برنامه‌سازی پیشرفته

تمرین کامپیوتری شماره ۶

استاد: **دکتر رامتین خسروی**

طراحان: **سعید زنگنه**، **حسنا سادات آزرم‌سا**، **غزل کلهر**

مهلت تحویل: دوشنبه ۱۸ آذر ۱۳۹۸، ساعت ۲۳:۵۵

مقدمه

هدف از این تمرین آشنایی شما با مفاهیم اولیه‌ی وراثت[1] و چندریختی[2] است. انتظار می‌رود تکنیک‌های برنامه‌نویسی‌ای را که در کلاس درس یا در هنگام تحویل تمرین‌های قبلی فراگرفته‌اید، به طور کامل در این تمرین به‌کار گیرید. این تمرین از سه بخش تشکیل شده است که بخش سوم آن امتیازی است. طراحی کلاس‌ها، نحوه‌ی ارث‌بری آن‌ها از یکدیگر و تعریف صحیح توابع مربوط به هر کدام از کلاس‌ها اهمیت بالایی دارد؛ به همین منظور پیشنهاد می‌شود قبل از پیاده‌سازی پروژه، ابتدا طراحی‌های مختلف را بررسی کرده و سپس مناسب‌ترین طراحی را پیاده‌سازی کنید.

مدیریت استثناها[3]

در طول اجرای برنامه، ممکن است انواع خطا در برنامه شما رخ دهد. در این تمرین، انتظار می‌رود شما با پرتاب[4]کردن نمونه[5]هایی از کلاس‌هایی که ادامه ذکر خواهد شد و گرفتن[6] آن‌ها به خطاها رسیدگی کنید. این کلاس‌ها از کلاس `std::exception` ارث برده‌اند و شما باید تابع `what` این کلاس‌ها را Override کنید و در هنگام گرفتن این استثناء[7]ها، خروجی تابع `what` را در جریان خطای استاندارد[8] چاپ کنید.

خطاهایی که انتظار می‌رود که شما در برنامه‌تان به آن‌ها رسیدگی کرده باشید، در شرح تمرین آمده‌اند.

۱. ارزش‌یابی درخت عبارت

درخت عبارت از تعدادی رأس عملگر و عملوند تشکیل شده است. برگ‌های درخت عملوندهایی در مبنای ۱۰ هستند و سایر گره‌های درونی درخت عملگرها خواهند بود. در این مسئله چهار عملگر جمع، ضرب،منفی و میانه را پیاده سازی خواهید نمود. از بین این چهار عملگر جمع و ضرب را دومتغیره، منفی را تک متغیره و میانه را چند متغیره فرض کنید و پس از ایجاد درخت عبارت، اطمینان حاصل کنید که این محدودیت‌های مطرح شده رعایت شده باشد. مثلا برای یک عملگر دو متغیره دقیقا دو فرزند قرار داده شده باشد. در صورت وجود خطا در درخت عبارت به خطای احتمالی با استفاده از Exception Handling رسیدگی کنید و به برنامه خاتمه دهید.

---

[1] Inheritance
[2] Polymorphism
[3] Exception Handling
[4] Throw
[5] Instance
[6] Catch
[7] Exception
[8] Standard Error Stream

Add(x,y)	Mult(x,y)	Not(x)	Med(x,y,z,...)
x+y	x*y	-x	میانه‌ی مجموعه‌ی ورودی

برای محاسبه‌ی حاصل یک درخت عبارت از برگ‌های حرکت می‌کنیم و عملگرها را در هر سطح اعمال می‌کنیم تا به ریشه برسیم. مقدار مورد نظر برابر حاصل عملگر ریشه خواهد بود.

جزییات توابعی که انتظار می‌رود شما در این تمرین پیاده‌سازی کنید به شرح زیر است:

معرفی یک راس عملوند:

```
void add_operand_node(const std::size_t id, const std::size_t parent_id, const int value);
```

ورودی‌های این تابع ابتدا یک شناسه‌ی یکتا برای آن رأس است. در صورت فراخوانی این تابع با مقدار id تکراری باید یک استثنا با توضیح "Duplicate node ID." پرتاب کنید. فرض کنید همان ابتدا یک رأس با شناسه‌ی صفر(ریشه) حاضر است و نیازی به افزودن آن نیست. رأس ریشه صرفا برای راحتی پیاده‌سازی در نظر گرفته شده و از نوع خاصی نیست.

