استاد: **رامتین خسروی**
طراح: **فرزاد حبیبی**
مهلت تحویل: دوشنبه ۲۸ بهمن ۱۳۹۸، ساعت ۲۳:۵۵

# ۱ سربسته

هدف از این تمرین آشنایی شما با ورودی/خروجی استاندارد [۱] ، کار با فایل و همین‌طور بردار [۲] ها در زبان C++ می‌باشد. شما در این تمرین به پیاده‌سازی یک رمزکننده‌ی ساده‌ی جریانی می‌پردازید. رمزنگاری جریانی [۳] جهت رمز کردن یک جریان از کاراکترها در این پروژه مورد استفاده قرار می‌گیرد.

داستان از این قرار است که بابک می‌خواهد چند فایل را برای انیس ارسال کند. اما متاسفانه به دلیل قطعی اینترنت مجبور است این فایل‌ها را توسط پیک‌موتوری برای انیس ارسال کند. از طرفی دوست ندارد که پیک موتوری به هیچ‌عنوان محتویات فایل‌ها را متوجه شود. شما که در حال گذراندن درس برنامه‌سازی پیشرفته هستید به بابک پیشنهاد همکاری در قبال یک مبلغ هنگفت را می‌دهید. بابک نیز این پیشنهاد را قبول می‌کند.

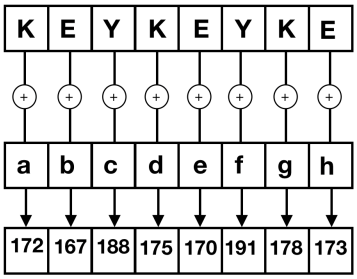
## ۱.۱ رمزنگاری

در رمزنگاری فایل‌های ورودی این برنامه از دو روش مختلف استفاده می‌شود. اجازه دهید روش اول را **ساده** و روش دوم را **پیچیده** نام‌گذاری کنیم. در هر دوی این روش از یک کلید با چند کاراکتر برای رمزنگاری استفاده می‌شود. تعداد کاراکترهای این کلید همواره از تعداد کاراکترهای فایلی که جهت رمزنگاری استفاده می‌شود کمتر است.

### ۱.۱.۱ ساده

در این روش کاراکترهای کلید به تعداد کاراکترهای پیام تکرار می‌شوند و در انتها، فایل رمز شده از جمع دوبه‌دوی این کاراکترها با یکدیگر ایجاد می‌شود.

به عنوان مثال فرض کنید کلیدی که بابک قصد رمز کردن فایل با آن را دارد **KEY** باشد. همین‌طور محتویات فایلی که قصد رمزنگاری آن را دارد **abcdefgh** باشد. در این حالت کلید که ۳ کاراکتر دارد به اندازه‌ی محتویات فایل تکرار می‌شود تا هر کدام از کاراکترهای فایل بتوانند با یک کاراکتر دیگر رمز شوند. در نهایت کد ASCII هر کارکتر از فایل با کد ASCII کاراکتر کلید جمع می‌شود و یک عدد به عنوان مقدار رمز شده تولید می‌شود. شکل ۱ شیوه‌ی رمزنگاری این مثال را نمایش می‌دهد.




شکل ۱: مثال رمزنگاری ساده

برای آشنایی بیشتر با کد ASCII می‌توانید به این لینک مراجعه کنید.

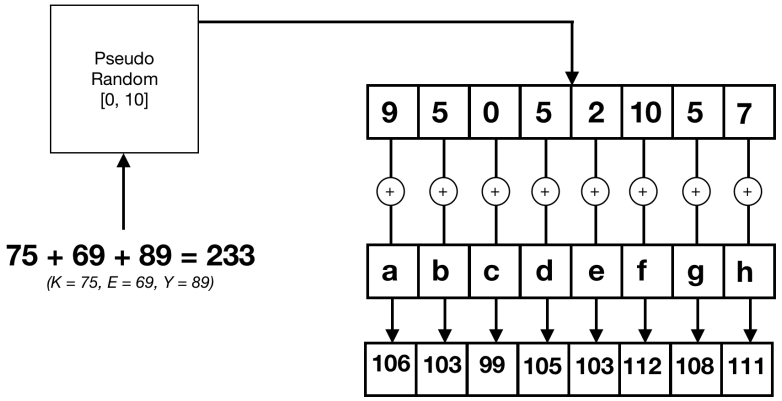
[1] standard I/O
[2] vector
[3] Stream Cipher

### ۲.۱.۱ پیچیده

در این روش از یک تابع شبه‌تصادفی [۴] جهت رمزنگاری استفاده می‌شود. توابع شبه‌تصادفی با گرفتن یک کلید [۵] همواره یک توالی یکسان از اعداد تصادفی ایجاد می‌شود. بنابراین انتظار می‌رود که خروجی این تابع با یک کلید یکسان همواره طی فراخوانی‌های متوالی یکسان باشد.

