ENGGEN 131 –Semester Two, 2013

Oh no, it's Uno!

Project deadline: 10:00pm, Sunday 20th October
Worth: 10%

Peer review deadline: 10:00pm, Friday 25th October
Worth: 2%

Introduction

Uno is, by far, the most popular and exciting card game in the world.[1] In this project, you will be creating and shuffling an Uno deck, verifying that a card played in a game is valid, and defining **your own** strategy for playing the game that can be used to compete against the strategies of other students.

Uno is played with a deck of 108 special cards. Each player begins with 7 cards, and the aim of the game is to be the first player to discard your entire hand. Players take turns discarding one card from their hand at a time, as long as that card "matches" (based on certain criteria) the top card on the discard pile.

The complete set of 108 Uno cards consists of:

- 4 "wild draw four" cards
- 4 "wild" cards
- 25 red coloured cards
- 25 green coloured cards
- 25 blue coloured cards
- 25 yellow coloured cards

The 25 cards of each colour consist of two "skip", two "reverse" and two "draw two" cards. The remaining 19 cards of a particular colour are called "number" cards, and are labelled with numbers between 0 and 9. There is just one card labelled 0, and two cards with each of the other 9 numbers.

[1] *this statement is completely unsubstantiated*

How to play

There are several variants on the rules for Uno. Please read through the following rules that apply to this project.

Number of players
A game consists of four players.

To begin one game, the Uno deck is shuffled and each player is dealt 7 cards. The remainder of the deck is placed face down, and the top card is turned over and becomes the first card in the discard pile. This revealed top card will not be a "wild" card – if, at the start of a game, the top card on the discard pile is one of the 8 "wild" cards, then the next card will be turned upwards and so on, until a coloured card is at the top of the discard pile.

Matching cards
The first player to have a turn is selected at random. Play then continues in a random direction (either clockwise or anti-clockwise). Players take turns placing one card at a time on top of the discard pile. The played card must "*match"* the card at the top of the discard pile – either the **colour** or the **rank** (0-9, skip, draw two, reverse) of the chosen card must match the card on the top of the discard pile. If a numbered card (0-9) is played, then play simply continues to the next player. If one of the other types of cards is played, the following happens:

- a "skip" – when this card is played, the *next* player forfeits their turn (play skips to the following player)
- a "reverse" – when this card is played, the direction of play changes
- a "draw two" – when this card is played, the *next* player must pick up two cards from the deck and then forfeit their turn (play skips to the following player)

Wild cards
The only exception to the rule that the played card must match the colour or rank of the top card of the discard pile applies to the "wild" cards. A "wild" or a "wild draw four" card may be played *at any time*. The player who plays such a card then chooses what the colour will be from that point on (until changed by a subsequent card). If a "wild" is played, the new colour is chosen by the player and then play continues with the next player. If a "wild draw four" is played, the new colour is chosen by the player, the *next* player then picks up four cards and forfeits their turn, and play continues with the following player.

If no card can be played
If a player cannot play a card on their turn (because they have no matching card in their hand), then they must pick up one card from the deck. That is then the end of their turn – they cannot immediately play the card they picked up.

No voluntary "picking up"
A player must play a card, if they are able to, on their turn. A player cannot choose to pick up a card from the deck if there is a card in their hand that they could have played (this includes "wild" cards).

Penalty
If a player attempts to play a card that is not valid, not only will they be verbally ridiculed by the other three players, but they will be penalised and will have to pick up two cards from the deck (including the card they attempted to play, and then forfeit their turn). The same penalty applies if playing a "wild" card but forgetting to call the new colour.

Empty deck
If at any point the deck becomes empty, the discard pile (with the exception of the top card) will be shuffled and then becomes the new deck.

Scoring

Although the winner of a single game is the player who is the first to discard their entire hand, the winner of a **tournament** is the player that has the highest number of points after playing multiple games.

As soon as one game is over (i.e. a player plays their last card) that player earns a number of points which are calculated based on the remaining cards in all of the other players' hands. Any of the number cards are worth their face value (0-9), any of the other special coloured cards ("skip", "draw two", "reverse") are worth 20 points, and any "wild" or "wild draw four" is worth 50 points.

The winner's score is the sum of all of the remaining cards in every other player's hands.

