TRƯỜNG ĐẠI HỌC CÔNG NGHỆ - ĐHQGHN

Lập trình nâng cao ứng dụng trong đo lường và điều khiển

Khoa Cơ học kỹ thuật và Tự động hóa

Bài tập giữa kỳ chuẩn bị cho bài thi cuối kỳ

Dự kiến cách thức/nội dung

Thi vấn đáp, trình bày về chương trình tự viết:

Bài tập giữa kỳ:

Xây dựng chương trình kết nối với thiết bị đơn lẻ:

- Cảm biến -> hiển thị kết quả.
- Động cơ -> điều khiển vị trí, tốc độ, chiều quay.

Bài thi cuối kỳ:

Xây dựng chương trình kết nối quản lý nhiều hơn hai thiết bị phối hợp với nhau:

- Điều khiển nhiều động cơ (Robot).
- Kết nối nhiều cảm biến, tổng hợp dữ liệu, hiển thị kết quả.
- Sử dụng thông tin từ cảm biến để điều khiển động cơ.

Các bước thiết kế phần mềm

Giới thiệu

Khi đã đạt được một số kỹ năng, kiến thức cơ bản về lập trình (cú pháp, chế độ biên dịch, sử dụng thư viện, gỡ lỗi, làm việc theo cách tổ chức nhiều file, ...), cần tiếp tục tìm hiểu kỹ hơn về cách thiết kế chương trình.

Khi bắt đầu để viết một chương trình, sau khi đã có một vài ý tưởng, để hiện thực hóa ý tưởng cần viết một chương trình.

Các lập trình viên mới thường gặp khó khăn trong việc tìm ra cách chuyển đổi ý tưởng đó thành mã thực tế.

Tuy nhiên, thực ra chúng ta đã có nhiều kỹ năng giải quyết vấn đề có được từ cuộc sống hàng ngày.

Điều quan trọng nhất cần nhớ (và điều khó làm nhất) là cần phải thiết kế chương trình trước khi bạn bắt đầu viết mã.

Giới thiệu

Có 5 bước thiết kế chương trình:

- Bước 1: Định nghĩa/xác định mục tiêu/mục đích của chương trình
- Bước 2: Định nghĩa/xác định yêu cầu
- Bước 3: Xác định công cụ, đích đạt được và kế hoạch lưu trữ
- Bước 4: Phân nhỏ vấn đề khó giải quyết thành các vấn đề dễ giải quyết
- Bước 5: Đưa ra được chuỗi sự kiện của chương trình

Có 3 bước thực hiện:

- Xây dựng/tổ chức trước hàm chính
- Thực hiện từng hàm
- Kiểm tra cuối cùng

Một số lời khuyên khi viết chương trình

5 bước thiết kế chương trình

Bước 1: Định nghĩa/xác định mục tiêu/mục đích của chương trình

Để viết một chương trình thành công, trước tiên cần xác định mục tiêu của mình là gì.

Lý tưởng nhất là nêu điều này trong một hoặc hai câu. Nó thường hữu ích khi thể hiện điều này như một kết quả mà người dùng phải đối mặt. Ví dụ: Cho phép người dùng sắp xếp danh sách tên và số điện thoại liên quan. Tạo dungeon ngẫu nhiên sẽ tạo ra các hang động trông thú vị. Tạo danh sách đề xuất cổ phiếu cho các cổ phiếu có cổ tức cao. Lập mô hình trong bao lâu để một quả bóng rơi từ tháp rơi xuống đất. Mặc dù bước này có vẻ hiển nhiên, nhưng nó cũng rất quan trọng.

Cần tránh trường hợp khi viết một chương trình mà không biết rõ chương trình đó sẽ làm những gì người lập trình muốn!

5 bước thiết kế chương trình

Bước 2: Định nghĩa/xác định yêu cầu

Mặc dù việc xác định vấn đề giúp xác định kết quả mong muốn, nhưng điều này vẫn còn mơ hồ.

