <p>Meaningful variable names (no single letter variable names).</p>			<p><input type="checkbox"/> -2 Insufficient comments</p> <ul style="list-style-type: none"> o your details at top of file, o program description, o all variable definitions, o all function definitions, o all function prototypes and o significant sections of code <p><input type="checkbox"/> -2 Inconsistent indentation and layout</p> <p><input type="checkbox"/> -2 Non-descriptive variable names</p>
<p>TOTAL</p>	<p>20</p>		
<p><i>The Graduate qualities being assessed by this assignment are indicated by an X:</i></p>			
<p>X GQ1: operate effectively with and upon a body of knowledge</p>	<p>GQ5: are committed to ethical action and social responsibility</p>		
<p>GQ2: are prepared for lifelong learning</p>	<p>X GQ6: communicate effectively</p>		
<p>GQ3: are effective problem solvers</p>	<p>GQ7: demonstrate an international perspective</p>		
<p>GQ4: can work both autonomously and collaboratively</p>			

This form meets the 2006 requirements of UniSA's Code of Good Practice: Student Assessment

SAMPLE OUTPUT

Please note that your values will be different to the sample provided as the numbers/values generated are random. :)

Sample output 1:

```
File      : wayby001_dice_poker.c
Author    : Batman
Stud ID   : 0123456X
Email ID  : wayby001
This is my own work as defined by the
University's Academic Misconduct Policy.
```

Would you like to play dice poker [y|n]? n

No worries... another time perhaps... :)

Sample output 2:

```
File      : wayby001_dice_poker.c
Author    : Batman
Stud ID   : 0123456X
Email ID  : wayby001
This is my own work as defined by the
University's Academic Misconduct Policy.
```

Would you like to play dice poker [y|n]? z
Please enter either 'y' or 'n'.

Would you like to play dice poker [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[2]	[5]	[6]	[5]	[4]
*	* *	* *	* *	* *
	*	* *	*	
*	* *	* *	* *	* *

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[5]	[3]	[4]	[1]	[3]
* *	*	* *		*
*	*		*	*
* *	*	* *		*

-- Player has One pair
-- Dealer has One pair

** Draw! **

Play again [y|n]? q
Please enter either 'y' or 'n'.

Play again [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[1]	[3]	[4]	[1]	[2]
	*	* *		*
*	*		*	
	*	* *		*

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[2]	[4]	[6]	[5]	[3]
*	* *	* *	* *	*
		* *	*	*
*	* *	* *	* *	*

-- Player has One pair
-- Dealer has Nothing special

** Player wins! **

Play again [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[1]	[4]	[3]	[6]	[5]
*	* *	*	* *	* *
*	*	*	*	*
	* *	*	* *	* *

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[6]	[4]	[4]	[4]	[6]
* *	* *	* *	* *	* *
* *				* *
* *	* *	* *	* *	* *

-- Player has Nothing special

-- Dealer has Full house

** Dealer wins! **

Play again [y|n]? n

Game Summary

=====

You played 3 games:

|--> Games won: 1

|--> Games lost: 1

|--> Games drawn: 1

Thanks for playing!

Sample output 3:

File : wayby001_dice_poker.c

Author : Batman

Stud ID : 0123456X

Email ID : wayby001

This is my own work as defined by the
University's Academic Misconduct Policy.

Would you like to play dice poker [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[4]	[6]	[1]	[2]	[1]
* *	* *		*	
	* *	*		*
* *	* *		*	

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[2]	[3]	[4]	[2]	[4]
*	*	* *	*	* *
	*			
*	*	* *	*	* *

-- Player has One pair

-- Dealer has Two pair

** Dealer wins! **

Play again [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[1]	[5]	[4]	[5]	[5]
	* *	* *	* *	* *
*	*		*	*
	* *	* *	* *	* *

Dealer's hand:


```

        Die 1   Die 2   Die 3   Die 4   Die 5
        [1]     [2]     [5]     [1]     [3]
          *      *      * *      *      *
          *      *      *      *      *
          *      *      * *      *      *

-- Player has Three of a kind
-- Dealer has One pair

** Player wins! **

Play again [y|n]? y

Player's hand:
        Die 1   Die 2   Die 3   Die 4   Die 5
        [4]     [3]     [1]     [1]     [1]
          * *      *      *      *      *
          * *      *      *      *      *

Dealer's hand:
        Die 1   Die 2   Die 3   Die 4   Die 5
        [2]     [6]     [5]     [1]     [5]
          *      * *      * *      *      *
          *      * *      *      *      *
          *      * *      * *      *      *

-- Player has Three of a kind
-- Dealer has One pair

** Player wins! **

Play again [y|n]? y

Player's hand:
        Die 1   Die 2   Die 3   Die 4   Die 5
        [1]     [4]     [4]     [2]     [3]
          *      * *      * *      *      *
          *      * *      * *      *      *

Dealer's hand:
        Die 1   Die 2   Die 3   Die 4   Die 5
        [6]     [6]     [2]     [6]     [6]
          * *      * *      *      * *      * *
          * *      * *      *      * *      * *
          * *      * *      *      * *      * *

-- Player has One pair
-- Dealer has Four of a kind

** Dealer wins! **

Play again [y|n]? n

Game Summary
=====
You played 4 games:
|--> Games won:    2
|--> Games lost:   2
|--> Games drawn:  0

Thanks for playing!

Sample output 4:
File      : wayby001_dice_poker.c
Author    : Batman
Stud ID   : 0123456X
Email ID  : wayby001
This is my own work as defined by the
University's Academic Misconduct Policy.