The **winner of a tournament** is the player with the highest total score over many games.

Your four tasks

There are four distinct tasks for you to complete. Each task requires you to write one function, although for some tasks you may wish to define additional functions. All of your functions should be written in a single source file, and each function must begin with **your** UPI. The examples below use the UPI "abcd001".

- **Task One**: write a function called "abcd001_CreateDeck" that creates a complete deck of Uno cards
- **Task Two**: write a function called "abcd001_ShuffleDeck" that shuffles the deck
- **Task Three**: write a function called "abcd001_ValidSelection" that checks if a player's selection is valid
- **Task Four**: write a function called "abcd001_Play" that implements your strategy for playing Uno

No main() function
Note that the source file (called abcd001_proj2.c) in which you write these functions **must not** include a main() function. Instead, to test these four functions you should use a **separate source** file that does include a main() function. A small example of such a source file (called do_not_submit.c) is provided to you, but it is likely you will want to improve this program so that it is more thorough in its testing of your functions. You will not submit the program that tests your functions. The markers will use their own marking program (with a main() function) to mark your project. A header file (called proj2.h) has also been provided to you.

do_not_submit.c

```
#include "proj2.h"

int main(void)
{
    ....
}
```

proj2.h

```
#define NUMBER    0
#define SKIP      1
#define REVERSE   2
#define DRAW_TWO  3
#define WILD      4
#define WILD_D4   5
#define PENALTY   6
....
```

abcd001_proj2.c

```
#include "proj2.h"

void abcd001_CreateDeck(Deck *deck)
{
    ....
}
void abcd001_ShuffleDeck(Deck *deck, int numberOfCards)
{
    ....
}

int abcd001_ValidSelection(GameState *gameInfo, int selection)
{
    ....
}

int abcd001_Play(GameState *gameInfo)
{
    ....
}
```

Getting started

Download the following files from Cecil:

- abcd001_proj2.c
- proj2.h
- do_not_submit.c

You will need to change **all occurrences** of “abcd001” (in both the abcd001_proj2.c file and the do_not_submit.c file) to your UPI. You will also need to change the name of the file “abcd001_proj2.c” so that it begins with your UPI and not “abcd001”.

You are now ready to go!

Task One: *create a deck of Uno cards*

Define the function:

```
void abcd001_CreateDeck(Deck *deck);
```

This function should initialise the Deck structure that the parameter “deck” points to. Each of the 108 elements in the cards array should represent one valid Uno card. Together, all 108 elements must represent a valid Uno deck. Use the constants defined in proj2.h to help you. Remember that the “value” field for each card should store the corresponding points value for that card (50 for “wilds”, 20 for special cards, and the face value for a number card).

It is now your responsibility to test that your CreateDeck() function works correctly. To do this, you should expand on the basic code written in the do_not_submit.c source file, and include some more thorough tests. For example, you could write a function that counts and displays how many cards of each colour are in the deck. Or, you might like to calculate the sum of all the point values of the cards in the deck, to ensure that this equals the value you expect.

Once you are confident that the deck has been initialised correctly, go on to Task Two.

Task Two: *shuffle a deck of Uno cards*

Define the function:

```
void abcd001_ShuffleDeck(Deck *deck, int numberOfCards);
```

This function should shuffle the order of the cards in the Deck structure that the parameter “deck” points to. The number of cards shuffled by this function is determined by the parameter “numberOfCards”. Only elements of the array from index position 0 to index position numberOfCards-1 should be shuffled when this function is called.

How can you test that this function is working correctly? One approach is to call the CreateDeck and ShuffleDeck functions a very large number of times and see if the probability of certain cards ending up in certain positions matches the theoretical probability. For example, if you shuffle a newly created deck, the probability that a green 0 appears in the first position of the deck would be 1/108.

DO NOT CALL the srand() function. This function must only be called once (to set the random seed for the pseudo-random number generator) and this will be called for you in the main() function (e.g. in do_not_submit.c)

Task Three: *determine that a selection is valid*

Define the function:

```
int abcd001_ValidSelection(GameState *gameInfo, int selection);
```

When the abcd001_Play() function is called (see Task Four), it returns an integer value representing the player's selection. The abcd001_ValidSelection() function checks whether or not the player's selection is valid for the given GameState structure. Before attempting this task, read the description for Task Four carefully, as that describes the values for "selection" that are valid.