Bước tiếp theo là suy nghĩ cụ thể về các yêu cầu. Yêu cầu là một từ ngữ hoa mỹ để chỉ cả những ràng buộc mà giải pháp của người phát triển cần tuân theo (ví dụ: ngân sách, dòng thời gian, không gian, bộ nhớ, v.v.), cũng như các khả năng mà chương trình phải thể hiện để đáp ứng nhu cầu của người dùng.

Để cụ thể lưu ý rằng các yêu cầu của người phát triển nên tập trung vào "cái gì", không phải "cách thức/phương pháp". Ví dụ: Số điện thoại nên được lưu lại để có thể gọi lại sau này. Hầm ngục ngẫu nhiên phải luôn có một con đường để đi từ lối vào đến lối ra. Các khuyến nghị về cổ phiếu nên tận dụng dữ liệu giá lịch sử. Người dùng sẽ có thể nhập chiều cao của tháp. Chúng tôi cần một phiên bản có thể kiểm tra trong vòng 7 ngày. Chương trình sẽ đưa ra kết quả trong vòng 10 giây kể từ khi người dùng gửi yêu cầu của họ. Chương trình sẽ gặp sự cố trong ít hơn 0,1% phiên người dùng.

Một vấn đề đơn lẻ có thể tạo ra nhiều yêu cầu và giải pháp sẽ không được "hoàn thành" cho đến khi nó đáp ứng tất cả chúng.

5 bước thiết kế chương trình

Bước 3: Xác định công cụ, đích đạt được và kế hoạch lưu trữ

Với một lập trình viên có kinh nghiệm, có nhiều bước khác thường diễn ra vào thời điểm này, bao gồm:

1. Xác định kiến trúc mục tiêu và / hoặc hệ điều hành mà chương trình của bạn sẽ chạy trên đó.
2. Xác định bộ công cụ bạn sẽ sử dụng.
3. Xác định xem bạn sẽ viết chương trình của mình một mình hay là một phần của nhóm.
4. Xác định chiến lược thử nghiệm / phản hồi / phát hành của bạn.
5. Xác định cách bạn sẽ sao lưu mã của mình.

Tuy nhiên, là một lập trình viên mới, câu trả lời cho những câu hỏi này thường rất đơn giản: Bạn đang viết một chương trình để sử dụng riêng, một mình, trên hệ thống của riêng bạn, sử dụng IDE bạn đã mua hoặc tải xuống và mã của bạn có thể không được ai sử dụng nhưng bạn. Điều này làm cho mọi thứ trở nên dễ dàng.

5 bước thiết kế chương trình

Bước 3: Xác định công cụ, đích đạt được và kế hoạch lưu trữ

Nếu nhóm phát triển định làm bất cứ thứ gì có độ phức tạp không tầm thường, nên có kế hoạch sao lưu mã của mình. Chỉ nén hoặc sao chép thư mục vào một vị trí khác trên một máy là không đủ (mặc dù điều này tốt hơn là không có gì). Nếu hệ thống gặp sự cố, sẽ làm mất mọi thứ.

Một chiến lược sao lưu tốt bao gồm việc xóa hoàn toàn bản sao mã khỏi một hệ thống. Có rất nhiều cách dễ dàng để thực hiện việc này: Zip nó và gửi nó qua email của chính người phát triển, sao chép nó vào Dropbox hoặc một dịch vụ đám mây khác, FTP nó vào một máy khác, sao chép nó vào một máy khác trên mạng cục bộ của nhóm phát triển hoặc sử dụng hệ thống kiểm soát phiên bản ở trên một máy khác hoặc trong đám mây (ví dụ: github).

Hệ thống kiểm soát phiên bản có thêm lợi thế là không chỉ có thể khôi phục các tệp của nhóm mà còn có thể khôi phục chúng về phiên bản trước đó theo cách thuận tiện.

5 bước thiết kế chương trình

Bước 4: Phân nhỏ vấn đề khó giải quyết thành các vấn đề dễ giải quyết

Trong thế giới thực, con người thường phải thực hiện những công việc rất phức tạp. Cố gắng tìm ra cách thực hiện những nhiệm vụ này có thể rất khó khăn.

Để giải quyết những trường hợp như vậy, chúng ta thường sử dụng phương pháp giải quyết vấn đề là: Nghĩ từ trên xuống và Làm từ dưới lên.