```

Would you like to play dice poker [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[5]	[5]	[5]	[5]	[5]
* *	* *	* *	* *	* *
*	*	*	*	*
* *	* *	* *	* *	* *

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[5]	[2]	[2]	[5]	[3]
* *	*	*	* *	*
*		*	*	*
* *	*	*	* *	*

-- Player has Five of a kind

-- Dealer has Two pair

** Player wins! **

Play again [y|n]? n

Game Summary

=====

You played 1 games:

|--> Games won: 1

|--> Games lost: 0

|--> Games drawn: 0

Thanks for playing!

Sample output 5:

File : wayby001_dice_poker.c

Author : Batman

Stud ID : 0123456X

Email ID : wayby001

This is my own work as defined by the
University's Academic Misconduct Policy.

Would you like to play dice poker [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[6]	[2]	[1]	[1]	[5]
* *	*			* *
* *		*	*	*
* *	*			* *

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[3]	[4]	[5]	[6]	[4]
*	* *	* *	* *	* *
*		*	* *	
*	* *	* *	* *	* *

-- Player has One pair

-- Dealer has One pair

** Draw! **

Play again [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[1]	[4]	[1]	[1]	[3]
*	* *	*	*	*
	* *			*

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[2]	[2]	[6]	[5]	[3]

```

      *      *      * *      * *      *
      *      *      * *      * *      *

```

```

-- Player has Three of a kind
-- Dealer has One pair

```

```

** Player wins! **

```

```

Play again [y|n]? y

```

```

Player's hand:

```

```

Die 1   Die 2   Die 3   Die 4   Die 5
[6]     [1]     [4]     [5]     [1]
* *     *      * *     * *     *
* *     *      *      *      *
* *     *      * *     * *

```

```

Dealer's hand:

```

```

Die 1   Die 2   Die 3   Die 4   Die 5
[3]     [4]     [1]     [6]     [1]
*      * *     *      * *     *
*      *      *      * *     *
*      * *     *      * *

```

```

-- Player has One pair
-- Dealer has One pair

```

```

** Draw! **

```

```

Play again [y|n]? y

```

```

Player's hand:

```

```

Die 1   Die 2   Die 3   Die 4   Die 5
[4]     [2]     [5]     [6]     [1]
* *     *      * *     * *     *
* *     *      *      * *     *
* *     *      * *     * *

```

```

Dealer's hand:

```

```

Die 1   Die 2   Die 3   Die 4   Die 5
[6]     [3]     [5]     [4]     [3]
* *     *      * *     * *     *
* *     *      *      *      *
* *     *      * *     * *

```

```

-- Player has Nothing special
-- Dealer has One pair

```

```

** Dealer wins! **

```

```

Play again [y|n]? y

```

```

Player's hand:

```

```

Die 1   Die 2   Die 3   Die 4   Die 5
[5]     [6]     [5]     [3]     [5]
* *     * *     * *     *      * *
*      * *     *      *      *
* *     * *     * *     *      * *

```

```

Dealer's hand:

```

```

Die 1   Die 2   Die 3   Die 4   Die 5
[6]     [3]     [5]     [2]     [3]
* *     *      * *     *      *
* *     *      *      *      *
* *     *      * *     *      *

```

```

-- Player has Three of a kind
-- Dealer has One pair

```

```

** Player wins! **

```

Play again [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[5]	[5]	[3]	[1]	[4]
* *	* *	*		* *
*	*	*	*	
* *	* *	*		* *

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[1]	[1]	[3]	[5]	[6]
		*	* *	* *
*	*	*	*	* *
		*	* *	* *

-- Player has One pair
-- Dealer has One pair

** Draw! **

Play again [y|n]? y

Player's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[5]	[1]	[6]	[5]	[3]
* *		* *	* *	*
*	*	* *	*	*
* *		* *	* *	*

Dealer's hand:

Die 1	Die 2	Die 3	Die 4	Die 5
[5]	[2]	[3]	[3]	[3]
* *	*	*	*	*
*		*	*	*
* *	*	*	*	*

-- Player has One pair
-- Dealer has Three of a kind

** Dealer wins! **

Play again [y|n]? n

Game Summary

=====

You played 7 games:
|--> Games won: 2
|--> Games lost: 2
|--> Games drawn: 3

Thanks for playing!