شما برای رمزنگاری از یک تابع تصادفی که کلید آن از جمع کدهای ASCII رشتهٔ کلید اصلی به دست می‌آید استفاده می‌کنید. شما به تعداد کاراکترهای فایل ورودی این تابع را صدا می‌زنید و هر بار خروجی آن که یک عدد شبه‌تصادفی بین ۰ تا ۱۰ ( خود ۰ و ۱۰ در این اعداد هستند) است را با کاراکتر متناظر جمع می‌کنید و کاراکتر رمز شده را در قالب یک عدد تولید می‌کنید.

برای نمونه، مثال بخش قبل را در نظر بگیرید. کلید تابع شبه‌تصادفی از جمع کدهای ACII سه کاراکتر **K** ، **E** و **Y** به‌دست می‌آید. در صورت ۸ بار فراخوانی این تابع، ۸ عدد تولید می‌شود که با جمع این ۸ عدد با کد ASCII هر کدام از کاراکترهای متناظر در فایل،اعداد رمز متناظر ایجاد می‌شوند.




شکل ۲: مثال رمزنگاری پیچیده

در زبان C++ برای تنظیم کردن کلید تابع شبه‌تصادفی از srand و برای هر بار فراخوانی آن و تولید یک عدد تصادفی جدید از تابع rand استفاده می‌شود. برای مثال قطعه کد زیر ۱۰ عدد تصادفی با کلید **KEY** تولید کرده و هر کدام از آن‌ها را چاپ می‌کند. خروجی این برنامه هر چند بار اجرا شود، یکسان است.

```
#include <iostream>
using namespace std;
int main()
{
   srand(int('K') + int('E') + int('Y'));
   for(int i = 0 ; i < 10 ; i++)
      cout << rand() % 11 << endl;
}
```

برای آشنایی بیشتر با این توابع می‌توانید به این لینک مراجعه کنید.

[4] pseudo random

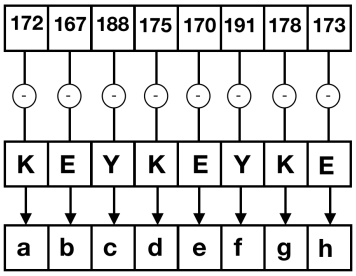
[۵] در حقیقت توابع شبه‌تصادفی یک ریشه یا seed دریافت می‌کنند و با آن توالی اعداد تصادفی را ایجاد می‌کنند. در این پروژه برای راحتی کار آن را کلید می‌نامیم.

## ۲.۱ رمزگشایی

منطقی است که وقتی انیس فایل‌ها را از پیک‌موتوری گرفت بتواند آن‌ها را رمزگشایی کند! پس شما باید به برنامه رمزگشایی فایل را برای هر کدام از روش‌های رمزنگاری اضافه کنید.

### ۱.۲.۱ ساده

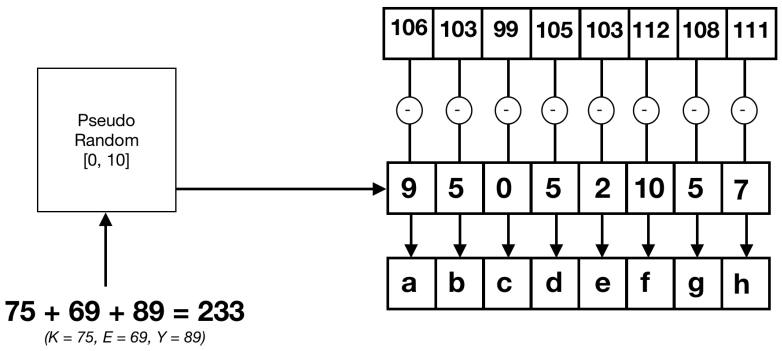
در این حالت از رمزگشایی کافیست تا کلید به تعداد اعداد فایل رمز شده تکرار شود و در نهایت مقدار اعدادی که در اثر رمزکردن کاراکترهای فایل اولیه به دست آمده‌اند، از مقدار کد ASCII کاراکتر کلید متناظر با آن کم شود. به عنوان نمونه رمزگشایی مثال بخش ۱.۱.۱ به صورت زیر انجام می‌شود.




شکل ۳: مثال رمزگشایی ساده

### ۲.۲.۱ پیچیده

در حالت پیچیده نیز با استفاده از تابع شبه‌تصادفی اعداد به ترتیب ایجاد می‌شوند و با کم کردن اعدادی که بر اثر رمز کردن به‌دست آمده‌اند از هر کدام از اعداد متناظر تصادفی، رشته‌ی اصلی ایجاد می‌شود. مثال بخش ۲.۱.۱ به شکل زیر رمزگشایی می‌شود.




