Bài tập 2: Crypto

Mục tiêu
+ Làm quen với các hàm và thư viện.
+ Học cách viết mật mã.

Tài liệu nên đọc
+ Từ trang 11 – 14 và trang 39 của http://www.howstuffworks.com/c.htm.
+ Chương 6, 7, 10, 17, 19, 21, 22, 30, và 32 của cuốn "Absolute Beginner's Guide to C"
+ Chương 7,8 và 10 của "Programming in C"

Tiêu chuẩn đánh giá
Bài làm của bạn sẽ được đánh giá trên 4 tiêu chí
Phạm vi: Mức độ mà code của bạn có thể thực hiện được những yêu cầu được nêu ra.
Độ chính xác: Độ phù hợp của code các bạn viết, dựa trên thông số chúng tôi đưa ra, cũng như mức độ lỗi của chương trình.
Thiết kế: Đoạn code của bạn viết có tốt hay ko (như là: độ rõ ràng, hiệu quả, hợp lý và logic)
Cách viết: Đoạn code của bạn có dễ đọc hay không (nghĩa là bạn cần ghi chú thích, chú ý cách đầu dòng và tên biến phải rõ ràng)

Sẵn sàng rồi chứ?
Đầu tiên, xem những đoạn phim ngắn về các vòng lặp, hàm, mã Caesar và đối số dòng lệnh:
https://youtu.be/HHmiHx7GGLE
https://youtu.be/Pi0Yf-jn7O8
https://youtu.be/36xNpbosfTY
https://youtu.be/X8PmYwnbLKM

Hãy chắc chắn rằng bạn khá thoải mái với việc trả lời những câu hỏi dưới đây khi nộp bài tập này!
+ Vòng lặp while khác vòng lặp do-while như thế nào? Khi nào thì vòng lặp do-while đặc biệt hữu ích?
+ "Biến chưa khai báo" thường biểu thị điều gì nếu được xuất ra bởi "make" (hoặc, "clang")?
+ Tại sao mật mã Caesar không được bảo mật cho lắm?
+ Hàm là gì?
+ Tại sao phải bận tâm đến việc viết hàm trong khi bạn có thể chỉ copy và paste đoạn code cần thiết?

Tiếp theo, hãy tự xem qua những ví dụ của tuần 2, mã nguồn của chúng có thể tìm được tại http://cdn.cs50.net/2014/fall/lectures/2/m/src2m/ và http://cdn.cs50.net/2014/fall/lectures/

2/w/src2w/, đi nhanh qua bất kì chương trình nào bạn đã thoải mái:
https://youtu.be/9zoRoz8Pq4E?list=PLhQjrBD2T380bVx_CQ2EXtlqZh0frwOsn

Cuối cùng, xem kĩ đối số dòng lệnh qua những ví dụ bổ sung từ tuần 3, mã nguồn cho các ví dụ này có thể được tìm thấy tại http://cdn.cs50.net/2014/fall/lectures/3/m/src3m/:
https://youtu.be/1VbHJz2L6dM
https://youtu.be/Ja8YoR-u9TA
https://youtu.be/mXj188eyRFE

Bắt đầu
Được rồi, hãy làm quen với chuỗi.

Trong file "initials.c", viết một chương trình yêu cầu tên người dùng (dùng "GetString" để thu được tên người dùng dưới dạng "chuỗi") rồi xuất những chữ cái đầu tiên dưới dạng viết hoa, không giãn cách, không chấm câu, tiếp theo là một dòng mới (\n) và không viết gì thêm. Bạn có thể cho rằng input của người dùng sẽ chỉ chứa các chữ cái (viết hoa hoặc viết thường) cộng với những khoảng cách đơn giữa các từ. Những người như "Joseph Gordon-Levitt", "Conan O'Brien" và "David J.Malan" sẽ không sử dụng chương trình của bạn.

Để tự động kiểm tra mã của bạn, chương trình của bạn phải chạy mỗi ví dụ dưới đây. Giả định rằng đoạn được gạch chân là những gì người dùng đã nhập.

```
jharvard@appliance (~/Dropbox/pset2): ./initials
Zamyla Chan
ZC

jharvard@appliance (~/Dropbox/pset2): ./initials
robert thomas bowden
RTB
```

Nếu bạn muốn kiểm tra tính chính xác của chương trình bằng check50, bạn có thể tiến hành như sau.

```
check50 2014.fall.pset2.initials initials.c
```

Nếu các bạn muốn xem chương trình initials do chúng tôi viết

trong ứng dụng này, các bạn có thể chạy dòng dưới đây:

~cs50/pset2/

Caesar vạn tuế!

