استاد: **امیرمسعود ربیعی**

طراح: **هدی برخوردارپور**، **علی رنجبر**

## ۱ مقدمه

در این تمرین کامپیوتری قصد داریم با نرم‌افزار متلب[۱] و متمتیکا[۲] آشنا شویم.

### ۱.۱ متلب

متلب، یک محیط نرم‌افزاری برای انجام محاسبات عددی و یک زبان برنامه‌نویسی نسل چهارم است که از ترکیب دو واژهٔ MATrix (ماتریس) و LABoratory (آزمایشگاه) ایجاد شده‌است. این نام حاکی از رویکرد ماتریس محور برنامه است، که در آن حتی اعداد منفرد هم به عنوان ماتریس در نظر گرفته می‌شوند.

### ۲.۱ متمتیکا

متمتیکا، یک نرم‌افزار جبری بسیار رایج، که توسط شرکت ولفرم ریسرچ پدید آورده شده است و اکثر توابع نرم‌افزاری موردنیاز در ریاضی و علوم طبیعی را در اختیار استفاده‌کنندگان آن قرار می‌دهد.

### ۳.۱ مقایسه‌ی متلب و متمتیکا

- جهت‌گیری متلب بیشتر برای کار با داده هاست (که در این بسیار خوب عمل می‌کند) اما با اینکه امکان محاسبات نمادین در متلب وجود دارد، این امکان در متمتیکا بسیار آسان‌تر و کارآمدتر است.
- متلب یک محیط برنامه‌نویسی در حوزه‌ی مهندسی است و چون محاسبات آن با استفاده از تقریب و تخمین‌های ریاضیست بنابراین در کارهای ریاضی کاربردی که اصل کار همان ساختن تقریب هاست ممکن است زیاد مناسب نباشد. متمتیکا یک نرم‌افزار ریاضی است که هم در ریاضیات وهم در مهندسی کاربرد دارد. محاسبات نمادین و محض مثل حدگیری و مسایل جبر را به راحتی انجام داده و تمام مراحل حل را به کاربر نشان می‌دهد.
- مصورسازی و رسم نمودار در هر دو نرم افزار به خوبی انجام می‌شود.
- ساختن رابط کاربری برای نرم‌افزار در متمتیکا بسیار آسان‌تر از متلب است.
- مهمترین انتقادات از متلب به خاطر متن بازنبودن و گران بودن آن است که امکان اجرای کدهای نوشته‌شده در متلب را در هر محیطی محدود می‌کند.

[۱] MATLAB
[۲] Mathematica

### ۴.۱ سیگنال‌ها در متلب

سیگنال‌های پیوسته_زمان (به اختصار پیوسته) متناظر با هر نقطه‌ای از محور زمان یک مقداری دارند در حالی که سیگنال‌های گسسته_زمان (به اختصار گسسته) فقط در مقادیر صحیح از محور زمانی مقدار دارند. $x[n]$ یک سیگنال گسسته را نشان می‌دهد که $n$ فقط می‌تواند مقادیر صحیح اختیار کند.

همان‌طور که می‌دانید ذخیره تمام مقادیر یک سیگنال پیوسته در طول یک بازه‌ی زمانی ناممکن است. پس چگونه سیگنال‌های پیوسته را پردازش کنیم؟ در آینده خواهید آموخت که چگونه یک سیگنال پیوسته را با نمونه‌برداری به سیگنال گسسته تبدیل می‌کنیم. (به کمک دستور syms می‌توان به شکل پیوسته کار کرد، که به هیچ وجه توصیه نمی‌شود و در صورت استفاده نمره‌ای تعلق نخواهد گرفت.)

## ۲ کانولوشن گسسته_زمان

هدف این تمرین نوشتن مفسر برای یک زبان برنامه‌نویسی بسیار ساده به نام **زبانچه** است. مثال زیر یک نمونه‌ای کوچک از برنامه‌های زبانچه است.

در زبانچه سه نوع دستور وجود دارد:

- **ورودی** دستور ورودی به شکل variable ? است که در آن variable نام یک متغیر است. اجرای این دستور موجب می‌شود که مفسر منتظر گرفتن یک عدد صحیح از ورودی بماند تا کاربر عدد را وارد کند.
- **خروجی** دستور خروجی به شکل variable ! است که در آن variable نام یک متغیر است. اجرای این دستور موجب می‌شود که مفسر مقدار متغیر مربوطه را در یک خط مجزا در خروجی بنویسد.
- **جایگزینی** شکل این دستور variable = expression است که با اجرای آن عبارت expression ارزشیابی شده، مقدار آن در متغیر variable گذاشته می‌شود.