شکل ۴: مثال رمزگشایی پیچیده

### ۳.۱ ورودی برنامه

برنامه به صورت کلی ۵ ورودی دارد. ورودی اول رمزنگاری یا رمزگشایی فایل را نشان می‌دهد. ورودی بعدی نوع آن را نشان می‌دهد. ورودی سوم کلید رمزگشایی یا رمزنگاری است. ورودی چهارم آدرس فایل ورودی جهت رمزنگاری یا رمزگشایی را نشان می‌دهد. و در نهایت ورودی آخر آدرس فایل خروجی بعد از یکی از عملیات‌های رمزنگاری یا رمزگشایی را نشان می‌دهد.

هر کدام از این ورودی‌ها در یک خط از ورودی استاندارد وارد می‌شوند. فرمت کلی ورودی در این برنامه به شکل زیر تعریف می‌شود.

```
$ g++ -std=c++11 sarbaste.cpp
$ ./a.out
<encrypt/decrypt>
<simple/complicated>
<key>
<input file path>
<output file path>
```

به عنوان مثال فرض کنید که برنامه بعد از کامپایل a.out نام داشته باشد. در صورتی که بخواهید یک فایل به اسم input.txt را با کلید PASSWORD با الگوریتم رمزنگاری ساده رمز کنید و در فایل output.txt بریزید، باید دستورها را به شکل زیر وارد کنید :

```
$ ./a.out
encrypt
simple
PASSWORD
./input.txt
./output.txt
```

### ۴.۱ خروجی برنامه

در صورتی که بخواهیم یک فایل را رمز کنیم، باید اعداد ایجاد شده از رمز کردن هر کدام از کاراکترهای فایل را در خط‌های جداگانه در فایل خروجی بنویسیم. همین‌طور اگر بخواهیم این اعداد که در یک فایل نوشته شده‌اند را رمزگشایی کنیم، در فایل خروجی باید رمزگشایی شده‌ی آن اعداد را به همان‌صورتی که قبل از رمزنگاری وجود داشته بنویسیم.

به عنوان مثال به فایل‌های زیر که متناظر با یکدیگر هستند دقت کنید.

| فایل رمزشده | فایل خام |
|---|---|
| ۱ 172<br>۲ 167<br>۳ 188<br>۴ 175<br>۵ 170<br>۶ 191<br>۷ 178<br>۸ 173<br>۹ | ۱ abcdefgh |

توجه داشته باشید که در انتهای فایل رمزشده حتماً یک خط خالی وجود دارد.

### ۵.۱ نکات پایانی

- در صورتی که یک فایل به آدرس فایل خروجی در ورودی برنامه وجود داشت، فایل باید بازنویسی [۶] شود.
- فرض می‌شود که همواره فایل ورودی وجود دارد و نیاز به رسیدگی خطاهای احتمالی وجود ندارد. همین‌طور فرض می‌شود ورودی ناخواسته به برنامه داده نمی‌شود و ورودی جز آن چه که در بخش ۳.۱ گفته شد به برنامه داده نمی‌شود.
- برای آشنایی با خواندن از فایل و نوشتن در آن می‌توانید به این لینک مراجعه کنید.
- می‌توانید به این فکر کنید که چرا روش پیچیده از روش ساده امن‌تر است و رمزنگاری روش ساده را به چه صورت می‌توان شکست!

## ۲ نحوهٔ تحویل

برنامهٔ خود را با نام A1-SID.cpp در صفحهٔ CECM درس بارگذاری کنید که SID شمارهٔ دانشجویی شماست؛ برای مثال اگر شمارهٔ دانشجویی شما ۸۱۰۱۹۸۹۹۹ باشد، نام پروندهٔ شما باید A1-810198999.cpp باشد.

- برنامهٔ شما باید در سیستم‌عامل لینوکس و با مترجم g++ با استاندارد c++11 ترجمه و در زمان معقول برای ورودی‌های آزمون اجرا شود.
- از صحت قالب[۷] ورودی‌ها و خروجی‌های برنامهٔ خود مطمئن شوید. برنامهٔ شما در هنگام تحویل حضوری به صورت اتوماتیک تست می‌شود؛ لذا، از دادن خروجی‌هایی که در صورت پروژه گفته نشده‌است اجتناب کنید.
- رعایت سبک برنامه‌نویسی درست و تمیز بودن برنامه‌ی شما در نمرهٔ تمرین تأثیر زیادی دارد.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت کشف تقلب مطابق قوانین درس با آن برخورد خواهد شد.