***تذکر: استفاده از ساختارهای کنترلی برای بررسی این شروط مجاز نیست.**

خطاها:

۱) اگر شناسه‌ی پدر در درخت موجود نبود باید یک استثنای با توضیح "Parent node not found." پرتاب شود.

۲) اگر یک رأس به عنوان فرزند یک رأس از نوع عملوند معرفی شد، باید یک استثنا با توضیح "Invalid parent." پرتاب شود.

۳)اگر دو رأس مختلف به عنوان فرزند ریشه معرفی شدند، باید یک استثنا با توضیح "Invalid parent." پرتاب شود.

معرفی یک رأس عملگر:

```
void add_operator_node(const std::size_t id, const std::size_t parent_id, const OperatorType operator);
```

این تابع در ورودی اول همانند تابع قبلی یک شناسه‌ی یکتا می‌گیرد که هم در مجموعه‌ی عملوندها و هم در مجموعه‌ی عملگرها یکتاست. ورودی دوم نیز مانند تابع قبلی بیانگر شناسه‌ی رأس پدر است. ورودی سوم یک شمارشگر از نوع OperatorType است که در کد داده‌شده موجود است. تمام خطاهای تابع قبلی در این تابع نیز باید بررسی شوند.

محاسبه‌ی درخت:

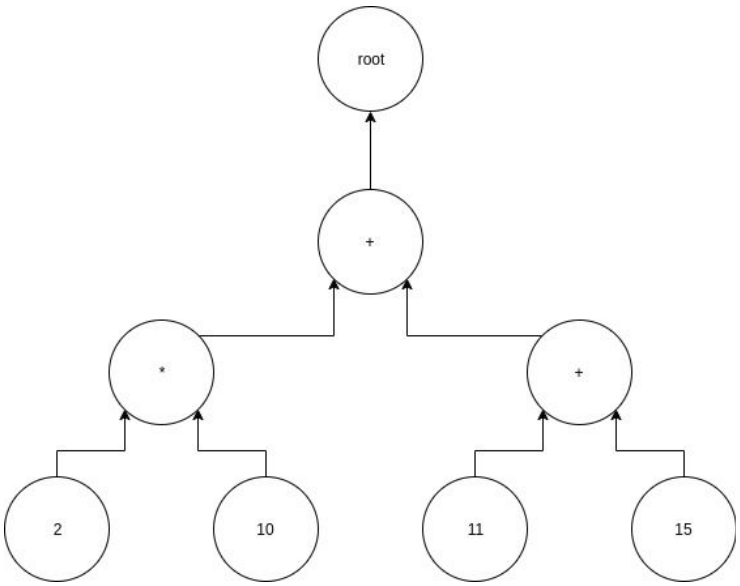
```
int evaluate_tree();
```

این تابع مقدار خروجی درخت را محاسبه می‌کند و به عنوان خروجی می‌دهد. می‌توانید از نداشتن دور در درخت ورودی اطمینان داشته باشید.

خطاها:

۱) اگر در محاسبه‌ی مقادیر متناظر رأس‌های عملگر یک درخت تعداد عملوندها با تعداد عملوندهای موردنیاز آن عملگر متفاوت بود، باید یک استثنا با توضیح "Invalid tree structure." پرتاب شود.

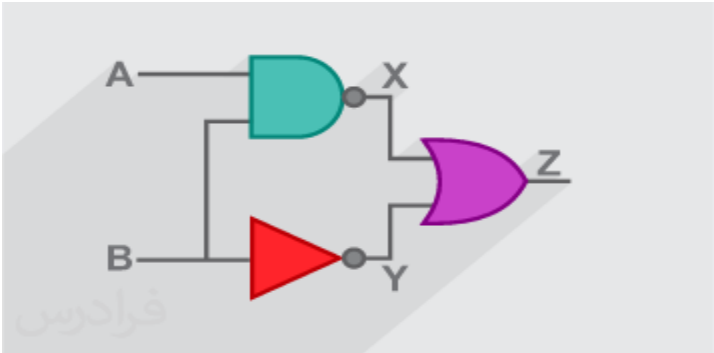
مثال از محاسبه‌ی درخت:



در شکل بالا نتیجه‌ی محاسبه‌ی درخت عدد ۴۶ خواهد بود.