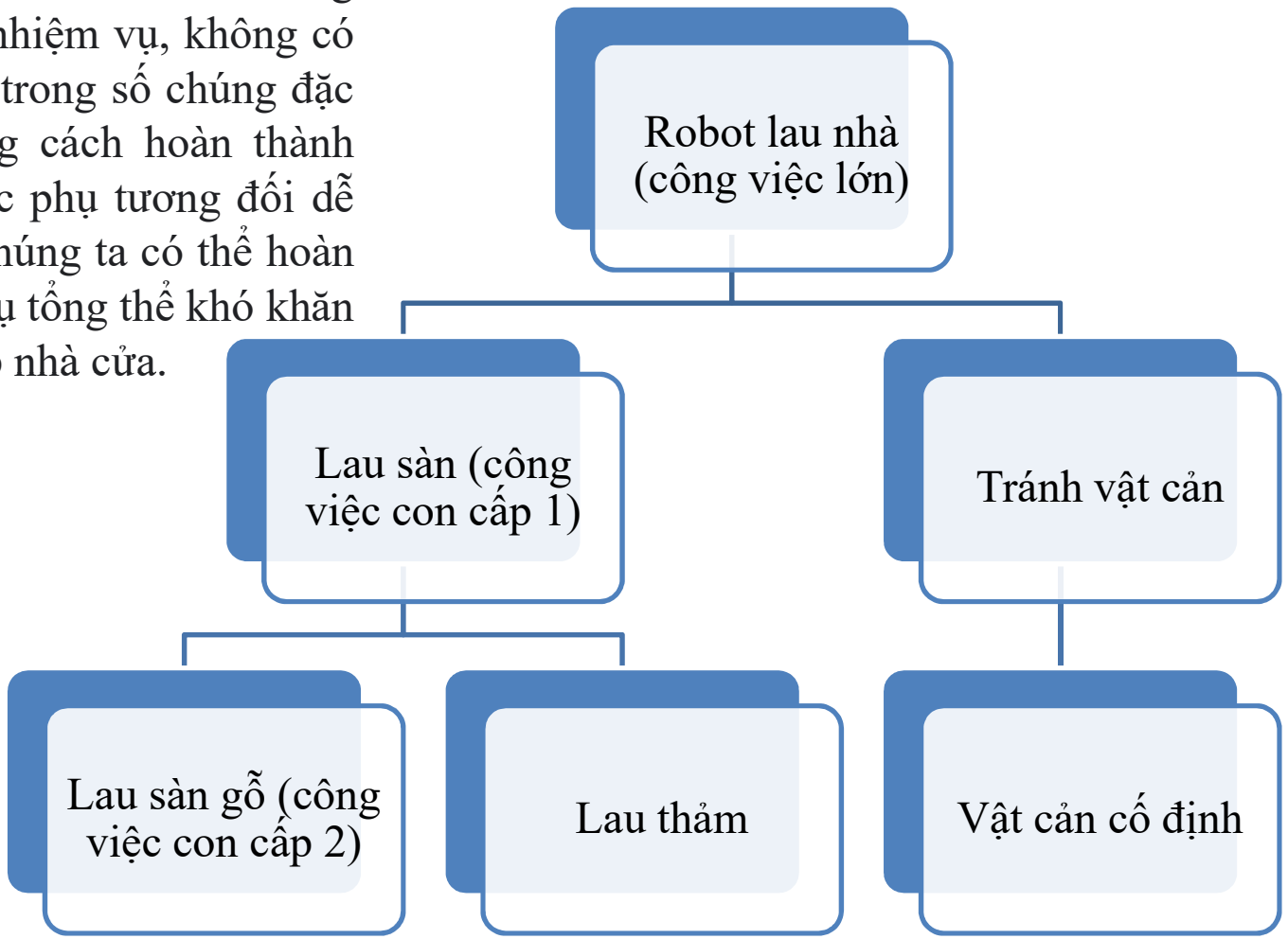
Nghĩa là, thay vì giải quyết một nhiệm vụ phức tạp duy nhất, người phát triển chia nhiệm vụ đó thành nhiều nhiệm vụ con, mỗi nhiệm vụ riêng lẻ sẽ dễ giải quyết hơn.