Hãy nhớ lại đoạn phim ngắn của David DiCiurcio, mật mã Caesar đã mã hóa các thông điệp bằng cách xoay mỗi chữ cái đến các vị trí k, từ A đến Z (cf. http://en.wikipedia.org/wiki/Caesar_cipher). Nói cách khác, nếu p là một văn bản thuần (vd: một thông điệp chưa được mã hóa) pi là kí tự thứ i trong p, và k là một khóa bí mật (vd: một số nguyên không âm), thì mỗi chữ cái, ci, trong văn bản mã hóa, c, được tính như sau:

ci = (pi + k) % 26

Công thức này có thể khiến mật mã trông có vẻ phức tạp hơn vốn dĩ, nhưng nó thật sự là một cách hay để diễn tả thuật toán chính xác và ngắn gọn. Và các nhà khoa học máy tính rất thích sự chính xác và ngắn gọn.

Ví dụ, giả sử khóa bí mật k là 13 và văn bản thuần p là "Nhớ uống Ovaltine!" Hãy mã hóa p với k để có được một văn bản mã hóa c bằng cách dịch chuyển từng chữ cái trong p đến vị trí cách nó 13 chữ, bằng cách:

Nhớ uống Ovaltine!

thành:

Ytấ êăyr Ahlevuyr!

Ta đã cố ý in những chữ trên trong font đơn cách để tất cả các chữ cái đều đứng thẳng hàng. Chú ý Y (chữ cái đầu tiên trong văn bản mã hóa) cách N (chữ cái đầu tiên trong văn bản thuần) 13 chữ . Tương tự, t (chữ cái thứ 2 trong văn bản mã hóa) cách h (chữ cái thứ 2 trong văn bản thuần) 13 chữ. Trong khi đó, ấ (chữ cái thứ 3 trong văn bản mã hóa) cách ớ (chữ cái thứ 3 trong văn bản thuần) 13 chữ, mặc dù chúng ta phải tính từ a đến y để có được kết quả. Và cứ tiếp tục. Đây không phải là

mật mã an toàn nhất, chắc chắn là vậy, nhưng cũng khá vui để thực hiện!

Nhân tiện, mã Caesar với khóa 13 thường được gọi là ROT13 (cf. http://en.wikipedia.org/wiki/ROT13). Tuy nhiên, trong thế giới thực, có lẽ sẽ tốt nhất khi dùng ROT26, được tin là có tính bảo mật gấp đôi.

Dù sao thì mục tiêu tiếp theo của bạn là viết trong "caesar.c" một chương trình mã hóa các thông điệp bằng cách sử dụng mã Caesar. Chương trình của bạn phải chấp nhận một đối số dòng lệnh duy nhất: một số nguyên không âm. Hãy gọi nó là k để thuận tiện khi thảo luận. Nếu chương trình của bạn chạy mà không có đối số dòng lệnh nào hoặc có nhiều hơn 1 đối số dòng lệnh, chương trình của bạn có thể sẽ hét vào người dùng và trả lại giá trị "1" (thường biểu thị lỗi) ngay lập tức qua câu lệnh sau:

return 1;

Nếu không, chương trình của bạn phải tiến hành yêu cầu người dùng nhập một chuỗi văn bản thuần và xuất văn bản đó ra với từng kí tự là chữ cái được dịch chuyển theo vị trí k; các kí tự không phải chữ cái phải được xuất ra y nguyên. Sau khi xuất văn bản mã hóa, chương trình của bạn phái thoát ra, với "main" trở lại thành "0", qua câu lệnh sau:

return 0;

Nếu bạn không dứt khoát trả lại "int" từ trong "main", 0 được tự động trả lại cho bạn. (Đúng vậy, với mỗi "return type" của nó, "main" cần trả lại một "int". Nhưng vào lúc khác sẽ nhiều hơn). Vì bây giờ bạn đang trả lại 1 để biểu thị lỗi, tốt nhất là trả lại 0 (bằng quy ước) để biểu thị thành công. Trong khi 0 thường diễn tả thành công, bất cứ int không phải 0 nào cũng thường biểu thị lỗi. Bằng cách đó, bạn có thể biểu thị trên 4 tỉ lỗi (vì một int thường là 32 bits)!

