برنامه‌سازی پیشرفته

تمرین کامپیوتری شماره ۶

مدرس: **رامتین خسروی**

طراحان: **سارا رضائی‌منش، پاشا براهیمی، علی عطاءاللّهی، سروش میرزاسروری، مهیار کریمی**

مهلت تحویل: دوشنبه ۱۹ اردیبهشت ۱۴۰۱، ساعت ۲۳:۵۵

مقدمه

هدف از این تمرین مهارت بیشتر شما در برنامه‌نویسی شیءگرا با استفاده از مفاهیم وراثت و چندریختی است. انتظار می‌رود از تکنیک‌های برنامه‌نویسی که تاکنون در کلاس درس فرا گرفته‌اید یا در هنگام تحویل حضوری تمرین‌ها به شما تذکر داده شده‌است در این تمرین استفاده کنید. طراحی کلاس‌ها، نحوهٔ 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 و سپس مناسب‌ترین طراحی را پیاده‌سازی کنید.

این تمرین از دو بخش تشکیل شده است. در بخش اول با مفاهیم وراثت و چندریختی به پیاده‌سازی یک کتاب‌خانه می‌پردازید و در بخش امتیازی یک سیستم سبد خرید را پیاده‌سازی می‌کنید.

مدیریت خطاها

برای مدیریت خطاها معمولا از مکانیزم استثنا استفاده می‌شود. اما چون هنوز با این مفهوم به طور کامل آشنا نشده‌اید، کافی است که در زمان بروز خطا پیام مربوط به خطا را چاپ کنید و با استفاده از تابع `exit` با مقدار `EXIT_SUCCESS` برای آرگومان آن، برنامه را تمام کنید.

در ادامه‌ی تمرین، در مورد خطاهای مورد نظر توضیح داده شده است. برای فهمیدن بهتر این بخش، به پرونده‌ی نمونه‌ی ورودی توجه کنید.

روش ارزیابی

برای ارزیابی درستی پیاده‌سازی شما در این تمرین، پرونده‌ی `main.cc` شما (که در آن تابع `main` نوشته شده) با پرونده‌ی آزمون جایگزین می‌شود و درستی اجرای برنامه‌ی شما با تابع `main` مورد نظر بررسی خواهد شد؛ بنابراین، حتما در پرونده‌ی `main.cc` که برای هر بخش تمرین می‌نویسید، تنها تابع `main` را پیاده‌سازی کنید.

برای هر بخش تمرین، تعریف یک کلاس به همراه امضای متدهای آن آمده است؛ با توجه به نکته‌ی بالا، نام کلاس و متدهای ذکر شده را تغییر ندهید؛ در صورت نیاز می‌توانید متدهای جدیدی به این کلاس اضافه کنید. در ادامه‌ی توضیح تمرین، تعریف کلاس‌های مورد نظر آمده است.

بخش اول: کتابخانه

در این بخش قرار است که یک سیستم مدیریت مستندات کتابخانه پیاده‌سازی کنیم. در این کتابخانه دو نوع مختلف کاربران یعنی دانشجویان و اساتید وجود دارند. همچنین سه نوع مختلف از مستندات به صورت کتاب عادی، کتاب مرجع و مجله داریم. کاربران می‌توانند مستندها را به امانت بگیرند، طول مدت امانت را به تمدید کنند، و مستندات را به کتابخانه بازگردانند. در ادامه به بررسی قوانین موجود در این کتابخانه می‌پردازیم. توجه داشته باشید که امانت گرفتن یک مستند بر اساس عنوان آن صورت می‌پذیرد و چند کپی مختلف از یک عنوان می‌تواند وجود داشته باشد.

محدودیت تعداد مستندات به امانت گرفته شده: هر استاد در هر زمان می‌تواند ۵ مستند و هر دانشجو می‌تواند ۲ مستند را در امانت داشته باشد.

مهلت تحویل مستندات: مهلت تحویل کتاب‌های عادی ۱۰ روز، کتاب‌های مرجع ۵ روز و مجله‌ها ۲ روز می‌باشد.

