 به نام خدا

تمرین سری اول

نام درس:

**برنامه نویسی پیشرفته**

نام استاد:

**دکتر جهانشاهی**

نگارش :

**سجاد قدیری**

فهرست صفحه

[تمرین سری اول 3](#_Toc96300838)

[1-1-تابع zeros و ones 3](#_Toc96300839)

[1-2- تابع random و show 4](#_Toc96300840)

[1-3- تابع multiply 5](#_Toc96300841)

[1-4-تابع sum 6](#_Toc96300842)

[1-5-تابع Transpose و Minor 7](#_Toc96300843)

[1-6-تابع determinant 8](#_Toc96300844)

[1-7-تابع Inverse 9](#_Toc96300845)

[1-8-تابع concatenate 10](#_Toc96300846)

[1-9-تابع ero\_swap و ero\_multiply 11](#_Toc96300847)

[1-10-تابع ero\_sum 12](#_Toc96300848)

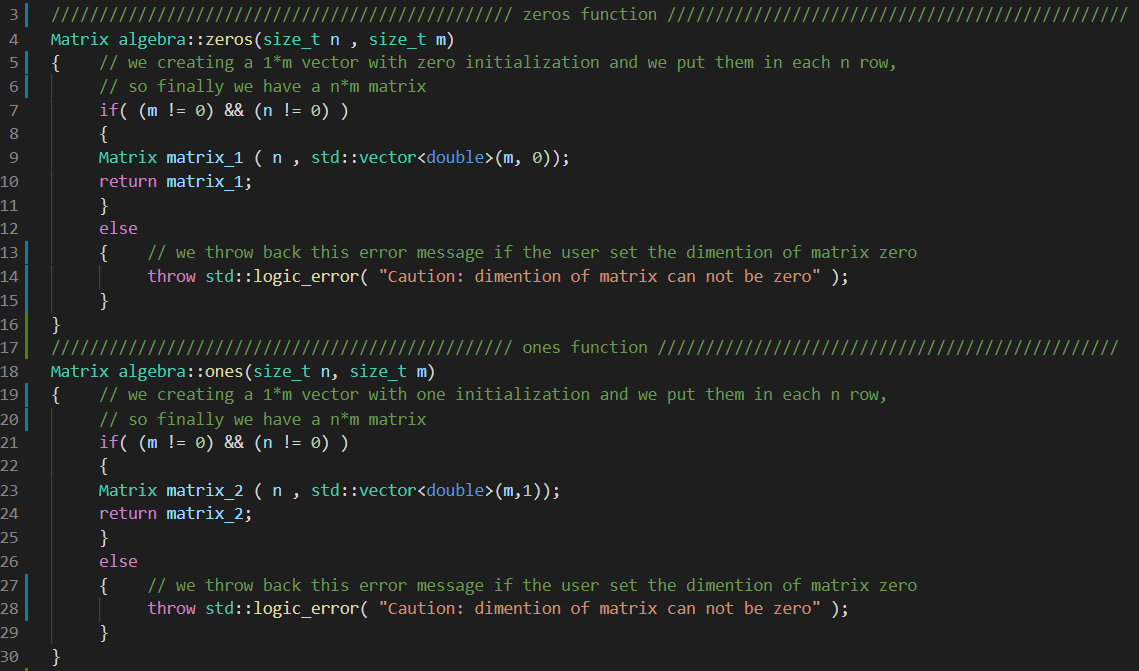
[1-11-تابع upper\_triangular 13](#_Toc96300849)

# تمرین سری اول

## 1-1-**تابع zeros و ones**

در این 2 تابع ابتدا خطاهای احتمالی که ممکن است از طرف کاربر رخ دهد را مورد بررسی قرار دادیم.