Tuy nhiên, dù chỉ tồn tại 26 chữ cái trong bảng chữ cái tiếng Anh, bạn có thể không giả sử rằng k sẽ nhỏ hơn hoặc bằng 26; chương trình của bạn nên hoạt động được cho tất cả các giá trị nguyên không âm của k nhỏ hơn 231 – 26. (Nói cách khác, bạn không phải lo chương trình của mình cuối cùng sẽ hỏng nếu người dùng chọn một giá trị k quá lớn hoặc gần như quá lớn để vừa hàm int. Bây giờ, thậm chí nếu k lớn hơn 26, các kí tự theo bảng chữ cái trong input của chương trinh nên được giữ

nguyên trong output của chương trình. Ví dụ, nếu k là 27, A không thể trở thành "[" mặc dù "[" cách A 27 kí tự trong ASCII; A nên trở thành B, vì 27 mod 26 bằng 1, các nhà khoa học máy tính có thể nói như vậy. Nói cách khác, các giá trị như k = 1 và k = 27 là tương đương nhau..

Chương trình của bạn phải giữ nguyên chữ viết hoa: các chữ cái được viết hoa dù được chuyển đổi vị trí vẫn phải giữ nguyên dạng viết hoa; các chữ cái viết thường dù được chuyển đổi vị trí vẫn phải giữ nguyên dạng viết thường.

Bắt đầu từ đâu đây? Chương trình này cần chấp nhận một đối số dòng lệnh k, vậy nên lần này bạn sẽ muốn khai báo main bằng:

```
int main(int argc, string argv[])
```

Hãy nhớ lại rằng "argv" là mảng chứa các chuỗi. Các bạn có thể coi mảng này như một hàng tủ chứa đồ trong phòng tập thể dục, bên trong mỗi tủ lại chứa vài giá trị nào đó (và vài đôi vớ nữa). Đối với trường hợp này, trong mỗi tủ như vậy là một chuỗi. Để mở (hay là index into) cái tủ đầu tiên, các bạn dùng cú pháp argv[0], vì các mảng luôn mang trị số bằng 0. Để mở tủ tiếp theo, các bạn dùng cú pháp argv[1]. Và cứ tiếp tục như thế. Tất nhiên, nếu có n tủ thì bạn nên dừng việc mở tủ khi đạt tới argv[n-1], vì argv[n] không tồn tại! (Hoặc là như thế hoặc là nó thuộc về một người khác, và nếu là vậy thì các bạn không nên mở nó.)

Vậy bạn có thể truy cập k với mã như

```
string k = argv[1];
```

giả sử rằng nó thật sự ở đó! Hãy nhớ lại rằng argc là một int bằng số chuỗi trong argv, nên tốt nhất là bạn nên kiểm tra giá trị của argc trước khi mở chiếc tủ có thể không tồn tại! Lý tưởng nhất, argc sẽ là 2. Vì sao ư? Hãy nhớ trong argv[0], theo mặc định, là tên riêng của chương trình. Vậy argc sẽ luôn ít nhất là bằng 1. Nhưng đối với chương trình này, bạn muốn người dùng cung cấp một đối số dòng lệnh k, trong trường hợp này thì argc nên là 2.

Bây giờ, chỉ vì người dùng nhập một số nguyên ở dấu nháy không có nghĩa là input của họ sẽ tự động được lưu trữ trong một int. Ngược lại, nó sẽ được lưu trữ như một chuỗi vô tình trông giống int! Và bạn sẽ cần đổi chuỗi đó thành một int thật sự. Rốt cuộc, một hàm atoi tồn tại chỉ vì mục đích đó. Đây là cách

bạn có thể dùng nó:

```
int k = atoi(argv[1]);
```

Lưu ý, lần này, chúng ta đã khai báo k là một int để bạn có thể thực hiện các thuật toán với nó. A, khá hơn rồi đấy. Nhân tiện, các bạn có thể tự giả sử rằng người dùng sẽ chỉ nhập các số nguyên ở dấu nháy. Các bạn không cần phải lo lắng họ sẽ nhập, ví dụ như, "foo" để gây khó khăn thêm (mặc dù đáp án của chúng tôi thật sự tính cả tình huống này); atoi sẽ trở về 0 trong những trường hợp như vậy.