تمدید مدت امانت مستندات:

- اعضای کتابخانه می‌توانند مدت زمان نگهداری کتاب‌های عادی و مرجع را **حداکثر دو بار** تمدید نمایند.
- تمدید مجلات امکان‌پذیر نیست.
- میزان تمدید همواره به همان میزان اولیه مهلت تحویل است؛ یعنی هر کاربر با یک بار تمدید کتاب عادی ۱۰ روز دیگر می‌تواند آن کتاب را نزد خود نگه دارد.
- تمدید یک مستند باید **پیش از پایان زمان مجاز تحویل آن مستند** اتفاق بیافتد.
- امکان تمدید در همان روزی که مستند دریافت شده است وجود ندارد.
- همچنین توجه داشته باشید که میزان تمدید همواره از موعد تحویل پیشین مورد محاسبه قرار میگیرد و نه از تاریخ تمدید. به عنوان مثال اگر یک کتاب عادی در روز ۱۳ به امانت گرفته شود، تا روز ۲۳ مهلت تحویل دارد و کاربر می‌تواند از روز ۱۴ تا ۲۳ مهلت امانت را تمدید کند که در این صورت کتاب تا روز ۳۳ مهلت خواهد داشت(در همان بازه هم میتواند تا روز ۴۳ تمدید کند).

جریمه‌ی تاخیر مستندات: اعمال جریمه از روز بعد از پایان گرفتن مهلت تحویل صورت می‌پذیرد. همینطور، جریمه برای کتاب‌های عادی و مرجع، بسته به مدت تاخیر متغیر است:

- **کتاب‌های مرجع:** برای ۳ روز اول دیرکرد، هزینه جریمه هر روز ۵۰۰۰ تومان و برای روزهای بعد هر روز ۷۰۰۰ تومان (مثلاً به ۵ روز دیرکرد، مبلغ ۲۹۰۰۰ تومان جریمه تعلق می‌گیرد).
- **کتاب‌های عادی:** برای ۷ روز اول دیرکرد، هزینه جریمه هر روز ۲۰۰۰ تومان، برای روزهای ۸ تا ۲۱، هر روز ۳۰۰۰ تومان و برای روزهای بعد هر روز ۵۰۰۰ تومان.
- **مجله‌ها:** برای مجله‌های منتشر شده در سال‌های قبل از ۱۳۹۰، هر روز ۲۰۰۰ تومان و برای مجلات منتشر شده در سال‌های بعد از آن، هر روز ۳۰۰۰ تومان.

تعریف کلاس اصلی

تعریف کلاس **Library** به صورت زیر می‌باشد:

```
class Library {
public:
    void add_student_member(string student_id, string student_name);
    void add_prof_member(string prof_name);
    void add_book(string book_title, int copies);
    void add_magazine(string magazine_title, int year, int number, int copies);
    void add_reference(string reference_title, int copies);
    void borrow(string member_name, string document_title);
    void extend(string member_name, string document_title);
    void return_document(string member_name, string document_title);
    int get_total_penalty(string member_name);
    vector<string> available_titles();
    int time_pass(int days);
private:
    // private implementation
};
```

شرح توابع بالا در جدول زیر آمده است:

add_student_member **add_prof_member**	اضافه کردن عضو دانشجو / استاد خطاها: • عضویت فردی با همان نام • تهی بودن رشته‌های نام یا شماره دانشجویی (فرمت شماره دانشجویی مهم نیست)
add_book **add_magazine**	اضافه کردن کتاب / مجله / مرجع

add_refrence	خطاها: • وجود مستندی با همان عنوان (title) • تهی بودن رشته عنوان • صفر یا منفی بودن سال انتشار (year) یا شماره (number) مجلات
borrow **extend** **return_document**	امانت گرفتن / تمدید کردن / بازگرداندن مستندی با عنوان document_title توسط عضوی با نام member_name
get_total_penalty	بازگرداندن مقدار کل جریمه تعلق‌گرفته به عضوی با نام member_name، شامل جریمه کتاب‌هایی که تحویل داده و جریمه کتاب‌هایی که اکنون در امانت دارد.
available_titles	بازگرداندن برداری از عناوین مستندات در دسترس برای امانت گرفتن
time_pass	گذشت زمان به مدت days روز (بخش مدل‌سازی زمان را در ادامه متن ببینید) خطاها: • منفی بودن days

مدل سازی زمان: برای این که شما را از دردسرهای مدیریت تاریخ راحت کنیم، فرض کنید در ابتدای ساخت یک Library، زمان جاری در یک مبدأ از پیش تعریف شده قرار دارد و هر بار که متد time_pass(d) صدا می‌شود، زمان جاری به مدت d روز به جلو می‌رود. فرض می‌شود تمام متدهای دیگر هنگام فراخوانی در زمان جاری انجام می‌شوند.