Once you have read through Task Four, you will realise that there are three different cases to consider:

The selection is -1
This selection indicates that the player cannot play a card on their turn and will therefore pick up a card. This selection is only valid if there is no card in the player's hand that they can play – you should confirm this.

The selection is between 0 and yourNumberOfCards-1
This selection indicates that the player has chosen to play the card at the specified index of the cards array. You should confirm that it is indeed valid for the card at this position of the array to be played.

The selection is greater than yourNumberOfCards
If the user chooses to play a "wild" card, then they must indicate the new colour as part of their selection. This is done by adding the colour chosen to the index position of the "wild" card. So, for example, consider the case where the player has a "wild" card at index position 3 of the cards array. If they want to play this card and set the new colour to yellow, the value they must return will be YELLOW + 3 (based on the constants in proj2.h, this selection will have a numeric value of 40003). You should confirm that if the player has selected a "wild", they have correctly chosen a new colour.

Your abcd001_ValidSelection() function should return true (i.e. 1) if the selection is valid, and false (i.e. 0) otherwise.

Task Four: *define a strategy for playing Uno*

Define the function:

```
int abcd001_Play(GameState *gameInfo);
```

When this function is called, it is passed the complete state of a game of Uno currently in progress. The state information is represented by the GameState structure which has the following fields:

```
typedef struct {
      int numberOfPlayers;
      int directionOfPlay;
      int lastColourCalled;
      int yourPosition;
      int yourNumberOfCards;
      Card yourCards[MAX_CARDS];
      int numberOfDiscards;
      Card discardPile[MAX_CARDS];
      int allPlayersCardsLeft[MAX_PLAYERS];
      int allPlayersColoursCalled[MAX_PLAYERS];
      int allPlayersScores[MAX_PLAYERS];
} GameState;
```

Each of these fields is briefly described next.

- `int numberOfPlayers;`
  The number of people playing the game. In our case, this will always be four.

- `int directionOfPlay;`
  Indicates the current direction of play. Either LEFT (which means the index of the *next* player is one less than the *current* player) or RIGHT (which means the index of the *next* player is one more than the *current* player).

- `int lastColourCalled;`
  This field stores the colour that was called the last time a “wild” or “wild draw four” was played. It will either be RED, GREEN, BLUE, YELLOW or NONE (if no wild has been played). So, if the top card on the discard pile is a “wild”, then this field will tell you the colour that you must play.

- `int yourPosition;`
  The index position (out of the four player positions, 0-3) that corresponds to ***your*** player

- `int yourNumberOfCards;`
  The number of cards that you currently have in ***your*** hand

- `Card yourCards[MAX_CARDS];`
  The actual cards that are in ***your*** hand.

- `int numberOfDiscards;`
  The number of cards currently in the discard pile.

- `Card discardPile[MAX_CARDS];`

  All of the cards that are currently on the discard pile. The bottom of the discard pile is at index 0 (this is the card that was first turned face up when the game started). The ***top card*** on the discard pile (which is the card that you must *match* when you select your card to play) is therefore at index **`numberOfDiscards - 1`**

- `int allPlayersCardsLeft[MAX_PLAYERS];`

  An array that contains the number of cards that each player has remaining in their hands

- `int allPlayersColoursCalled[MAX_PLAYERS];`

  An array that contains the last colour called by each player (if they have played a "wild" card in this game). Each entry will either be RED, GREEN, BLUE, YELLOW or NONE (if the corresponding player has not played a wild in this game).

- `int allPlayersScores[MAX_PLAYERS];`

  An array that contains the current score of each player over all games so far played in the tournament (this can be arbitrarily many games).

The return value
As you can see, the `abcd001_Play()` function that you must implement returns an integer. The value that you return represents the **index of the card in your hand** that you wish to play. For example, if your function returns the value 6, then the card that you are going to attempt to play will be the card stored at index position 6 in the `yourCards` array of the `GameState` structure (i.e. **`yourCards[6]`**). In this case, you must make sure that the card stored at **`yourCards[6]`** *matches* the card stored at **`discardPile[numberOfDiscards-1]`**.