Vì atoi được khai báo trong stdlib.h, bạn sẽ muốn #include file đầu trang đó lên đầu mã của chính bạn. (Về cơ bản, mã của bạn sẽ biên dịch mà không có nó ở đó, vì ta đã #include nó trong cs50.h rồi. Nhưng tốt nhất là đừng tin một thư viện khác sẽ #include các file đầu trang mà bạn biết là mình cần.)

Được rồi, một khi bạn đã lưu trữ k thành một int, bạn sẽ cần phải hỏi người dùng nhập văn bản thuần nào đó. Có thể GetString của chính CS50 sẽ giúp bạn.

Khi bạn đã có cả k và văn bản thuần, đã đến lúc để mã hóa văn bản thuần bằng k. Hãy nhớ rằng bạn có thể lặp lại các kí tự trong một chuỗi, in từng kí tự một, bằng mã như sau:

```
for (int i = 0, n = strlen(p); i < n; i++)
{
printf("%c", p[i]);
}
```

Nói một cách khác, argv là một mảng chứa các chuỗi, tương tự như vậy, chuỗi là một mảng chứa các ký tự. Cho nên, các bạn có thể dùng các dấu [ ] để tiếp cận từng ký tự một trong một chuỗi, giống y như cách các bạn tiếp cận các chuỗi trong mảng. Quá gọn, phải không? Dĩ nhiên, in từng ký tự trong một chuỗi ra không hẳn sẽ được tính là mã hoá. Nhưng cũng có thể nếu trên căn bản k là 0. Nói chung là phần ở trên đã có thể giúp các bạn hỗ trợ Cesar thiết kế mật mã của mình! Vì Cesar!

Nhân tiện, các bạn sẽ cần phải #include thêm một tập tin mở đầu để có thể sử dụng strlen

Và Zamyla cũng có một số gợi ý cho các bạn:
https://youtu.be/V6IDxl-3WAA

Để chúng tôi có thể tự động kiểm tra code của các bạn, chương trình của các bạn thực hiện mỗi một phần dưới đây. Giả sử những phần gạch dưới là những gì người dùng gõ vào.

```
jharvard@appliance (~/Dropbox/pset2): ./caesar 13
Nhớ uống Ovaltine!
Ytấ êăyr Ahlevuyr!
```

Bên cạnh atoi, các bạn có thể tìm thấy một số hàm tiện dụng đã được lưu lại trên https://reference.cs50.net/ dưới mục ctype.h và stdlib.h. Ví dụ như isdigit, nghe có vẻ thú vị đây. Và, sẵn nhắc tới việc wrapping (gói) các ký tự từ Z tới A (hoặc z tới a), các bạn đừng quên %, phép chia lấy số dư trong C. Các bạn có thể tra khảo thêm ở http://asciitable.com/, để tìm các mã ASCII cho những ký tự khác ngoài bảng chữ cái, trong trường hợp các bạn in nhầm ký tự nào đó.

Nếu bạn muốn kiểm tra tính chính xác của chương trình bằng check50, bạn có thể tiến hành như sau.

```
check50 2014.fall.pset2.caesar caesar.c
```

Và nếu các bạn muốn vọc một chút cách mà chúng tôi thiết kế bài caesar này, các bạn có thể chạy phần dưới đây.

```
~cs50/pset2/caesar
```

Parlez-vous français? (Các bạn có nói tiếng Pháp không?)

À, mật mã trước chả có chút gì là bảo mật. May mắn là chúng ta có một thuật toán tinh vi hơn ở đây: http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher. Mà các bạn đừng có bị mất tập trung bởi chủ đề thảo luận về tabula recta của bài viết đó. Mỗi ci có thể được tính toán bằng những thuật toán tương đối căn bản! Các bạn thậm chí không cần dùng tới mảng hai chiều.

Mật mã Vigenère củng cố hơn Caesar bởi nó mã hoá thông điệp

thông qua một dãy khoá (hay, theo một cách khác, là một từ khoá). Nói cách khác, nếu p là văn bản thuần nào đó và k là từ khoá (ví dụ, cho một chuỗi ký tự chữ cái, với 0 đại diện cho A và a, trong khi 25 đại diện cho z và Z), vậy thì mỗi một chữ, ci, trong dòng mật mã sẽ được tính toán theo cách sau:

ci = (pi + kj) % 26

Chú ý cách đoạn mã này dùng kj thay vì chỉ mỗi một mình k. Và các bạn nên nhớ. nếu k ngắn hơn p, vậy thì các chữ trong k phải được tái sử dụng nhiều lần cho tới khi p được mã hoá xong.

