C++

程式語言（二）

Introduction to Programming (II)

Case Studies & Supplementary

Joseph Chuang-Chieh Lin

Dept. CSE, NTOU

Platform/IDE

- Dev-C++

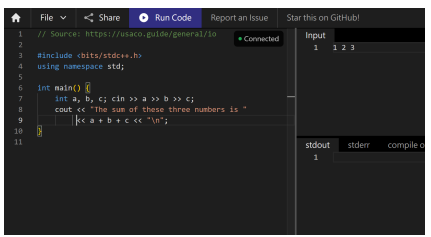
https://www.pngegg.com/en/search?q=Dev-C

- Codeblocks

https://icons8.com/icons/set/code-blocks

- OnlineGDB (https://www.onlinegdb.com/)

- Real-Time Collaborative Online IDE (https://ide.usaco.guide/)



Textbooks (We focusing on C++11)

- ***Learn C++ Programming by Refactoring* (由重構學習 C++ 程式設計). Pang-Feng Liu (劉邦鋒). NTU Press. 2023.**
- ***C++ Primer. 5th Edition.* Stanley B. Lippman, Josée Lajoie, Barbara E. Moo. 2019.**
- *Effective C++*. Scott Meyers. O'Reilly. 2016.
- *Thinking in C++. Vol. 1: Introducing to Standard C++*. 2nd Edition. Bruce Eckel. Prentice Hall PTR. 2000.

Useful Resources

- Tutorialspoint
  - https://www.tutorialspoint.com/cplusplus/index.htm
  - Online C++ Compiler
- Programiz
  - https://www.programiz.com/cpp-programming
- LEARN C++
  - https://www.learncpp.com/
- MIT OpenCourseWare - Introduction to C++
  - https://ocw.mit.edu/courses/6-096-introduction-to-c-january-iap-2011/pages/lecture-notes/
- Learning C++ Programming
  - https://www.programiz.com/cpp-programming
- GeeksforGeeks
  - https://www.geeksforgeeks.org/c-plus-plus/

enum class

- Use `enum class` instead of `enum`.
  - A new feature in C++11.

```
enum ourColor {
    RED,
    GREEN,
    BLUE
};

ourColor color = RED;
```

```
enum class ourColor {
    RED,
    GREEN,
    BLUE
};

ourColor color = ourColor::RED;
```

Previous Issue (I)

https://kheresy.wordpress.com/2019/03/27/using-enum-class/

```
#include <iostream>

enum ourColor {
  RED,
  GREEN,
  BLUE
};

enum ourFruit {
  APPLE,
  BANANA
};
```

```
int main() {
    ourColor c1 = RED;
    ourFruit f1 = APPLE;

    if (c1 == f1) {
        cout << "c1 equals f1" << endl;
    } else {
        cout << "c1 and f1 are not equal"

             << endl;
    }
    return 0;
}
```

Previous Issue (I): Compile error

https://kheresy.wordpress.com/2019/03/27/using-enum-class/

```
#include <iostream>

enum class ourColor {
  RED,
  GREEN,
  BLUE
};

enum class ourFruit {
  APPLE,
  BANANA
};
```

```
int main() {
    ourColor c1 = ourColor::RED;
    ourFruit f1 = ourFruit::APPLE;

    if (c1 == f1) {
        cout << "c1 equals f1" << endl;
    } else {
        cout << "c1 and f1 are not equal"

             << endl;
    }
    return 0;
}
```

Previous Issue (II)

https://kheresy.wordpress.com/2019/03/27/using-enum-class/

```
#include <iostream>

enum ourColor {
  RED,
  GREEN,
  BLUE
};

enum tLight {
  RED,
  YELLOW,
  Green
};
```

```
int main() {
    ourColor c1 = RED; // redefine; error!
    return 0;
}
```

Previous Issue (II)