No matching card in your hand
If you do have any valid card in your hand, you must play it. However, if you are not able to play a card in your hand because you do not have a card that matches the top card on the discard pile (and you have no "wild" cards), then you must return -1. In this case, you will pick up a card from the deck on your turn.

Playing a wild
If the card you are going to play is a "wild" or a "wild draw four" then you must indicate the new colour that you want to change to. The way to indicate this is to **ADD** the new colour to the index of the "wild" card that you want to play. For example, let's say that your "wild" card is stored at index 3 in your hand. Also, let's say that you want to change the colour to yellow. In this case, the value that you return must be YELLOW + 3. If you play a "wild" card, but do not indicate the colour in this way, this will be a penalty and you will have to pick up two cards from the deck.

Penalty
As mentioned above, you will be penalised if you do not specify a colour when playing a "wild". Also, if you attempt to play a card that does not match the top card of the discard pile, or if you do have a card that could be played in your hand but you return -1, then you will be penalized. In this case, you will have to pick up two cards from the deck.

Documentation
You **must** include a comment at the **top of your source** file that describes, **in plain English**, the strategy that you are using. You will be assessed on the clarity of this comment, and on how accurately it actually describes the strategy that is implemented by your code. As a guideline, you should aim for this description to consist of a **minimum** of 150 words – however for complex strategies it would be appropriate for this description to be longer.

Task Five: *for experts only! (voluntary)*

This task is voluntary – and challenging!

Write a program called "**abcd001_simulation.c**" that simulates an Uno tournament consisting of a very large number of games. Your program will call the "....._Play()" function for up to four different strategies when choosing which card to play. To check that your tournament is fair, if you have all four players using the same strategy, then they should accumulate approximately the same number of points over a very large number of games. Try switching between strategies to see which strategy is best!

Restrictions / rules

Please pay careful attention to these rules.

Global variables
You must not declare ANY global variables in your source file. All of the information about the current state of the game (that you may use in your strategy) is provided to you (in the `GameState` structure that is passed to your `abcd001_Play()` function). Your strategy must be based solely on that information.

Just make sure that any variables you declare are *local* – that is, they must be declared *INSIDE* the body of a function.

Additional functions
In addition to the required four functions in the abcd001_proj2.c source file that you submit, you may define as many other functions as you like. However, you **must** prefix the name of EVERY function you define with your UPI, for example: `abcd001_MyOtherFunction()`.

Function prototype declarations
The program that is used to test the functions in your abcd001_proj2.c source file will already have prototype declarations for these functions declared (just like in the example do_not_submit.c source file). If you have defined additional functions, you will need to include their prototype declarations at the top of the abcd001_proj2.c source file.

Preparing your code for submission

You are welcome to develop your solution to this project on your own system, however, prior to submission you **must ensure** that you test your program using Microsoft Visual Studio's Command Prompt tool. This is your responsibility – **if your source code does not compile in this environment, it will not be marked**.

It is **strongly recommended** that you follow the steps below **BEFORE YOU SUBMIT:**

STEP 1:	Create an empty folder on disk
STEP 2:	Copy **just** the source files for this project (`abcd001_proj2.c, proj2.h` and `do_not_submit.c`) into this empty folder.
STEP 3:	Open a command window (as described in Lab 7) and change the current directory to the folder that contains these two files
STEP 4:	Compile the source file using the command line tool, with the warning level on 4: `cl /W4 *.c` If there are warnings, **you should fix them**. You **should not submit** code that generates **any** warnings.

Do not submit code that does not compile, or that compiles but generates warnings

<u>What to submit:</u>

Submit ONLY the source file **abcd001_proj2.c** that contains your definitions of the four required functions.

Academic honesty (this is important!)

This project is an **assessed piece of coursework**, and it is essential that the work you submit reflects what you are capable of doing. You **must not copy any source code** for this project and submit it as your own work. You must also **not allow anyone to copy your work**. All submissions for this project will be checked, and any cases of copying/plagiarism will be dealt with severely. We really hope there are no issues this semester in ENGGEN131, so please be sensible!

Ask yourself:

> have I written the source code for this project myself?

If the answer is "no", then **please talk to us before the projects are marked**.