عبارت‌ها در زبانچه از ترکیب تعدادی متغیر و عدد ثابت با عملگرهای جمع و تفریق به دست می‌آیند. تمام متغیرها و ثابت‌ها مقدار عدد صحیح دارند. نیازی به تعریف متغیرها نیست و اگر متغیری پیش از این که مقدار بگیرد استفاده شود، به طور پیش‌فرض حاوی مقدار صفر فرض می‌شود. به این ترتیب، اگر به ورودی برنامه‌ی فوق به ترتیب مقادیر ۳ و ۸ داده شود، خروجی برنامه دو عدد ۱۳ و ۱۰ خواهد بود که در دو خط جداگانه نوشته می‌شوند. دقت کنید که مقدار متغیر n در خط قبل از آخر برابر صفر در نظر گرفته می‌شود.

نام متغیرها در یک برنامه‌ی زبانچه ترکیبی از حروف بزرگ و کوچک و ارقام است و حتماً با یک حرف شروع می‌شود. حروف بزرگ و کوچک متمایز فرض می‌شوند. فرض کنید اعداد ثابت نامنفی هستند و در یک نوع داده‌ی[۳] ۳۲بیتی جا می‌شوند. در دو طرف اعداد و اسامی متغیرها و عملگرها می‌تواند تعداد دلخواهی (صفر یا بیشتر) فاصله‌ی خالی باشد. هر دستور در یک خط نوشته می‌شود و خط‌های خالی در برنامه نادیده گرفته می‌شوند.

برنامه‌ی شما یک برنامه‌ی زبانچه را به همراه یک ورودی برای آن دریافت می‌کند و خروجی برنامه‌ی زبانچه را به ازای آن ورودی مشخص می‌کند.

برای نگهداری مقادیر متغیرها می‌توانید از کلاس map که در سرآیند[۴] map در دسترس است استفاده کنید. برای خواندن یک خط کامل از ورودی نیز می‌توانید از تابع getline در سرآیند iostream کمک بگیرید.

[۳] data type
[۴] header

### ۱.۲ ورودی

ورودی برنامه دو بخش دارد: برنامه‌ی زبانچه و ورودی برنامه‌ی زبانچه. بخش اول برنامه‌ی زبانچه است که در انتهای آن خطی وجود دارد که حاوی رشته‌ی run است. بعد از این خط ورودی برنامه‌ی زبانچه می‌آید. این ورودی شامل تعدادی خط است که هر کدام حاوی یک عدد صحیح نامنفی است. این اعداد به دستورهای ورودی برنامه‌ی زبانچه داده می‌شوند.

پایان ورودی با نویسه‌ی[۵] پایان فایل (EOF) مشخص می‌شود. در خط فرمان لینوکس می‌توانید با ترکیب Ctrl + D این نویسه را ارسال کنید.

### ۲.۲ خروجی

خروجی برنامه‌ی زبانچه را در خروجی بنویسید.

اگر برنامه‌ی زبانچه دارای خطای نحوی باشد **بدون اجرای برنامه** رشته‌ی `Syntax error at line n` را بنویسید که در آن n شماره‌ی اولین خطی از برنامه‌ی زبانچه است که خطا دارد.

اگر ورودی برنامه‌ی زبانچه کم‌تر از مواردی که مورد انتظار برنامه است ورودی مشخص کرده باشد، مثلاً در برنامه چهار دستور ورودی است اما ورودی فقط سه عدد را مشخص کرده است، برنامه‌ی زبانچه را تا جایی که امکان دارد اجرا کنید و خروجی‌ها را بنویسید و در انتها رشته‌ی `Unexpected end of input` را در خروجی بنویسید.

### ۳.۲ ورودی و خروجی نمونه

| ورودی نمونه | خروجی نمونه |
|---|---|
| | |
| | |
| | |

## ۳ نحوه‌ی تحویل

برنامه‌ی خود را با نام A1-SID.cpp در صفحه‌ی CECM درس بارگذاری کنید که SID شماره‌ی دانشجویی شماست؛ برای مثال اگر شماره‌ی دانشجویی شما ۸۱۰۱۹۷۹۹۹ باشد، نام پرونده‌ی شما باید A1-810197999.cpp باشد.

[۵] character

- برنامه‌ی 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[۶] ورودی‌ها و خروجی‌های برنامه‌ی خود مطمئن شوید.
- رعایت سبک برنامه‌نویسی درست و تمیز بودن برنامه‌ی شما در نمره‌ی 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 مقایسه‌ی خروجی برنامه با خروجی مورد انتظار

مقایسه‌ی 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[۷] داد تا هنگام اجرای مکرر برنامه نیازی به نوشتن مکرر ورودی‌های مختلف در خط فرمان نباشد:

```
./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
```

[۶] format
[۷] redirect

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[۸] توجه کنید:

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