استثناها

در ادامه پیام خطای مربوط به هرکدام از استثناهایی که در بخش قبل توضیح داده شده‌اند، آمده است:

- عضویت فردی با همان نام:

```
Name already exists
```

- تهی بودن رشته‌های نام یا شماره دانشجویی یا رشته‌ی عنوان:

```
Empty field
```

- وجود مستندی با همان عنوان (title):

```
A document with the specified name already exists
```

- صفر یا منفی بودن شماره (number) مجلات:

```
Invalid number
```

- صفر یا منفی بودن سال انتشار (year):

```
Invalid year
```

- منفی بودن days:

```
Invalid day
```

- تمدید کردن در روز قرض گرفتن کتاب:

```
You can’t extend and borrow a document on the same day
```

- تمدید پس از پایان مهلت تحویل:

```
You can’t renew a document after receiving a penalty
```

- بیش از دوبار تمدید کردن یک مستند:

```
You can’t renew a document more than two times
```

- قرض گرفته شدن بیش از دو مستند توسط دانشجو و بیش از پنج مستند توسط استاد:

```
Maximum number of allowed borrows exceeded
```

- تمدید کردن مجله:

```
You can’t renew magazines
```

- تمدید یا بازگرداندن مستندی که قرض نگرفته ایم:

```
You have not borrowed this document
```

- قرض کردن دوباره مستند در بازه ای که آن را قرض داریم:

```
You borrowed this document already
```

- قرض کردن مستندی که وجود ندارد یا تمام شده است:

```
This document does not exist
```

بخش امتیازی: سبد خرید

یک فروشگاه بزرگ قصد دارد که برای سفارش‌های خود یک سامانه‌ی محاسبه‌ی قیمت طراحی کند. پس از انجام مکاتبات جان‌فرسا، مدیران این فروشگاه مسئولیت طراحی این بخش را به دانشکده‌ی برق و کامپیوتر دانشگاه تهران سپردند.

در این فروشگاه، کالاهایی برای فروش وجود دارند. به هر کالا در انبار این فروشگاه یک شناسه‌ی یکتا اختصاص داده شده است. با داشتن این شناسه‌ی یکتا می‌توانیم قیمت و وزن کالای مورد نظر را از انبار دریافت کنیم. در انبار این

فروشگاه، تعدادی نامتناهی جعبه نیز وجود دارد که برای بسته‌بندی کالاها قابل استفاده هستند. همچنین، در هر جعبه‌ای می‌توانیم تعداد دلخواهی کالا قرار دهیم. همچنین، می‌توانیم یک جعبه را نیز در یک جعبه‌ی دیگر بگذاریم.
سامانه‌ای که برای سبد خرید این فروشگاه طراحی می‌کنیم باید بتواند لیستی از کالاها و جعبه‌ها را دریافت کند و هزینه‌ی نهایی را که مجموع هزینه‌ی کالاها و جعبه‌هاست محاسبه نماید. هزینه‌ی کالایی که در یک جعبه قرار نگرفته برابر قیمت خود آن کالا می‌باشد. برای محاسبه‌ی هزینه‌ی یک جعبه از رابطه‌ی زیر استفاده می‌کنیم:

$$\text{Box Cost} = \left(\sum_{\text{Item} \in \text{Box}} \text{Item Cost} \right) + \text{Weight Cost}$$

برای محاسبه‌ی هزینه‌ی وزن نیز مطابق روش زیر عمل می‌کنیم:

- اگر مجموع وزن محتویات جعبه کمتر از **۲۰ واحد وزن** باشد:

$$\text{Weight Cost} = \left\lfloor \left(\sum_{\text{Item} \in \text{Box}} \text{Item Weight} \right) / 2 \right\rfloor$$

- اگر مجموع وزن محتویات جعبه بیش‌تر یا مساوی **۲۰ واحد وزن** باشد:

$$\text{Weight Cost} = \left\lfloor \left(\sum_{\text{Item} \in \text{Box}} \text{Item Weight} \right) / 10 \right\rfloor$$

توجه کنید که جعبه‌ها نیز می‌توانند در یکدیگر قرار بگیرند. همچنین، خود جعبه‌ها وزن ندارند و تنها وزن کالاها در نظر گرفته می‌شود.