https://kheresy.wordpress.com/2019/03/27/using-enum-class/

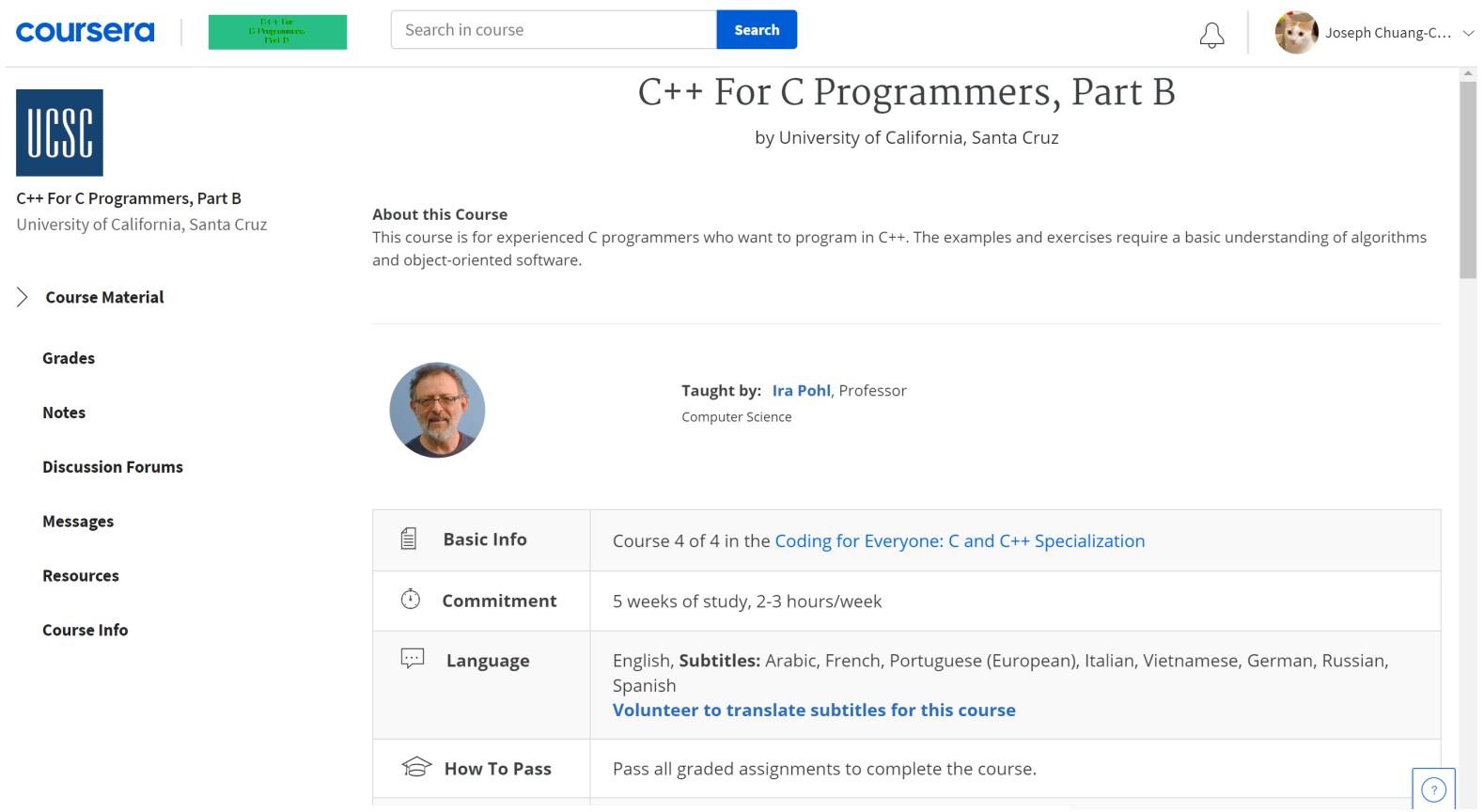
```
#include <iostream>

enum class ourColor {
  RED,
  GREEN,
  BLUE
};

enum class tLight {
  RED,
  YELLOW,
  Green
};
```

```
int main() {
    ourColor c1 = ourColor::RED; // safe!
    return 0;
}
```

A Sample Project:

Poker Probabilities

Material refer to C++ For C Programmers (Coursera)

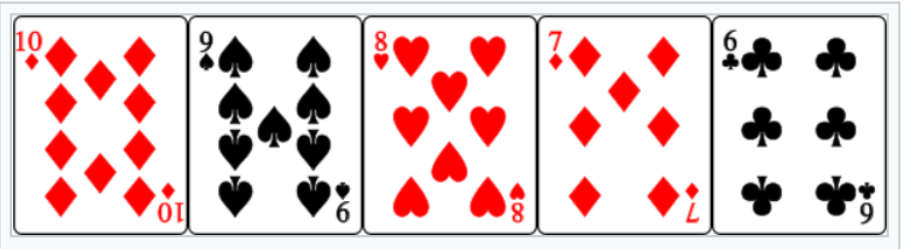


https://www.coursera.org/learn/c-plus-plus-b/home/info

Project: Card Probability

- flush

- straight



https://en.wikipedia.org/wiki/List_of_poker_hands

What is the probability of a random shuffle having a flush, straight or a straight-flush?

