Enumerated Types

- Enumerated types allow you to define any ordered group of identifiers as a new type, e.g.

```
enum Weekday {SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
              THURSDAY, FRIDAY, SATURDAY};
```

- Variables of an enumerated type may take on any value in the ordered group.

```
Weekday day = MONDAY;
```

- The enumerated values are integer constants in disguise; the first tee in the list is 0, the next 1, and so on. If `day` is set to `MONDAY` as in the above example then `cout << day << endl;` will give `1`.
- Constants that make up types cannot be shared. A subsequent `enum Weekend {SATURDAY, SUNDAY};` is illegal since these values already appear in the `Weekday` type.
- For another example consider.

```
enum Season {FALL, WINTER, SPRING, SUMMER};
void writeSeason(Season s)
{
      switch (s)
      {
            case FALL:
                  cout << "Fall" << endl;
                  break;
            case WINTER:
                  cout << "Winter" << endl;
                  break;
            case SPRING:
                  cout << "Spring" << endl;
                  break;
            case SUMMER:
                  cout << "Summer" << endl;
                  break;
      }
}
```

enum in C [C]

- When using `enum` in C the `enum` declaration *must* be present, whereas C++ deduces that an `enum` is being used, e.g.

```
enum Season {FALL, WINTER, SPRING, SUMMER};

void writeSeason(enum Season seas)
{
      …

enum Weekday day = MONDAY;
```

- Note the additional struct declarations shown in **bold**.
- C++ deduces it's a struct and doesn't need the struct keyword.