Nếu những nhiệm vụ phụ đó vẫn còn quá khó để giải quyết, chúng có thể được chia nhỏ hơn nữa.

Bằng cách liên tục chia các nhiệm vụ phức tạp thành các nhiệm vụ đơn giản hơn, cuối cùng có thể đạt đến điểm mà mỗi nhiệm vụ riêng lẻ đều có thể quản lý được, nếu không muốn nói là nhỏ nhặt.

5 bước thiết kế chương trình

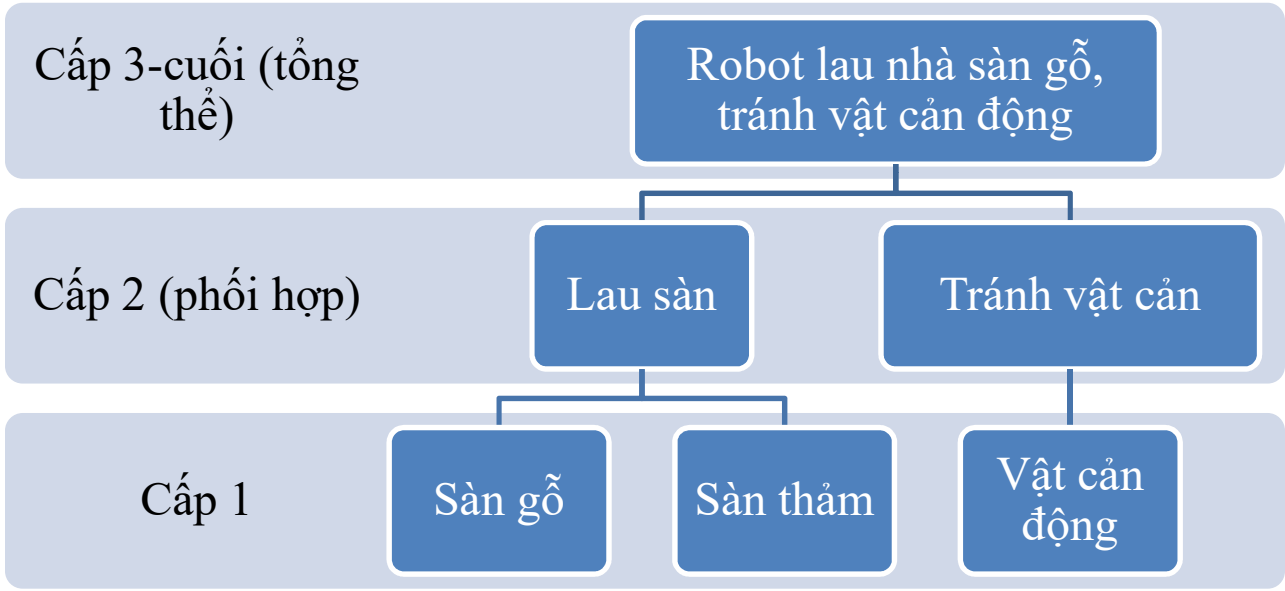
Bây giờ chúng ta có một hệ thống phân cấp các nhiệm vụ, không có nhiệm vụ nào trong số chúng đặc biệt khó. Bằng cách hoàn thành từng hạng mục phụ tương đối dễ quản lý này, chúng ta có thể hoàn thành nhiệm vụ tổng thể khó khăn hơn là dọn dẹp nhà cửa.



5 bước thiết kế chương trình

Bước 4: Phân nhỏ vấn đề khó giải quyết thành các vấn đề dễ giải quyết

Một cách khác để tạo hệ thống phân cấp nhiệm vụ là làm như vậy **từ dưới lên**. Trong phương pháp này sẽ bắt đầu từ một danh sách các nhiệm vụ đơn giản và xây dựng hệ thống phân cấp bằng cách nhóm chúng lại.



Sử dụng phương pháp từ dưới lên, chúng ta có thể sắp xếp chúng thành một hệ thống phân cấp các mục bằng cách tìm cách nhóm các mục có điểm tương đồng với nhau.