The Refined Code on OnlineGDB

- https://onlinegdb.com/D5mm_YxfA

Header files:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#include <vector>
#include <algorithm>
```

Suits & Pips

```
// suits
enum class suit: short {
    SPADE, HEART, DIAMOND, CLUB
};
```

```
class pips {
public:
    pips(int val): v(val) { assert(v>0 && v<14); }
    friend ostream& operator<<(ostream& out, const pips& p);
    int get_pips() { return v; }
private:
    int v;
};
```

Card

```
class card {
public:
    card(): s(suit::SPADE), v(1) {}
    card(suit st, pips pv): s(st), v(pv) {}
    friend ostream& operator<<(ostream& out, const card& c);
    suit get_suit() { return s; }
    pips get_pips() { return v; }
private:
    suit s;
    pips v;
};
```

Ostream << overloading

```
ostream& operator<<(ostream& os, const suit& s) {
   os << static_cast<std::underlying_type<suit>::type>(s);
   return os;
}

ostream& operator<<(ostream& os, const pips& p) {
    os << p.v;
    return os;
}

ostream& operator<<(ostream& os, const card& c) {
    os << "pips: " << c.v << "suit: " << c.s << endl;
    return os;
}
```

Initialization of the deck & Print

```
void init_deck(vector<card> & d) {
    int i;
    for (i=1; i<14; i++) {
        card c(suit::SPADE, i);
        d[i-1] = c;
    }
    for (i=1; i<14; i++) {
        card c(suit::HEART, i);
        d[i+12] = c;
    }
    for (i=1; i<14; i++) {
        card c(suit::DIAMOND, i);
        d[i+25] = c;
    }
    for (i=1; i<14; i++) {
        card c(suit::CLUB, i);
        d[i+38] = c;
    }
}
```

```
void print(vector<card> &deck) {
    for (auto p=deck.begin(); p!=deck.end(); ++p) {
    // for (auto card_val: deck) cout << card_val
        cout << *p;
    }
    cout << endl;
}
```

Check if the deck is a flush

```
bool is_flush(vector<card> &hand) {
    suit s = hand[0].get_suit();
    for (auto p=hand.begin(); p!=hand.end(); ++p) {
        if (s != p->get_suit()) {
            return false;
        }
    }
    return true;
}
```

Check if the deck is a straight

```
bool is_straight(vector<card> &hand) {
    int pips_v[5];
    int i = 0;
    for (auto p=hand.begin(); p!=hand.end(); ++p) {
        pips_v[i++] = (p->get_pips()).get_pips();
    }
    sort(pips_v, pips_v+5); // feed the range for the iterator
    if (pips_v[0] != 1) { // not ACE
        return (pips_v[0] == pips_v[1]-1 && pips_v[1] == pips_v[2]-1)
        && (pips_v[2] == pips_v[3]-1 && pips_v[3] == pips_v[4]-1);
    } else {
        return (pips_v[0] == pips_v[1]-1 && pips_v[1] == pips_v[2]-1)
        && (pips_v[2] == pips_v[3]-1 && pips_v[3] == pips_v[4]-1)
        || (pips_v[1] == 10) && (pips_v[2] == 11) && (pips_v[3] == 12)
        && (pips_v[4] == 13);
    }
}
```

Straight and Flush

```
bool is_straight_flush(vector<card> &hand) {
    return is_flush(hand) && is_straight(hand);
}
```

Main Function

```
vector<card> deck(52);
srand(time(0));
init_deck(deck);
int num_shuffles;
int flush_count = 0;
int str_count = 0;
int str_flush_count = 0;
cout << "How many shuffles? ";
cin >> num_shuffles;
```

```
for (int loop=0; loop<num_shuffles; ++loop) {
        random_shuffle(deck.begin(), deck.end());
        vector<card> hand(5);
        int i=0;
        for (auto p=deck.begin(); i<5; ++p) {
            hand[i++] = *p;
        }
        if (is_flush(hand)) {
            flush_count++;
        }
        if (is_straight(hand)) {
            str_count++;
        }
        if (is_straight_flush(hand)) {
            str_flush_count++;
        }
}
```

```
    cout << "Flushes: " << flush_count << " out of " << num_shuffles << endl;
    cout << "Straights: " << str_count << " out of " << num_shuffles << endl;
    cout << "Straight Flushes: " << str_flush_count
         << " out of " << num_shuffles << endl;
```

Some Notices about the Iterators

Container	Iterator type	Container	Iterator type
Sequence containers (first class)		*Unordered associative containers (first class)*	
`vector`	random access	`unordered_set`	bidirectional
`array`	random access	`unordered_multiset`	bidirectional
`deque`	random access	`unordered_map`	bidirectional
`list`	bidirectional	`unordered_multimap`	bidirectional
`forward_list`	forward		
Ordered associative containers (first class)		*Container adapters*	
`set`	bidirectional	`stack`	none
`multiset`	bidirectional	`queue`	none
`map`	bidirectional	`priority_queue`	none
`multimap`	bidirectional		

C++ algorithms library

- https://en.cppreference.com/w/cpp/algorithm

Notices about `sort()`

- Arrange the elements in the range from `XXX.begin()` up to but not including `XXX.end()` in ascending order.
- The `sort()` algorithm requires its two iterator arguments to be random-access iterators.
  - Available data types or containers: built-in arrays and STL containers **array**, **vector** and **deque**.

Revisit to `is_straight()`