تعریف کلاس اصلی

تعریف کلاس **CartManager** به صورت زیر می‌باشد:

```
class CartManager {
public:
    CartManager();
    ~CartManager();
    void add_item(int id, int cost, int weight);
    void add_box(int id);
    void add_to_box(int box_id, int id);
    int evaluate_cart();
private:
    // private implementation
};
```

در ادامه توضیح عملکرد متدهای این کلاس آمده است:

add_item **add_box**	اضافه کردن کالا / جعبه به سبد خرید خطاها: • کالا یا جعبه‌ای با شناسه‌ی مورد نظر در سبد خرید وجود داشته باشد.
add_to_box	اضافه کردن یک کالا / جعبه به جعبه‌ای دیگر. در این تابع **box_id** جعبه‌ی مقصد خواهد بود. همچنین، تضمین می‌شود که این تابع به‌ازای هر مقدار **id** حداکثر یک بار صدا زده می‌شود (کالایی را از یک جعبه به جعبه‌ی دیگر منتقل نمی‌کنیم.) خطاها: • حداقل یکی از دو شناسه در سبد خرید نباشند. • شناسه‌ای که برای جعبه‌ی مقصد مشخص شده است، متعلق به یک جعبه نباشد.
evaluate_cart	محاسبه‌ی هزینه‌ی نهایی سبد خرید با توجه به توضیحات ارائه شده

استثناها

در ادامه پیام خطای مربوط به هرکدام از استثناهایی که در بخش قبل توضیح داده شده‌اند، آمده است:

- کالا یا جعبه‌ای با شناسه‌ی مورد نظر در سبد خرید وجود داشته باشد:

```
Item already exists in cart
```

- حداقل یکی از دو شناسه در سبد خرید نباشند:

```
Not found in cart
```

- شناسه‌ای که برای جعبه‌ی مقصد مشخص شده است، متعلق به یک جعبه نباشد:

```
Not a box
```

راهنمایی

توصیه می‌شود برای انجام این بخش تمرین، با الگوی طراحی Composite آشنا شوید. برای آشنایی بیش‌تر با این الگو می‌توانید به این راهنما مراجعه کنید.

نکات تکمیلی

- طراحی درست وراثت، رعایت سبک برنامه‌نویسی درست و تمیز بودن کد برنامه‌ی شما در نمره‌ی تمرین تأثیر زیادی دارد. برای مثال استفاده از if یا switch case برای تشخیص نوع زیرکلاس یک کلاس پدر نشان‌دهنده طراحی نادرست وراثت است.
- در تمامی مراحل این پروژه سعی کنید از قوانین ارث‌بری استفاده کنید و هر جا که ممکن است رفتار کلاس‌ها را به صورت چندریخت (polymorphic) پیاده‌سازی کنید و از بررسی مجزای کلاس‌ها خودداری کنید.

نحوه‌ی تحویل

- ابتدا مطمئن شوید که پرونده‌های شما دقیقا مطابق الگوی زیر است:

```
.
├── library
│   └── main.cc
│   └── makefile
│   └── ...
└── shopping_cart
    └── main.cc
    └── makefile
    └── ...
```

سپس پرونده‌های خود را در قالب یک **پرونده‌ی zip** با نام A6-<SID>.zip در صفحهٔ elearn درس بارگذاری کنید که SID شمارهٔ دانشجویی شماست؛ برای مثال اگر شماره‌ی دانشجویی شما ۸۱۰۱۰۰۹۹۹ است، نام پروندهٔ شما باید A6-810100999.zip باشد.

- **دقت کنید** که پرونده‌ی zip آپلودی شما باید پس از Unzip شدن شامل پرونده‌های پروژه شما مطابق قالب بالا باشد و از zip کردن پوشه‌ای که داخل آن فایل‌های پروژه‌تان قرار دارد خودداری فرمایید.
- برنامه شما باید حتماً طراحی شیءگرا داشته باشد و حتما در آن از وراثت و چندریختی برای بررسی انواع سوالات و دریافت پاسخ‌ها استفاده شده باشد. این موضوع قسمت بزرگی از نمره‌ی شما را شامل می‌شود.
- **دقت کنید** که پروژه‌ی شما باید Multi-file باشد و Makefile داشته باشد. همین‌طور در Makefile خود مشخص کنید که از استاندارد C++11 استفاده می‌کنید.
- **دقت کنید** که نام پرونده‌ی اجرایی شما در هر دو بخش باید `main` باشد.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت کشف تقلب مطابق قوانین درس با آن برخورد خواهد شد.