اینکه کاربر ابعاد ماتریس را برابر با صفر قرار ندهد و اگر به اشتباه این اتفاق رخ داد یک پیام اخطار برای او بازگردانده شود. اگر این اتفاق رخ نداد ، یک بردار با ابعاد 1\*m که با مقادیر اولیه صفر مقداردهی شده اند را ساخته و به کمک این بردار ماتریس با ابعاد n\*1 را پر می‌کنیم. در واقعا با توجه به تعریف ماتریسی که در فایل header استفاده کردیم ماتریس های ما نوعی بردارهایی از بردار هستند.(vector of vectors)

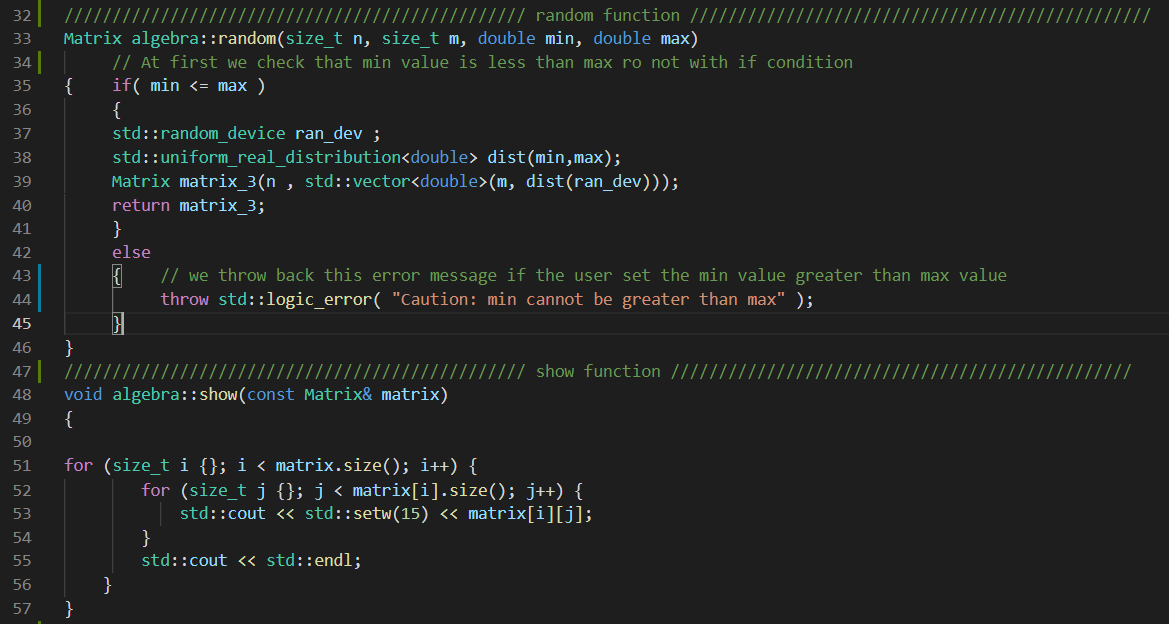


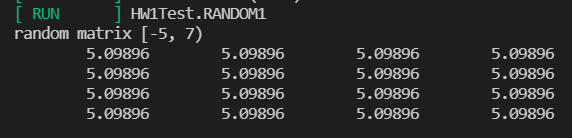
## 1-2- تابع random و show

در این 2 تابع ابتدا خطاهای احتمالی که ممکن است از طرف کاربر رخ دهد را مورد بررسی قرار دادیم.

اینکه کاربر مقدار min را از max بیشتر مقداردهی نکند(و برعکس)و اگر به اشتباه این اتفاق رخ داد یک پیام اخطار برای او بازگردانده شود.اگر این اتفاق رخ نداد، با استفاده از یک شی از random\_device

و با محاسبه اختلاف مقدار min و max و سپس ایجاد یک ماتریس با ابعاد دلخواه کاربر(n\*m) که مقدار آن با همان دستورdist به صورت رندوم انتخاب می‌شود.