۲. محاسبه‌ی خروجی شبکه‌ی منطقی

مدارهای منطقی جزء جداناپذیری از وسایل الکترونیکی هستند و در هر یک از این وسایل مدارهای منطقی زیادی به چشم می‌خورد. هر مدار منطقی از تعدادی گیت و سیم تشکیل شده است. هر گیت یک خروجی و تعدادی ورودی دارد(هر ورودی یک بیت ۰ یا ۱ می‌باشد) که توسط سیم‌ها می‌توانیم خروجی یا ورودی گیتی را به ورودی گیتی دیگر وصل کنیم. یک نمونه از مدارهای منطقی در شکل زیر قابل مشاهده است.



انواع گیت‌های مورد استفاده در پروژه و عملیات منطقی هرکدام در جدول زیر توضیح داده شده‌است.

Gate	Function
and	زمانی که تمام ورودی‌ها ۱ باشند خروجی ۱ و در غیر این‌صورت خروجی ۰ خواهد بود.

or	زمانی که حداقل یکی از ورودی‌ها ۱ باشد خروجی ۱ و در غیر این‌صورت خروجی ۰ خواهد بود.
not	اگر ورودی ۱ باشد خروجی ۰ و اگر ورودی ۰ باشد خروجی ۱ خواهد بود.
xor	اگر تعداد یک‌های ورودی فرد باشد خروجی ۱ و در غیر این‌صورت خروجی ۰ خواهد بود.
nand	زمانی که حداقل یکی از ورودی‌ها ۰ باشد خروجی ۱ و در غیر این‌صورت خروجی ۰ خواهد بود.
nor	زمانی که تمام ورودی‌ها ۰ باشند خروجی ۱ و در غیر این‌صورت خروجی ۱ خواهد بود.

شرح تمرین

در این تمرین شما یک مدار منطقی را شبیه‌سازی می‌کنید. جزئیات توابعی که انتظار می‌رود پیاده‌سازی کنید در ادامه آمده است. خطاهایی که انتظار می‌رود به آن رسیدگی کنید در ادامه آمده است. در صورتی که به هرکدام از خطاها برخورد کردید به روش مناسب پیام مشخص‌شده را در جریان خطای استاندارد چاپ کرده و بدون اجرای تابعی که خطا در آن رخ داده‌است **به ادامه برنامه بپردازید.** در واقع از توابع فراخوانی‌شده نباید هیچ‌گونه استثنایی پرتاب شوند.

اضافه کردن گیت به مدار

ورودی‌ها و خروجی گیت‌ها هر کدام با شناسه‌ای یکتا مشخص می‌شوند. با فراخوانی تابع زیر یک گیت از انواع تعریف‌شده در جدول بالا ساخته می‌شود. شناسه یکتای ورودی‌ها و خروجی گیت نیز در این تابع مشخص داده می‌شود. برای پیاده‌سازی این تابع می‌توانید از variadic functions در c++ استفاده کنید. برای اطلاعات بیشتر درباره این نوع توابع می‌توانید به این لینک مراجعه کنید.
همچنین نشانه noexcept در رابط کاربری توابع به کامپایلر این تضمین را می‌دهد که تابع استثنائی پرتاب نمی‌کند.(استثناهای پرتاب شده باید در داخل این توابع گرفته و رسیدگی شوند) برای اطلاعات بیشتر درباره این نشانه می‌توانید به این لینک مراجعه کنید.

```
void add_gate_to_circuit(const GateType gate, const int number_of_gate_inputs , const int output_id, const int input1_id, …) noexcept;
```

خطاها

۱) اگر شناسه‌های داده‌شده قبلا استفاده شده‌بودند و یکتا نبود یک استثنا با پیام "Duplicate ID." پرتاب شود.
۲) اگر گیت از نوع Not بود و تعداد ورودی‌های گیت بیشتر از یک ورودی بود یک استثنا با پیام "Not should have 1 input." پرتاب شود.

اضافه کردن سیم به مدار

با فراخوانی تابع زیر خروجی یک گیت به ورودی گیت دیگر متصل میشود.(برای راحتی کار فرض کنید وصل شدن دو ورودی به هم تست نمی‌شود.) توجه شود لزوما پارامتر اول خروجی نیست.

```
void connect(const int first_id, const int second_id) noexcept;
```

خطاها

۱) خروجی دو گیت توسط سیم نمی‌توانند به هم متصل شوند در این صورت باید یک استثنا با پیام "Override by two driver." پرتاب شود.

