enum class

Ben Langmead
ben.langmead@gmail.com
www.langmead-lab.org

Source markdown available at github.com/BenLangmead/c-cpp-notes
enum class

We have often used integers to describe *categorical* data

```c
#include <stdlib.h>

int main() {
    void *memory = malloc(1000000);
    if (memory == NULL) {
        return 1; // failure
    }
    // do something with memory
    return 0; // success
}
```

Returning 0 means “success”, 1 means “failure”
enum class

struct Card {
    int rank; // 1=ace, 2=two, ..., 10=ten
    // 11=jack, 12=queen, 13=king
    int suit; // 0=heart, 1=club, 2=diamond, 3=spade

    Card(int r, int s) : rank(r), suit(s) { }
};

int has advantages; e.g. we can compare ranks with <

Also has disadvantages:

- Mapping between ints and suits is arbitrary
- If we mix up rank and suit – e.g. Card c(3, 13) – compiler can’t catch it
enum class

does not create a *categorical* type

(C & C++ have an older mechanism called simply enum that we won’t discuss here)
#include <iostream>

using std::cout; using std::endl;

enum class Suit {
    HEART, CLUB, DIAMOND, SPADE
};

struct Card {
    int rank; // 1=ace, 11=jack, 12=queen, 13=king
    Suit suit;

    Card(int r, Suit s) : rank(r), suit(s) { }
};

int main() {
    Card c(1, Suit::CLUB); // ace of clubs
    cout << "c.suit = " << (int)c.suit << endl;
    return 0;
}
$ g++ -c enum_1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o enum_1 enum_1.o
$ ./enum_1

c.suit = 1

Behind the scenes, an enum class is really an int

- Starts at 0, so HEART=0, CLUB=1, DIAMOND=2, SPADE=3

C++ will refuse to implicitly convert between enum class and int; we had to explicitly cast for cout

    cout << "c.suit = " << (int)c.suit << endl;
enum class Suit { HEART, CLUB, DIAMOND, SPADE };  

enum class Rank {
    // "= 1" to start numbering at 1 instead of 0
    ACE = 1, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING
};

struct Card {
    Rank rank;
    Suit suit;

    Card(Rank r, Suit s) : rank(r), suit(s) { }
};
enum class

// *** cards.cpp ***

#include <iostream>
#include "cards.h"

using std::cout; using std::endl;

int main() {
    Card c1(Rank::SEVEN, Suit::HEART);
    Card c2((Rank)10, Suit::DIAMOND);
    cout << "c1= " << (int)c1.rank << " , s= " << (int)c1.suit << endl;
    cout << "c2= " << (int)c2.rank << " , s= " << (int)c2.suit << endl;
    return 0;
}

$ g++ -c cards.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o cards cards.o
$ ./cards

c1=7, s=0
c2=10, s=2
enum class

We get a compiler error if we mix up rank & suit

```cpp
#include <iostream>
#include "cards.h"

using std::cout; using std::endl;

int main() {
    Card c1(Suit::HEART, Rank::SEVEN); // oops!
    cout << "c1=\" << (int)c1.rank << ", s=\" << (int)c1.suit << endl;
    return 0;
}
```
$ g++ -c cards_error.cpp -std=c++11 -pedantic -Wall -Wextra

cards_error.cpp: In function 'int main()':
cards_error.cpp:7:37: error: no matching function for call to 'Card::Card(Suit, Rank)'
   Card c1(Suit::HEART, Rank::SEVEN); // oops!
          ^

In file included from cards_error.cpp:2:0:
cards.h:15:5: note: candidate: Card::Card(Rank, Suit)
   Card(Rank r, Suit s) : rank(r), suit(s) { }
       ^~~~

cards.h:15:5: note: no known conversion for argument 1 from 'Suit' to 'Rank'
cards.h:11:8: note: candidate: constexpr Card::Card(const Card&)
   struct Card {
       ^~~~

cards.h:11:8: note: candidate expects 1 argument, 2 provided
cards.h:11:8: note: candidate: constexpr Card::Card(Card&&)
cards.h:11:8: note: candidate expects 1 argument, 2 provided