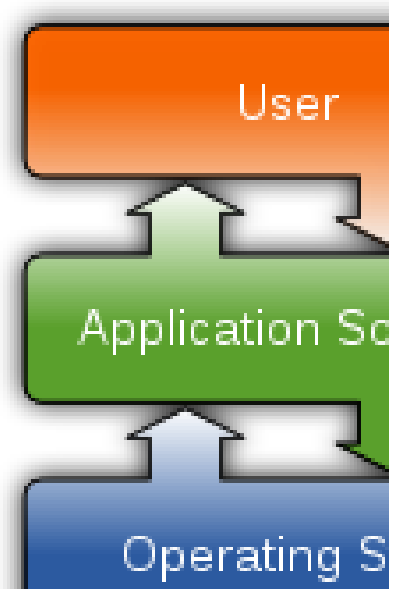
5 bước thiết kế chương trình

Thiết kế cấu trúc phân cấp nhiệm vụ cực kỳ hữu ích trong lập trình, bởi vì khi có trong tay hệ thống phân cấp nhiệm vụ, tức là về cơ bản bạn đã xác định cấu trúc của chương trình tổng thể.

Nhiệm vụ cấp cao nhất (trong trường hợp này là "Dọn dẹp nhà cửa") trở thành hàm chính (vì nó là vấn đề đang cần cố gắng giải quyết). Các hàm nhỏ/thành phần trở thành các hàm trong chương trình (hàm chính).

Điều này cho phép khi một trong các thành phần (hàm) quá khó thực hiện, ta chỉ cần chia mục đó thành nhiều mục / hàm con.

Cuối cùng, người phát triển sẽ đạt đến điểm mà mỗi chức năng thành phần trong chương trình không thể thực hiện được nếu không có sự tổ chức thành tổng thể.



5 bước thiết kế chương trình

Bước 5: Đưa ra được chuỗi sự kiện của chương trình

Sau khi thực hiện Bước 4 chương trình đã có cấu trúc, lúc này là lúc xác định cách liên kết tất cả các nhiệm vụ thành phần/ hàm thành phần với nhau.

Bước đầu tiên là xác định chuỗi sự kiện sẽ được thực hiện. Ví dụ, khi bật nguồn robot lau nhà, robot sẽ thực hiện các công việc trên theo trình tự nào? Ví dụ một trình tự là:

-Kết nối mạng.

-Tìm kiếm chương trình.

-Đợi lệnh từ người dùng.

-Lau nhà.

-Tránh vật cản.

-Xây dựng bản đồ.

-Kiểm tra pin.

-Về dock sạc.

3 bước thực hiện coding

Bước 1: Xây dựng/tổ chức trước hàm chính

Bây giờ đã sẵn sàng để bắt đầu triển khai. Các trình tự trên có thể được sử dụng để phác thảo chương trình chính. Đừng lo lắng về đầu vào và đầu ra trong lúc này.

Lưu ý rằng nếu định sử dụng phương pháp "phác thảo" này để xây dựng chương trình, thì các hàm thành phần sẽ không biên dịch vì chưa có định nghĩa. Nhận xét về các lệnh gọi hàm cho đến khi bạn sẵn sàng triển khai các định nghĩa hàm là một cách để giải quyết vấn đề này. Ngoài ra, có thể khai báo các hàm tự tạo (tạo các hàm giữ chỗ với các phần thân trống) để chương trình sẽ biên dịch.

3 bước thực hiện coding

Bước 1: Xây dựng/tổ chức trước hàm chính

```
int main()
{
//    doBedroomThings();
//    doBathroomThings();
//    doBreakfastThings();
//    doTransportationThings();

    return 0;
}
```

Or in the case of the calculator:

```
int main()
{
    // Get first number from user
//    getUserInput();

    // Get mathematical operation from user
//    getMathematicalOperation();

    // Get second number from user
```

3 bước thực hiện coding

Bước 2: Thực hiện từng hàm

Trong bước này, đối với mỗi chức năng, sẽ cần thực hiện ba việc:

1. Xác định nguyên mẫu hàm (đầu vào và đầu ra)
2. Viết hàm
3. Kiểm tra chức năng

Nếu các chức năng đủ chi tiết, thì mỗi chức năng phải khá đơn giản và dễ hiểu. Nếu một chức năng nhất định vẫn có vẻ quá phức tạp để triển khai, có lẽ nó cần được chia nhỏ thành các chức năng con để có thể dễ dàng triển khai hơn (hoặc có thể bạn đã làm điều gì đó không đúng thứ tự và cần xem lại trình tự các sự kiện của mình).