Thử thách cuối cùng trong tuần này của các bạn là viết, trong "vinegere.c", một chương trình mã hoá thông điệp sử dụng bộ mã Vinegère. Chương trình này phải chấp nhận một đối số dòng lênh đơn: một từ khoá, k, chỉ chứa các ký tự chữ cái. Nếu ai đó chạy chương trình của các bạn mà không sử dụng một đối số dòng lệnh đơn, sử dụng nhiều hơn một đối số dòng lệnh đơn, hoặc một đối số dòng lệnh đơn có chứa các ký tự không nằm trong bảng chữ cái, thì chương trình của các bạn phải báo lỗi và tự thoát ra, với giá trị "main" trở về 1 (để báo hiệu lỗi cho chương trình của chúng tôi kiểm tra). Nếu không, chương trình của các bạn phải tiếp tục nhắc người dùng nhập một chuỗi văn bản thuần, p, để tiếp tục mã hoá dựa theo bộ mã của Vinegère với k, cuối cùng in kết quả ra rồi thoát. Giá trị "main" phải trở về 0.

Đối với các ký tự của k, các bạn phải quy định cho a và A là 0, b và B là 1,.... cho tới z và Z bằng 25. Thêm vào đó, chương trình của các bạn chỉ được phép áp dụng bộ mã Vinegère cho các ký tự chữ cái trong p. Tất cả các ký tự khác (chữ số, biểu tượng, khoảng trống, các dấu câu, v.v...) phải được xuất ra nguyên vẹn không thay đổi. Hơn nữa, nếu đoạn mã của các bạn muốn chuyển chữ cái thứ tự j của k thành chữ cái thứ tự i của p, nhưng ký tự i đó không thuộc bảng chữ cái, thì các bạn phải chuyển ký tự j đó thành chữ cái tiếp theo trong các chữ cái xuất hiện trong p; các bạn không được phép bỏ qua chữ cái đó. Cuối cùng các bạn phải giữ nguyên chữ viết thường và viết hoa của mỗi ký tự trong p.

Không biết phải bắt đầu từ đâu? May cho các bạn là chương trình này cũng khá là tương tự với "caesar"! Chỉ là lần này, các bạn phải quyết định xem sẽ sử dụng ký tự nào trong k khi lặp lại các ký tự đó trong p.

Và lại là Zamyla đây, với một số gợi ý:
https://youtu.be/Uma2HZMPm2M

Để chúng tôi có thể tự chạy kiểm tra code của các bạn, chương trình của các bạn phải thực hiện từng phần sau đây; các phần in đậm là các phần nhập mẫu.

jharvard@appliance (~/Dropbox/pset2): ./vigenere bacon
Meet me at the park at eleven am (Gặp tôi ở công viên lúc mười một giờ sáng)
Negh zf av huf pcfx bt gzrwep oz

Làm thế nào để kiểm tra chương trình của mình, bên cạnh việc dự đoán kết quả sẽ được xuất ra, nếu cho các bạn một số dữ liệu được nhập vào? À, nhắc lại cho các bạn biết chúng tôi là những trợ giảng tốt bụng. Cho nên chúng tôi đã viết sẵn một chương trình gọi là "devinegere". Nó chỉ chấp nhận một đối số dòng lệnh duy nhất (một từ khoá), nhưng nó có thể nhận một dòng mật mã nhập vào rồi xuất một văn bản thuần ra.

Để sử dụng chương trình của chúng tôi, chạy như sau:

~cs50/pset2/devigenere k

Ở con nháy của các bạn, k là một từ khoá nào đó. Các bạn sẽ muốn dán kết quả xuất ra từ chương trình của mình vào chỗ dữ liệu nhập vào trong chương trình của chúng tôi; nhớ kỹ, dùng cùng một từ khoá nhé. Lưu ý là các bạn không cần phải tự viết chương trình "devinegere", chỉ cần viết "vinegere" là được...

Nếu bạn muốn kiểm tra tính chính xác của chương trình bằng check50, bạn có thể tiến hành như sau.

check50 2014.fall.pset2.vigenere vigenere.c

Và nếu các bạn muốn vọc chương trình "vinegere" do chúng tôi viết trong ứng dụng này, thì các bạn có thể chạy dòng dưới đây:

~cs50/pset2/vigenere

Cố lên nhé!