قرار دادن و به‌روزرسانی یک مقدار بر روی ورودی گیت

با فراخوانی تابع زیر می‌توان بر روی ورودی یک گیت با شناسه مشخص یک بیت ۰ یا ۱ قرار داد. اگر یک مقدار در حال حاضر بر روی ورودی گیت وجود داشته باشد این مقدار به‌روزرسانی می‌شود.

```
void put_value_on_gate_input(const int input_id, const int value) noexcept;
```

خطاها

۱) یک مقدار فقط می‌تواند بر روی ورودی گیتی که قبلا با سیم به گیت دیگری متصل نشده‌است قرار گیرد در صورتی که شناسه این ویژگی را نداشت یک استثنا با پیام "Override by two driver." پرتاب شود.

۲) اگر مقدار داده شده جزو مقادیر معتبر نبود یک استثنا با پیام "Invalid value." پرتاب شود.

۳) اگر شناسه مشخص‌شده شناسه یک ورودی نبود یک استثنا با پیام "Invalid ID." پرتاب شود.

مشاهده خروجی یک گیت

با فراخوانی تابع زیر باید بتوان خروجی یک گیت با شناسه مشخص شده را مشاهده کرد.

```
void print_gate_output(const int output_id) noexcept;
```

خطاها

۱) اگر شناسه مشخص‌شده شناسه یک خروجی نبود یک استثنا با پیام "Invalid ID." پرتاب شود.

۲) اگر یکی از ورودی‌های لازم برای ارزیابی خروجی گیت مشخص‌شده مقدار نداشت یک استثنا با پیام "Bad initialization." پرتاب شود.

نحوه نمایش خروجی یک گیت

با فراخوانی تابع بالا خروجی بر اساس فرمت زیر در کنسول چاپ می‌شود:

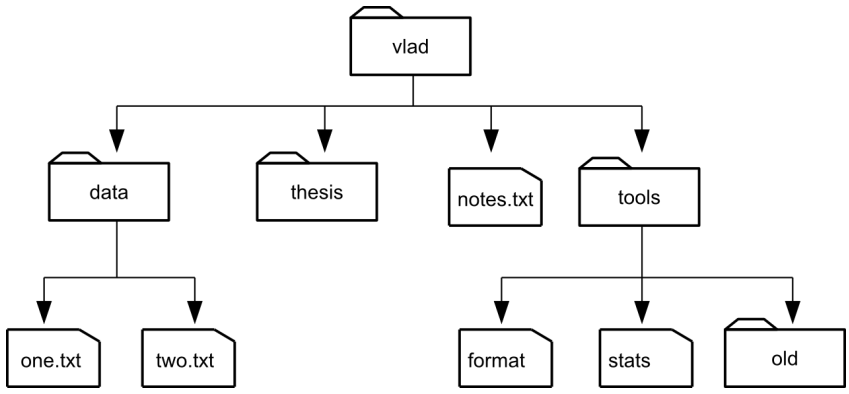
The current output of gate with output id “output_id” is “value”.

که به جای "output_id" شناسه خروجی گیت و به جای "value" مقدار خروجی (۰ یا ۱) قرار می‌گیرد.

۳. چاپ درخت فایل‌ها(امتیازی)

فایل سیستم

در علم کامپیوتر، فایل سیستم وظیفه‌ی کنترل کردن نحوه ذخیره‌سازی و بازیابی اطلاعات ذخیره‌شده در حافظه را برعهده دارد. بدون استفاده از فایل سیستم، اطلاعات بدون هیچ ساختار خاصی در حافظه ذخیره می‌شدند و امکان بازیابی اطلاعات ذخیره‌شده وجود نداشت. با اختصاص دادن نام و شناسه یکتا به هر مجموعه‌ی اطلاعات می‌توان اطلاعات را از هم تشخیص داده و آن‌ها را بازیابی کرد. هر فایل سیستم از تعدادی پرونده و پوشه تشکیل شده‌است. یک نمونه از فایل سیستم در شکل زیر قابل مشاهده است.



پوشه

پوشه ساختاری در فایل سیستم است که شامل ارجاعاتی به سایر پوشه‌ها و پرونده‌ها است. به پوشه‌ای که در پوشه‌ی دیگری قرار دارد اصطلاحا زیرپوشه می‌گویند. هر پوشه می‌تواند صفر، یک یا چند زیرپوشه و یا پرونده داشته باشد.