3 bước thực hiện coding

```
1  #include <iostream>
2
3  // Full implementation of the getUserInput function
4  int getUserInput()
5  {
6      std::cout << "Enter an integer ";
7      int input{};
8      std::cin >> input;
9
10     return input;
11 }
12
13 int main()
14 {
15     // Get first number from user
16     int value{ getUserInput() }; // Note we've included code here to test the return valu
17     std::cout << value; // debug code to ensure getUserInput() is working, we'll remove t
18
19     // Get mathematical operation from user
20 //    getMathematicalOperation();
21
22     // Get second number from user
```

3 bước thực hiện coding

Đầu tiên, xác định rằng hàm getUserInput không có đối số và sẽ trả về giá trị int trở lại trình gọi. Điều đó được phản ánh trong nguyên mẫu hàm có giá trị trả về là int và không có tham số.

Tiếp theo, viết phần nội dung của hàm, đó là 4 câu lệnh đơn giản.

Cuối cùng, triển khai một số mã tạm thời trong hàm main để kiểm tra xem hàm getUserInput (bao gồm cả giá trị trả về của nó) có hoạt động chính xác hay không.

Có thể chạy chương trình này nhiều lần với các giá trị đầu vào khác nhau và đảm bảo rằng chương trình đang hoạt động như mong đợi. Nếu tìm thấy thứ gì đó không hoạt động, sẽ dễ dàng biết được vấn đề nằm ở đoạn mã nào vừa viết.

Khi chương trình chắc chắn hoạt động như dự định, lúc này có thể xóa mã thử nghiệm tạm thời và tiến hành triển khai hàm tiếp theo (chức năng getMatheatologyOperation). Hãy nhớ: Không triển khai toàn bộ chương trình trong một lần. Làm việc theo từng bước, kiểm tra từng bước trước khi tiếp tục.

3 bước thực hiện coding

Bước 3: Kiểm tra cuối cùng

Khi chương trình đã "hoàn thành", bước cuối cùng là kiểm tra toàn bộ chương trình và đảm bảo nó hoạt động như dự kiến. Nếu nó không hoạt động, hãy sửa nó.

Một số lời khuyên khi viết chương trình

Giữ cho các chương trình bắt đầu từ đơn giản. Thông thường các lập trình viên mới có tầm nhìn rộng lớn về tất cả những điều họ muốn chương trình của họ thực hiện. "Tôi muốn viết một trò chơi nhập vai với đồ họa và âm thanh cùng những quái vật và ngục tối ngẫu nhiên, với một thị trấn bạn có thể ghé thăm để bán những vật phẩm bạn tìm thấy trong ngục tối".

Nếu cố gắng viết một thứ gì đó quá phức tạp để bắt đầu, sẽ trở nên choáng ngợp và chán nản vì sự thiếu tiến bộ. Thay vào đó, hãy làm cho mục tiêu đầu tiên càng đơn giản càng tốt, một cái gì đó chắc chắn nằm trong tầm tay. Ví dụ: "Tôi muốn có thể hiển thị trường 2 chiều trên màn hình".

Một số lời khuyên khi viết chương trình

Thêm các tính năng theo thời gian. Khi đã có một chương trình đơn giản hoạt động tốt, lúc này có thể thêm các tính năng cho nó. Ví dụ: khi bạn có thể hiển thị trường của mình, hãy thêm một nhân vật có thể đi bộ xung quanh. Khi bạn có thể đi lại, hãy thêm những bức tường có thể cản trở tiến trình của bạn. Khi bạn đã có tường, hãy xây một thị trấn đơn giản từ chúng. Sau khi bạn có một thị trấn, hãy thêm thương nhân.

