Refactoring & Optimizing

4 - Cải tiến và tối ưu hóa
https://github.com/chauttm/advprogram

Nội dung

- Vấn đề tồn đọng
  - màn hình text bị trôi theo mỗi lần đánh → xấu,
  - chưa hiển thị các lần đoán sai để người chơi dễ hơn
  - code chưa tối ưu
- Tiếp tục cải tiến và tối ưu hóa
  - Phiên bản 1.1: code tốt hơn
  - Phiên bản 1.2: giao diện tốt hơn
- Kỹ thuật
  - Truyền tham số bằng giá trị, tham chiếu, tham chiếu hằng
  - Từ khóa **const**

Phiên bản 1.1: Cải tiến code

- Cơ chế truyền tham trị
- Tham biến
- Hằng tham chiếu
- Code trong sáng, an toàn

Cơ chế truyền tham trị

```
string update(string guessedWord, string secretWord, char guess)
{
    for (int i = secretWord.length(); i >= 0; i--) {
        if (secretWord[i] == guess) {
            guessedWord[i] = guess;
        }
    }
    return guessedWord;
}
```

Chuyện gì xảy ra với guessedWord và secretWord ở đây?

```
...//in main function
    if (contains(secretWord, guess))
        guessedWord = update(guessedWord, secretWord, guess);
...
```

Cơ chế truyền tham trị

```
string update(string guessedWord, string secretWord, char guess)
{
    for (int i = secretWord.length(); i >= 0; i--)
        if (secretWord[i] == guess) {
            guessedWord[i] = guess;
        }
    }
    return guessedWord;
}
```

```
...//in main function
    if (contains(secretWord, guess))
        guessedWord = update(guessedWord, secretWord, guess);
...
```

sao chép xâu ký tự:

- 02 lần từ đối số vào tham số
- 01 lần từ giá trị trả về vào biến nhận kết quả

Dùng tham biến để tránh sao chép

```
void update(string& guessedWord, string& secretWord, char guess)
{
    for (int i = secretWord.length(); i >= 0; i--) {
        if (secretWord[i] == guess) {
            guessedWord[i] = guess;
        }
    }
}
```

Đọc và ghi trực tiếp vào các string secretWord, guessedWord của main()

Không còn nhu cầu return

Không còn sao chép string

```
...//in main function
    if (contains(secretWord, guess))
        update(guessedWord, secretWord, guess);
...
```

Dùng tham biến để tránh sao chép

```
void update(string& guessedWord, string& secretWord, char guess)
{
    for (int i = secretWord.length(); i >= 0; i--) {


}
```

- Lợi ích của việc dùng tham biến
  - Giảm thời gian chạy do không phải sao chép dữ liệu
  - Giảm bộ nhớ do không phải tạo biến mới
  - Giảm nguy cơ lỗi khi dùng bộ nhớ động (sẽ quay lại sau)
- *Chú ý:*
  - Bài này chỉ để demo phương pháp. Việc cải tiến thời gian chạy với bài này không có ý nghĩa thực tiễn.
  - Thực tế chỉ cần quan tâm cải tiến chương trình chạy chậm so với nhu cầu

Dùng tham biến để tránh sao chép

```
void update(string& guessedWord, string& secretWord, char guess)
{
    for (int i = secretWord.length(); i >= 0; i--) {
```

- Nhược điểm:
  - Chưa cấm được hàm update() sửa dữ liệu không nên sửa, chẳng hạn word

```
}
```

- Cách giải quyết:
  - Khai báo const cho tất cả các tham số không được sửa

```
void update(string& guessedWord, const string& secretWord, char guess)
..
```

Lời khuyên

Để giảm bớt xử lý trong khi vẫn đảm bảo code an toàn:

Đối với các tham số không tầm thường

- Dùng tham chiếu đối với biến được ghi

  update(string& guessedWord,...)

- Dùng hằng tham chiếu đối với biến chỉ đọc

  update(... const string& secretWord)

Clean code

Hiện không còn gì để cải tiến

- Quy trình top-down + chia để trị đã cho ta tính mô đun hóa trong chương trình, các hàm được phân chia hợp lý
- Cách viết hàm theo kiểu kể chuyện kèm việc chú ý đặt tên biến tên hàm có nghĩa ngay từ đầu đã làm chương trình dễ hiểu
- Ta đã chú ý khai báo const cho tất cả các giá trị không được thay đổi → code đã an toàn

Nhu cầu cải tiến/refactor code sẽ xuất hiện khi ta tiếp tục sửa chương trình để cải thiện giao diện hoặc thêm tính năng mới (các bài sau)

Tại sao code cần trong sáng?

- “Chương trình chạy đúng” là yêu cầu không thể thiếu và quan trọng bậc nhất.
  Bên cạnh đó, còn có các tiêu chí khác rất hữu ích.
- “Code trong sáng dễ hiểu” giúp
  - dễ bảo trì,
  - dễ phát triển tiếp
  - dễ tìm lỗi khi chương trình chạy sai
  - giảm mắc lỗi trong khi lập trình, nhất là lỗi logic
    - Kết quả chấm bài làm GuessIt chơi nhiều lần: 100% các bài nộp có lỗi logic (liên tục sinh lại số cần đoán, sinh số cần đoán mỗi một lần...) đều là các bài dùng các vòng lặp lồng nhau thay vì tách hàm.

Tại sao code cần an toàn?

- Code an toàn giúp ta giảm nguy cơ lỗi
  - Vô tình sửa các biến không được sửa,
  - Sửa sai làm dữ liệu vi phạm ràng buộc
- Không phải một mình ta viết một chương trình, không thể tin tưởng người khác cẩn thận và biết hết những gì cần tránh.
- Không thể tin tưởng chính mình không bao giờ nhầm/quên

Phiên bản 1.2 (tự làm)

- Chống trôi màn hình
  - Có thể in nhiều dòng trống trước khi vẽ giá treo cổ để đẩy hẳn hình ảnh của lần đoán trước ra khỏi màn hình và cố định giá treo cổ mới tại đáy màn hình
- Hiển thị các chữ cái đã đoán sai
  - Thêm một biến string chứa các chữ cái đã đoán sai và cập nhật mỗi lần đoán sai, hiển thị mỗi lần chạy renderGame
- Chuẩn hóa chữ hoa chữ thường ở input,
  - ‘R’ hay ‘r’ đều là đoán đúng cho từ “car”
  - Gợi ý: thư viện <cctype> hàm tolower()