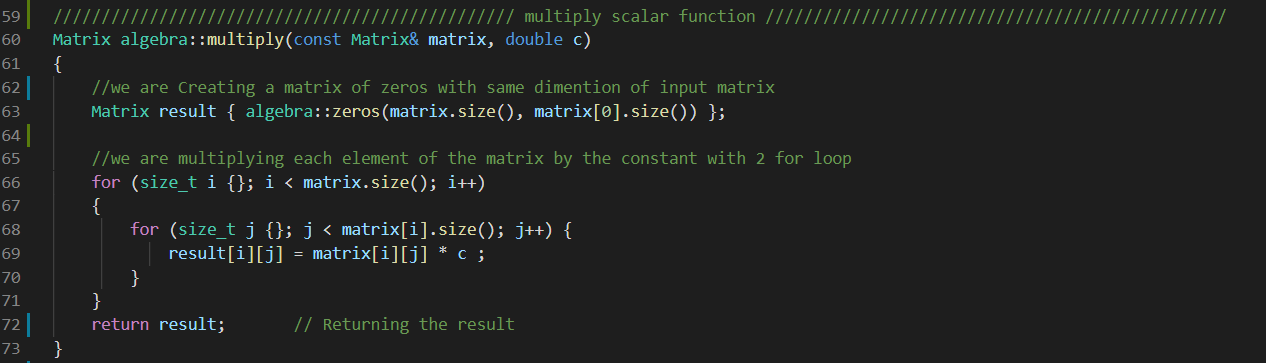


همچنین در تابع show با استفاده از دستور set width که در کتابخانه iomanip قرار دارد می‌توانیم خروجی را در موقعیت دلخواهی از صفحه خروجی قرار دهیم که ما بر روی 15 قرار دادیم و به این ترتیب ماتریس هایی به فرم زیر در خروجی چاپ شد:

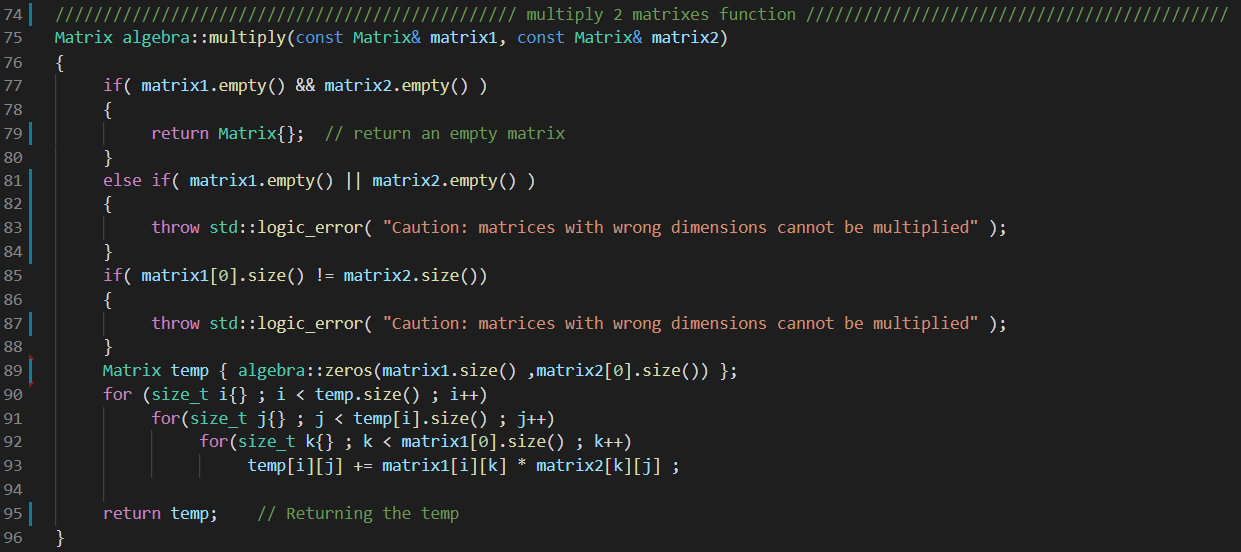
## 1-3- **تابع multiply**

در این تابع ابتدا یک ماتریس با مقادیر اولیه تماما صفر به کمک تابع zeros که قبلا ساخته ایم می‌سازیم.

ابعاد این ماتریس را نیز همان ابعاد ماتریسی که در ورودی تابع گرفتیم در نظر می‌گیریم.

سپس با 2 حلقه تو در تو المان به المان درایه های ماتریس را در اسکالر c که در ورودی تابع گرفتیم ضرب می‌کنیم و سپس نتیجه را در خروجی برمی‌گردانیم.