## آ مقایسهٔ خروجی برنامه با خروجی مورد انتظار

مقایسهٔ خروجی برنامه با خروجی مورد انتظار با چشم شاید برای برنامه‌های کوچک که خروجی کمی تولید می‌کنند و روند اجرای کوتاهی دارند میسر باشد، برای برنامه‌های بزرگ‌تر با مسیر اجرای پیچیده کاری دشوار است. برای این کار می‌توان از ابزارهایی که در سیستم‌عامل لینوکس در دسترس است استفاده کرد.

در حالت عادی، برای ترجمه و اجرای یک برنامه از این دستورها استفاده می‌شود:

```
g++ -std=c++11 helloworld.cpp
./a.out
```

در این حالت برنامه ورودی‌اش را از ورودی استاندارد stdin (خط فرمان) می‌خواند و خروجی را نیز در خروجی استاندارد stdout (صفحه‌ی خط فرمان) می‌نویسد.

برای اجرای راحت‌تر برنامه، می‌توان ورودی را در پرونده مانند in.txt نوشت و سپس محتوای آن را به ورودی استاندارد تغییر مسیر[۸] داد تا هنگام اجرای مکرر برنامه نیازی به نوشتن مکرر ورودی‌های مختلف در خط فرمان نباشد:

[6] overwrite
[7] format
[8] redirect

```
./a.out < in.txt
```

همچنین، می‌توان خروجی برنامه را به پرونده‌ای مانند out.txt تغییر مسیر داد تا بتوان بعداً هم به آن دسترسی داشت:

```
./a.out > out.txt
```

ترکیب این دو عمل نیز امکان‌پذیر است:

```
./a.out < in.txt > out.txt
```

فرض کنیم خروجی مورد انتظار برای ورودی in.txt در پرونده‌ای به نام sol.txt قرار دارد. می‌توان با استفاده از دستور `diff` خروجی حاصل از اجرای برنامه را با خروجی مورد انتظار مقایسه کرد.

برای این کار، ابتدا ورودی را از in.txt به برنامه می‌دهیم و خروجی برنامه را در پرونده‌ای مانند out.txt ذخیره می‌کنیم. سپس با دستور `diff` پرونده‌ی out.txt را با sol.txt مقایسه می‌کنیم.

```
g++ -std=c++11 helloworld.cpp
./a.out < in.txt > out.txt
diff out.txt sol.txt
```

اگر پرونده‌ها یکسان باشند، دستور `diff` هیچ خروجی‌ای تولید نمی‌کند. وگرنه، تفاوت‌های دو پرونده را نشان می‌دهد.

هر بخش از خروجی این دستور با شماره‌ی خطوط آغاز می‌شود: شماره‌ی خطوط در پرونده‌ی قدیمی (سمت چپ)، یکی از حروف `d`، `a` یا `c` و شماره‌ی خطوط در پرونده‌ی جدید (سمت راست). حرف میان شماره‌ی خطوط نوع تغییرات را نشان می‌دهد:

- **d: حذف شدن** محتوای محذوف بعد از < نمایش داده می‌شود.
- **a: افزوده شدن** محتوای جدید بعد از > نمایش داده می‌شود.
- **c: تغییر** محتوای قدیمی بعد از < نمایش داده می‌شود. سپس خطی شامل `---` می‌آید. بعد از آن، محتوای جدید بعد از > نمایش داده می‌شود.

به این مثال[9] توجه کنید:

[9] https://en.wikipedia.org/wiki/Diff

قديمي

```
This part of the
document has stayed the
same from version to
version.  It shouldn't
be shown if it doesn't
change.  Otherwise, that
would not be helping to
compress the size of the
changes.

This paragraph contains
text that is outdated.
It will be deleted in the
near future.

It is important to spell
check this dokument. On
the other hand, a
misspelled word isn't
the end of the world.
Nothing in the rest of
this paragraph needs to
be changed. Things can
be added after it.
```

جديد

```
This is an important
notice! It should
therefore be located at
the beginning of this
document!

This part of the
document has stayed the
same from version to
version.  It shouldn't
be shown if it doesn't
change.  Otherwise, that
would not be helping to
compress the size of the
changes.

It is important to spell
check this document. On
the other hand, a
misspelled word isn't
the end of the world.
Nothing in the rest of
this paragraph needs to
be changed. Things can
be added after it.

This paragraph contains
important new additions
to this document.
```

```
0a1,6
> This is an important
> notice! It should
> therefore be located at
> the beginning of this
> document!
>
11,15d16
< This paragraph contains
< text that is outdated.
< It will be deleted in the
< near future.
<
17c18
< check this dokument. On
---
> check this document. On
24a26,29
>
> This paragraph contains
> important new additions
> to this document.
```