```
bool is_straight(vector<card> &hand) {
    int pips_v[5];
    int i = 0;
    for (auto p=hand.begin(); p!=hand.end(); ++p) {
        pips_v[i++] = (p->get_pips()).get_pips();
    }
    sort(pips_v, pips_v+5); // feed the range for the iterator
    if (pips_v[0] != 1) { // not ACE
        return (pips_v[0] == pips_v[1]-1 && pips_v[1] == pips_v[2]-1)
        && (pips_v[2] == pips_v[3]-1 && pips_v[3] == pips_v[4]-1);
    } else {
        return (pips_v[0] == pips_v[1]-1 && pips_v[1] == pips_v[2]-1)
        && (pips_v[2] == pips_v[3]-1 && pips_v[3] == pips_v[4]-1)
        || (pips_v[1] == 10) && (pips_v[2] == 11) && (pips_v[3] == 12)
        && (pips_v[4] == 13);
    }
}
```

Using a lambda expression

```
bool is_straight(vector<card> &hand) {
    sort(pips_v, pips_v+5, [](const card& a, const card& b)
       { return (a.get_pips()).get_pips() < (b.get_pips()).get_pips();} );
    int pips_v[5];
    int i = 0;
    for (auto p=hand.begin(); p!=hand.end(); ++p) {
        pips_v[i++] = (p->get_pips()).get_pips();
    }
    if (pips_v[0] != 1) { // not ACE
        return (pips_v[0] == pips_v[1]-1 && pips_v[1] == pips_v[2]-1)
        && (pips_v[2] == pips_v[3]-1 && pips_v[3] == pips_v[4]-1);
    } else {
        return (pips_v[0] == pips_v[1]-1 && pips_v[1] == pips_v[2]-1)
        && (pips_v[2] == pips_v[3]-1 && pips_v[3] == pips_v[4]-1)
        || (pips_v[1] == 10) && (pips_v[2] == 11) && (pips_v[3] == 12)
        && (pips_v[4] == 13);
    }
}
```

Recall: Lambda Expression in C++

- In C++11 and later, a **lambda expression** is a convenient way of defining an anonymous function object *right at the location* where it's invoked or passed as an argument to a function.
  - Especially when it’s not going to be reuse and not worth naming.

```
[ capture clause ] (parameters) -> return-type
{
   definition of method
}
```

https://www.geeksforgeeks.org/lambda-expression-in-c/
https://blog.gtwang.org/programming/lambda-expression-in-c11/

Modified Poker Probability Code

- https://onlinegdb.com/vdGeAd2QIg

Card (Modified...)

```
class card {
public:
    card(): s(suit::SPADE), v(1) {}
    card(suit st, pips pv): s(st), v(pv) {}
    friend ostream& operator<<(ostream& out, const card& c);
    suit get_suit() const { return s; }
    pips get_pips() const { return v; }
private:
    suit s;
    pips v;
};
```

Where did I find the solution? => https://tinyurl.com/cdhm48af

Exercise

- Add a function:

```
bool is_straight_flush(vector<card> &hand)
```

  to the program https://www.onlinegdb.com/vdGeAd2QIg

  to compute the number of fullhouses.

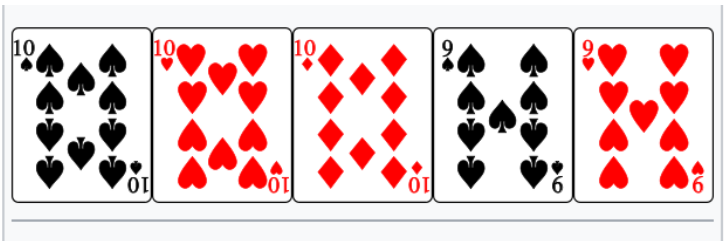
- Sample output:

```
Flushes: 3 out of 1000
Straights: 6 out of 1000
Straight Flushes: 0 out of 1000
Fullhouses: 4 out of 1000
```

Full House

- From Wikipedia:

葫蘆 (夫佬，富而好施，三帶一對)	Full house	三張同一點數的牌，加一對其他點數的牌。	8♠	8♣	8♦	K♠	K♥



Discussions & Questions