22. Merge Conflicts, const Functions

CPSC 120: Introduction to Programming
Kevin A. Wortman ~ CSU Fullerton

Agenda

1. Preparing for Review Sessions
2. Merge Conflicts
3. const Functions

Reminders:

- 120A exam in finals week
- 120L Portfolio due Wed
- Student Opinion Questionnaires (SOQs) due Fri

2. Socially Responsible Computing Survey

Introduction Script

Survey Link

https://www.surveymonkey.com/r/SRC_student_consent_and_survey

3. const Functions

Review: Principle of Least Privilege

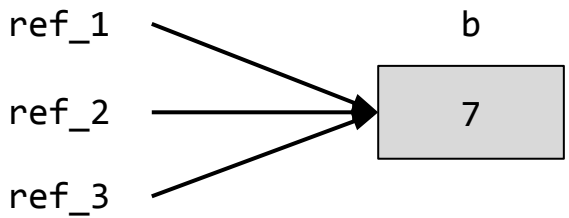
- **Principle of least privilege**: only grant access that is truly necessary
  - “Need to know basis”
  - Evident in iOS and Android apps
  - Ex. only let an app access your location if there is a legitimate need
- Prevents
  - Bugs causing undue harm
  - Spyware

const

- const (in general):
  - “constant”
  - objects do not change
- const reference:
  - variable is a reference to an object
  - reference cannot be used to change the object
- const member function:
  - function cannot change any member variable

Review: const Reference

- `const` keyword: variable **cannot be modified**
- Grants **read-only access** to an object
- Appropriate for loops/functions that have no business modifying the object
- Principle of Least Privilege
- Prevents modifying through the reference by accident



Review: Syntax: lvalue reference to const

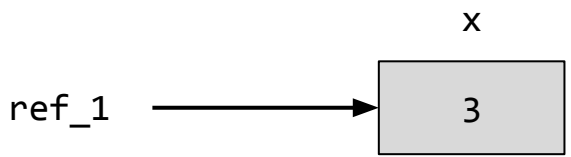
type:

const *referent-type* **&**

Semantics:

- Reference to an object that **may not be modified** (compile error)
- const reference can refer to non-const lvalue
  - (stricter)
- non-const reference **cannot** refer to const reference
  - (would break const protection)

```
int x{ 3 };
const int& ref_1{ x };
int& ref_2{ ref_1 }; // compile error
++x; // OK
++ref_1; // compile error
```



Review: const in for loops

- Recall best practice: for-each variable is a reference
- Best practice: **declare for-each variable reference const**
- Unsafe:

```
std::vector<double> scores{ 91.0, 102.5, 86.0, 110.0, 58.5, 102.0 };
for (double& score : scores) {
  std::cout << score << "\n";
}
```

- Safe:

```
for (const double& score : scores) {
  std::cout << score << "\n";
}
```

const Member Function

- Recall **member function:** function inside a class
- By default, member functions are not const
- Mark a function const by writing **const** after parenthesis () in prototype
  - As usual, declaration and definition must match
- const member function **cannot modify any data members**
- Attempt to do so is **compile error**
  - Compiler helps find bugs at compile time

Example: ScoreBoardEntry

```
// .h file

class ScoreboardEntry {
public:
 ScoreboardEntry();

 ScoreboardEntry(const std::string& name,
                 int score);

 const std::string& Name() const;
 int Score() const;

private:
 std::string name_;
 int score_;
};
```

const member functions

```
// .cc file

ScoreboardEntry::ScoreboardEntry()
: score_(0) { }

ScoreboardEntry::ScoreboardEntry(
  const std::string& name, int score)
: name_(name), score_(score) { }

const std::string& ScoreboardEntry::Name() const {
 return name_;
}

int ScoreboardEntry::Score() const {
 return score_;
}
```

Purpose of const Member Functions

- Principle of least privilege
- Only give permission to modify data members to functions that actually need to do that
- Prevents logic errors
  - Member function modifies variable by mistake
  - Caller does not expect data member to change
- **Best practice:** every member function is const, unless it has a specific need to modify a data member
  - Constructors initialize *all* data members, so can never be const
  - **Accessors are almost always const**

Examples of const Member Functions

- std::vector: empty, size
- Magick::ColorRGB: accessors for red, green, blue