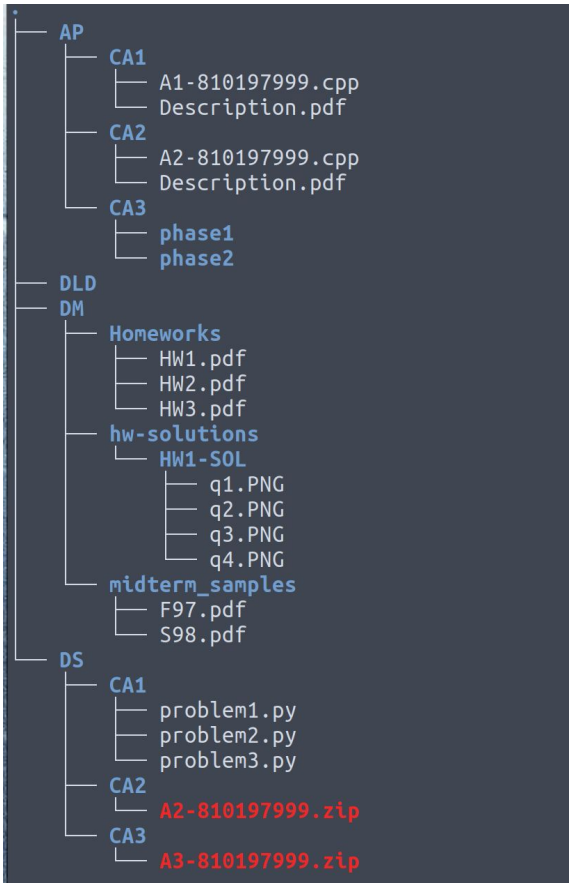
پرونده

پرونده یک منبع کامپیوتری است که برای ذخیره‌سازی اطلاعات به صورت مجزا در حافظه دستگاه استفاده می‌شود. برای ذخیره‌سازی انواع تصاویر، ویدئوها، متن‌ها و ... می‌توان از پرونده‌ها استفاده کرد. همچنین پرونده‌ها را می‌توان با استفاده از برنامه‌های کامپیوتری باز کرد، تغییر داد و بست.

قالب‌های نمایش فایل سیستم

فایل سیستم را می‌توان با قالب‌های گوناگونی مشاهده کرد. از جمله قالب‌های متداول قالب list، grid view می‌باشد. همچنین ترتیب پرونده‌ها و پوشه‌های یک فایل سیستم را می‌توان بر اساس پارامترهای گوناگونی معین کرد. برای مثال می‌توان تنظیمات یک فایل سیستم را طوری تغییر داد تا پرونده‌ها و پوشه‌ها به صورت مرتب‌شده براساس نام و با قالب لیست نشان داده‌شوند.

یکی دیگر از قالب‌های نمایش فایل سیستم، قالب درخت است. در این قالب محتویات موجود در دایرکتوری با ساختار سلسله مراتبی نمایش داده می شوند و این روند به طور بازگشتی روی پوشه های موجود در سیستم، پیاده می شود. یک نمونه از این نمایش در شکل زیر قابل مشاهده است.



شرح تمرین

در این تمرین شما در ابتدا با دستوراتی که جزئیات آن ها در ادامه آمده است، یک فایل سیستم ایجاد می کنید و در نهایت آن را در قالب درخت نمایش می دهید.

اضافه‌کردن پوشه

با فراخوانی تابع زیر، پوشه جدید با نام مشخص شده در پوشه فعلی ساخته می شود:

```
void add_folder(const std::string folder_name)
```

اضافه‌کردن پرونده

با فراخوانی تابع زیر، پرونده جدید با نام مشخص شده در پوشه فعلی ساخته می شود:

```
void add_file(const std::string file_name)
```

خطاها

۱) نام پوشه‌ها و پرونده‌های موجود در پوشه فعلی یکتاست. در صورت تداخل اسمی پرونده‌ها و پوشه‌های یک پرونده باید یک استثنا با پیغام "Duplicate file name." پرتاب شود. این خطا برای دستور اضافه‌کردن پوشه هم باید بررسی شود.

تغییر دادن پوشه فعلی

با فراخوانی تابع زیر، پوشه فعلی به مسیر داده شده تغییر می کند:

```
void change_directory(const std::string path)
```

مسیری که در این دستور داده می شود، همواره به صورت نسبی است. اگر آدرس وارد شده معتبر نباشد باید یک استثنا با پیغام "InvalidPath" پرتاب شود. اگر در پوشه‌ی ریشه هم بودیم، آدرس .. یک آدرس نامعتبر محسوب می‌شود.

