ECE2800J

Programming and Introductory Data Structures

enum

Learning Objectives:

Know when to use enum type

Know how to use enum type

Categorizing Data

Introducing enums

- In addition to single constants, we may need to categorize data.

- For example, there are four different suits in cards:
  - Clubs ♣
  - Diamonds ♦
  - Hearts ♥
  - Spades ♠

- You could encode each of these as a separate integer like:

```
const int CLUBS = 0;
const int DIAMONDS = 1;
// and so on…
```

Categorizing Data

Introducing enums

```
const int CLUBS = 0;
const int DIAMONDS = 1;
```

- Unfortunately, encoding information this way is not very convenient.

- For example, consider the predicate `isRed()`

```
bool isRed(int suit);
 // REQUIRES: suit is one of Clubs,
 //           Diamonds, Hearts,
 //           or Spades
 // EFFECTS:  returns true if the color
 //           of this suit is red.
```

Categorizing Data

Introducing enums

```
const int CLUBS = 0;
const int DIAMONDS = 1;

bool isRed(int suit);
// REQUIRES: suit is one of Clubs,
//           Diamonds, Hearts, or Spades
// EFFECTS:  returns true if the color
//           of this suit is red.
```

- This is annoying, since we need this REQUIRES clause; not all integers encode a suit.
- There is a better way: the **enumeration** (or **enum**) type.

Categorizing Data

enums

- You can define an enumeration type as follows:

```
enum Suit_t {CLUBS, DIAMONDS,
                 HEARTS, SPADES};
```

- To define variables of this type you say:

```
Suit_t suit;
```

- You can initialize them as:

```
Suit_t suit = DIAMONDS;
```

- Once you have such an enum type defined, you can use it as an argument, just like anything else.
- Enums are passed by-value, and can be assigned.

Categorizing Data

enums

- With enum, the specification for the function `isRed()` can be simplified by removing the REQUIRES clause.

```
bool isRed(Suit_t s);
// EFFECTS:  returns true if the color
//           of this suit is red.
```

Categorizing Data

enums

```
bool isRed(Suit_t s) {
  switch (s) {
      case DIAMONDS:
      case HEARTS:
            return true;
            break;
      case CLUBS:
      case SPADES:
            return false;
            break;
      default:
            assert(0);
            break;
  }
}
```

Categorizing Data

enums

- If you write

```
enum Suit_t {CLUBS, DIAMONDS,
                 HEARTS, SPADES};
```

  then numerically

```
CLUBS = 0, DIAMONDS = 1,
HEARTS = 2, SPADES = 3
```

- Using this fact, it will sometimes make life easier

```
Suit_t s = CLUBS;
const string suitname[] = {“clubs”,
    “diamonds”, “hearts”, “spades”};
cout << “suit s is ” << suitname[s];
```

References

- enum
    - C++ Primer, 4th Edition, Chapter 2.7