Ask yourself:

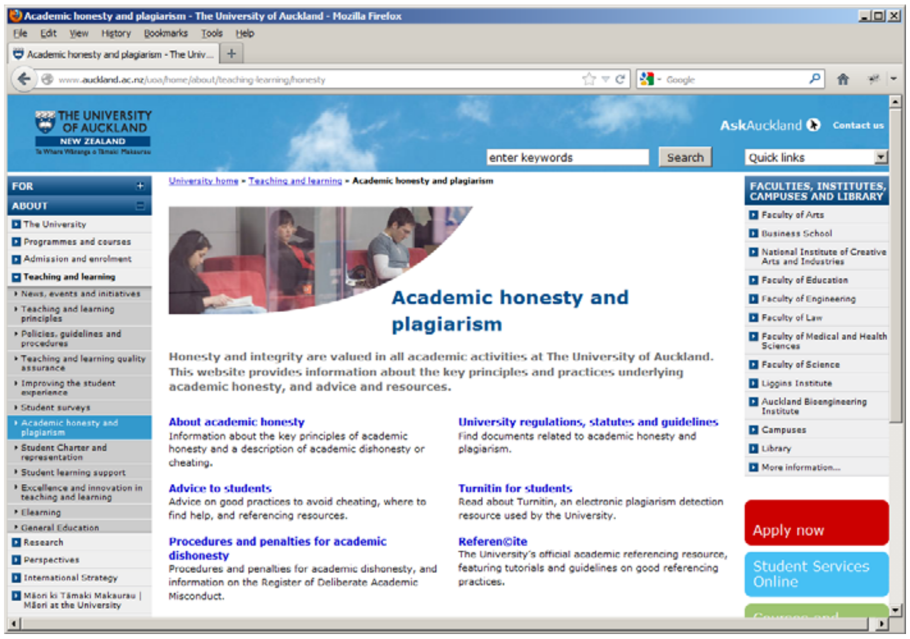
> have I given *anyone* access to the source code that I have written for this project?

If the answer is "yes", then **please talk to us before the projects are marked**.

Once the projects have been marked it is too late.

There is more information regarding The University of Auckland's policies on academic honesty and plagiarism here:

```
http://www.auckland.ac.nz/uoa/home/about/teaching-learning/honesty
```


Academic honesty and plagiarism - The University of Auckland - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Academic honesty and plagiarism - The Univ...
www.auckland.ac.nz/uoa/home/about/teaching-learning/honesty
Google
THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau
AskAuckland
Contact us
enter keywords
Search
Quick links
FOR
ABOUT
The University
Programmes and courses
Admission and enrolment
Teaching and learning
News, events and initiatives
Teaching and learning principles
Policies, guidelines and procedures
Teaching and learning quality assurance
Improving the student experience
Student surveys
Academic honesty and plagiarism
Student Charter and representation
Student learning support
Excellence and innovation in teaching and learning
Elearning
General Education
Research
Perspectives
International Strategy
Māori ki Tāmaki Makaurau | Māori at the University
University home » Teaching and learning » Academic honesty and plagiarism
FACULTIES, INSTITUTES, CAMPUSES AND LIBRARY
Faculty of Arts
Business School
National Institute of Creative Arts and Industries
Faculty of Education
Faculty of Engineering
Faculty of Law
Faculty of Medical and Health Sciences
Faculty of Science
Liggins Institute
Auckland Bioengineering Institute
Campuses
Library
More information...
Apply now
Student Services Online

Academic honesty and plagiarism

Honesty and integrity are valued in all academic activities at The University of Auckland. This website provides information about the key principles and practices underlying academic honesty, and advice and resources.

About academic honesty
Information about the key principles of academic honesty and a description of academic dishonesty or cheating.

University regulations, statutes and guidelines
Find documents related to academic honesty and plagiarism.

Advice to students
Advice on good practices to avoid cheating, where to find help, and referencing resources.

Turnitin for students
Read about Turnitin, an electronic plagiarism detection resource used by the University.

Procedures and penalties for academic dishonesty
Procedures and penalties for academic dishonesty, and information on the Register of Deliberate Academic Misconduct.

Referen©ite
The University's official academic referencing resource, featuring tutorials and guidelines on good referencing practices.