مرتب کردن بر اساس نام

با فراخوانی تابع زیر، محتویات فایل‌سیستم با شروع از پوشه فعلی بر اساس نام آن ها مرتب می شوند:

```
void sort_by_name()
```

مرتب کردن بر اساس زمان ساخت

با فراخوانی تابع زیر، محتویات فایل‌سیستم با شروع از پوشه فعلی بر اساس زمان ساخت به طور صعودی (از قدیمی‌ترین تا جدیدترین) مرتب می شوند:

```
void sort_by_creation_time()
```

نمایش درختی

با فراخوانی تابع زیر، محتویات فایل‌سیستم با شروع از پوشه فعلی نمایش داده می شوند. ترتیب نمایش محتویات یک پوشه بر اساس آخرین وضعیت فایل‌سیستم است.

```
void show_tree()
```

به طور پیش‌فرض ترتیب نشان دادن پرونده‌ها بر اساس نام است.

نحوه‌ی تحویل

- کدهای هر بخش را در پوشه‌ای جداگانه با شماره آن بخش قرار دهید و این پوشه‌ها را در قالب یک پرونده‌ی زیپ با نام A6-SID.zip در صفحهٔ CECM درس بارگذاری کنید که SID شمارهٔ دانشجویی شماست؛ برای مثال اگر شماره‌ی دانشجویی شما ۸۱۰۱۹۷۹۹۹ باشد، نام پروندهٔ شما باید A6-810197999.zip باشد.
- برای سه تابع از توابع سوال‌های یک یا دو با استفاده از چهارچوب Catch2 **آزمون واحد** بنویسید. بخشی از نمره شما به این قسمت تعلق دارد.
- این پروژه حتما باید به روش شیءگرایی و به صورت Multi File پیاده‌سازی شود همچنین استفاده از makefile اجباری است.
- برنامهٔ شما باید در سیستم عامل لینوکس و با مترجم g++ با استاندارد c++11 ترجمه و در زمان معقول برای ورودی های آزمون اجرا شود. **دقت کنید** که باید در makefile خود مشخص کنید که از استاندارد c++11 استفاده می‌کنید.
- **دقت کنید** برای این تمرین، یک رابط در اختیار شما قرار داده شده است که درستی برنامه‌ی شما از طریق فراخوانی توابع این رابط سنجیده می‌شود. در واقع تابع اصلی[9] برنامه‌ی شما با توابع اصلی آزمون[10] جایگزین می‌شود و خروجی برنامه مورد بررسی قرار می‌گیرد. **دقت کنید** که نام پرونده‌ای که تابع اصلی شما برای هر یک از پرسش‌ها در آن قرار دارد باید **main.cpp** باشد که با توابع اصلی آزمون جایگزین می‌شود. برای آشنایی بیشتر با رابطه برنامه‌نویسی کاربردی[11]، می‌توانید از این لینک استفاده کنید.
- **تأکید می‌شود** در پایان رشته‌هایی که در توابع what در استثناها بر می‌گردانید حتما کاراکتر '\n' وجود داشته باشد.
- **تأکید می‌شود** که هدف از پروژه طراحی و استفاده‌ی صحیح از مفاهیم وراثت و چندریختی می‌باشد و استفاده از **if یا switch case** برای تشخیص نوع عملگرها در بخش اوّل و نوع گیت‌ها در بخش دوم قابل قبول نخواهد بود.
- تمیزی کد، شکستن مرحله‌به‌مرحلهٔ مسئله و طراحی مناسب، در کنار تولید خروجی دقیق و درست، بخش مهمی از نمره‌ی شما را تعیین خواهد کرد.
- درستی برنامه‌ی شما از طریق آزمون‌های خودکار سنجیده می‌شود؛ بنابراین پیشنهاد می‌شود که با استفاده از ابزارهایی مانند diff خروجی برنامه خود را با خروجی‌هایی که در اختیارتان قرار داده شده است مطابقت دهید. همچنین دقت شود که نام پرونده‌ی اجرایی شما برای هر بخش باید شامل نام آن بخش باشد. برای مثال، نام پرونده‌ی اجرایی مربوط به بخش اوّل باید به صورت 1.out باشد.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت کشف تقلب مطابق قوانین درس با آن برخورد خواهد شد.

---

[9] Main

[10] Test

[11] Application Programming Interface