



۱ مفسر زبانچه

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

```
۱ ? num1
۲ ? num2
۳ diff = num2 - num1
۴ z = num2 + diff
۵ ! z
۶ k = diff + z - 8 + n
۷ ! k
```

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^۱ ۳۲ بیتی 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^۲ `map` در دسترس است استفاده کنید. برای خواندن یک خط کامل از ورودی نیز می‌توانید از تابع `getline` در سرآیند `iostream` کمک بگیرید.

¹data type

²header

۱.۱ 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^۳ پایان فایل (EOF) مشخص می‌شود. در خط فرمان لینوکس می‌توانید با ترکیب Ctrl + D این نویسه را ارسال کنید.

۲.۱ 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.

۳.۱ ورودی و خروجی نمونه

ورودی نمونه	خروجی نمونه
<pre> ۱ ? num1 ۲ ? num2 ۳ diff = num2 - num1 ۴ z = num2 + diff ۵ ! z ۶ k = diff + z - 8 + n ۷ ! k ۸ run ۹ 3 ۱۰ 8 </pre>	<pre> 13 10 </pre>
<pre> ۱ ? a ۲ !a ۳ ? b ۴ r=a+2 ۵ ! r ۶ run ۷ 5 </pre>	<pre> 5 Unexpected end of input </pre>
<pre> ۱ ? var ۲ ! var ۳ ۴ var = -2 + var ۵ ! var ۶ run ۷ 3 </pre>	<pre> Syntax error at line 4 </pre>

^۳character

۲ 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.

- برنامه‌ی شما باید در سیستم عامل لینوکس و با مترجم g++ با استاندارد c++11 ترجمه و در زمان معقول برای ورودی‌های آزمون اجرا شود.
- از صحت قالب^۴ 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 سیستم عامل 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^۵ 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
```

^۴format
^۵redirect

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 a، d 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 مثال⁶ توجه کنید:

قدیمی	جدید	
1 This part of the	1 This is an important	0a1,6
2 document has stayed the	2 notice! It should	> This is an important
3 same from version to	3 therefore be located at	> notice! It should
4 version. It shouldn't	4 the beginning of this	> therefore be located at
5 be shown if it doesn't	5 document!	> the beginning of this
6 change. Otherwise, that	6	> document!
7 would not be helping to	7 This part of the	>
8 compress the size of the	8 document has stayed the	11,15d16
9 changes.	9 same from version to	< This paragraph contains
10	10 version. It shouldn't	< text that is outdated.
11 This paragraph contains	11 be shown if it doesn't	< It will be deleted in the
12 text that is outdated.	12 change. Otherwise, that	< near future.
13 It will be deleted in the	13 would not be helping to	<
14 near future.	14 compress the size of the	17c18
15	15 changes.	< check this dokument. On
16 It is important to spell	16	---
17 check this dokument. On	17 It is important to spell	> check this document. On
18 the other hand, a	18 check this document. On	24a26,29
19 misspelled word isn't	19 the other hand, a	>
20 the end of the world.	20 misspelled word isn't	> This paragraph contains
21 Nothing in the rest of	21 the end of the world.	> important new additions
22 this paragraph needs to	22 Nothing in the rest of	> to this document.
23 be changed. Things can	23 this paragraph needs to	
24 be added after it.	24 be changed. Things can	
	25 be added after it.	
	26	
	27 This paragraph contains	
	28 important new additions	
	29 to this document.	

⁶<https://en.wikipedia.org/wiki/Diff>