Bằng cách thêm dần từng tính năng, chương trình sẽ ngày càng phức tạp hơn mà không làm choáng ngợp trong quá trình này.

Một số lời khuyên khi viết chương trình

Tập trung vào một khu vực tại một thời điểm. Đừng cố gắng viết mã mọi thứ cùng một lúc và đừng phân chia sự chú ý của bạn cho nhiều nhiệm vụ. Tập trung vào một nhiệm vụ tại một thời điểm. Sẽ tốt hơn nhiều nếu có một nhiệm vụ đang làm việc và năm nhiệm vụ chưa được bắt đầu hơn là sáu nhiệm vụ đang hoạt động một phần. Nếu bạn chia nhỏ sự chú ý của mình, bạn có nhiều khả năng mắc sai lầm và quên những chi tiết quan trọng.

Một số lời khuyên khi viết chương trình

Kiểm tra từng đoạn mã khi bạn thực hiện. Các lập trình viên mới thường sẽ viết toàn bộ chương trình trong một lần chuyển. Sau đó, khi họ biên dịch nó lần đầu tiên, trình biên dịch báo cáo hàng trăm lỗi. Điều này có thể không chỉ đáng sợ, nếu mã của bạn không hoạt động, có thể khó tìm ra lý do tại sao. Thay vào đó, hãy viết một đoạn mã, sau đó biên dịch và kiểm tra nó ngay lập tức. Nếu nó không hoạt động, bạn sẽ biết chính xác vị trí của vấn đề và sẽ dễ dàng khắc phục. Khi bạn chắc chắn rằng mã hoạt động, hãy chuyển sang phần tiếp theo và lặp lại. Có thể mất nhiều thời gian hơn để viết xong mã của bạn, nhưng khi bạn hoàn thành, toàn bộ công việc sẽ hoạt động và bạn sẽ không phải mất gấp đôi thời gian để cố gắng tìm ra lý do tại sao không.

Một số lời khuyên khi viết chương trình

Đừng đầu tư vào việc hoàn thiện mã sớm. Bản thảo đầu tiên của một tính năng (hoặc chương trình) hiếm khi tốt. Hơn nữa, các chương trình có xu hướng phát triển theo thời gian, khi bạn thêm các khả năng và tìm ra những cách tốt hơn để cấu trúc mọi thứ. Nếu bạn đầu tư quá sớm vào việc đánh bóng mã của mình (thêm nhiều tài liệu, tuân thủ đầy đủ các phương pháp hay nhất, thực hiện tối ưu hóa), bạn có nguy cơ mất tất cả khoản đầu tư đó khi cần thay đổi mã. Thay vào đó, hãy làm cho các tính năng của bạn hoạt động ở mức tối thiểu và sau đó tiếp tục. Khi bạn tự tin vào giải pháp của mình, hãy thoa các lớp sơn bóng liên tiếp. Đừng nhắm đến mục tiêu hoàn hảo - các chương trình không tầm thường không bao giờ hoàn hảo và luôn có điều gì đó khác có thể được thực hiện để cải thiện chúng. Hãy đến đủ tốt và tiếp tục.

Một số lời khuyên khi viết chương trình

Hầu hết các lập trình viên mới sẽ viết tắt nhiều bước và đề xuất này (bởi vì nó có vẻ như rất nhiều công việc và / hoặc nó không thú vị bằng viết mã). Tuy nhiên, đối với bất kỳ dự án không tầm thường nào, việc làm theo các bước sau chắc chắn sẽ giúp bạn tiết kiệm rất nhiều thời gian về lâu dài. Lập kế hoạch trước một chút sẽ tiết kiệm rất nhiều việc gỡ lỗi ở cuối. Tin tốt là một khi bạn cảm thấy thoải mái với tất cả những khái niệm này, chúng sẽ bắt đầu đến với bạn một cách tự nhiên hơn. Cuối cùng, bạn sẽ đạt được điểm mà bạn có thể viết toàn bộ các hàm mà không cần lập kế hoạch trước.