برای ضرب 2 ماتریس نیز ابتدا خطاهای احتمالی که ممکن است از طرف کاربر رخ دهد را مورد بررسی قرار دادیم. اینکه کاربر 2 ماتریس خالی وارد نکرده باشد و یا اینکه اگر یکی از آنها خالی بود پیام اخطاری

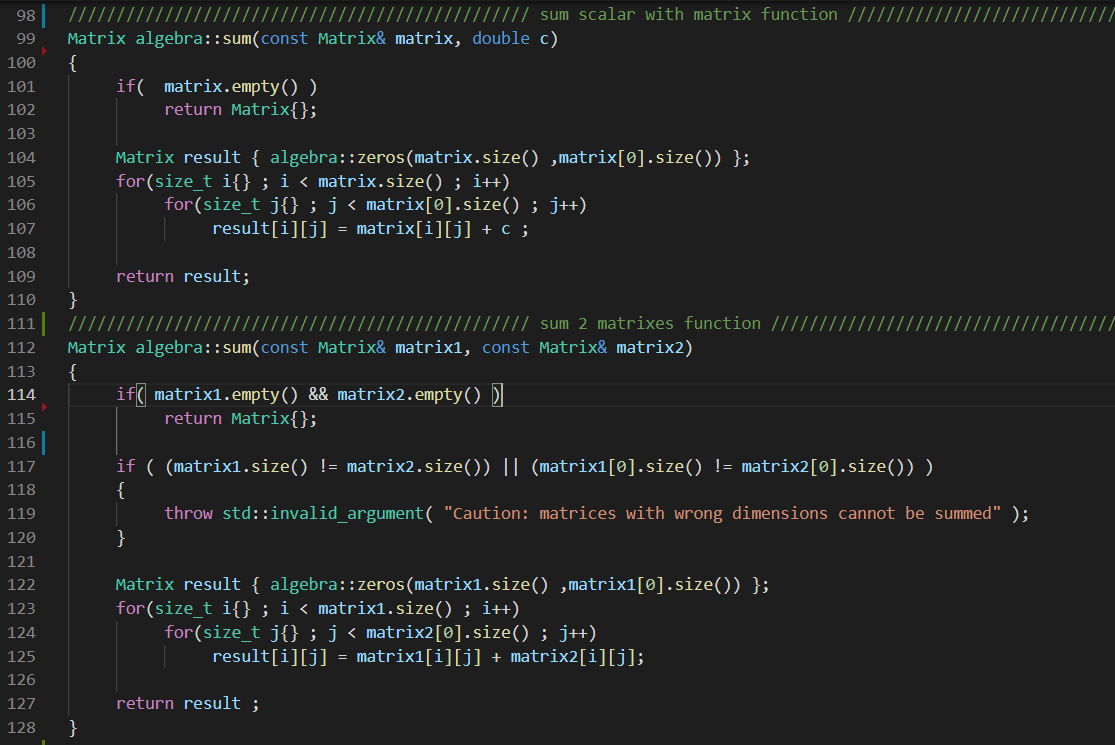
از جنس logic\_error و یا اینکه از لحاظ ابعاد ماتریس ها، مشکلی در ضرب 2 ماتریس وجود داشت پیام اخطاری برگردانده شود.

## 1-4-**تابع sum**

در این تابع ابتدا خطاهای احتمالی که ممکن است از طرف کاربر رخ دهد را مورد بررسی قرار دادیم. اینکه کاربر ماتریس خالی وارد نکرده باشد و اگر این اتفاق افتاد یک ماتریس خالی برگردانده می‌شود.

سپس یک ماتریس با مقادیر اولیه تماما صفر به کمک تابع zeros که قبلا ساخته ایم می‌سازیم.

ابعاد این ماتریس را نیز همان ابعاد ماتریسی که در ورودی تابع گرفتیم در نظر می‌گیریم.

سپس با 2 حلقه تو در تو المان به المان درایه های ماتریس را با اسکالر c که در ورودی تابع گرفتیم جمع می‌کنیم و سپس نتیجه را در خروجی برمی‌گردانیم.

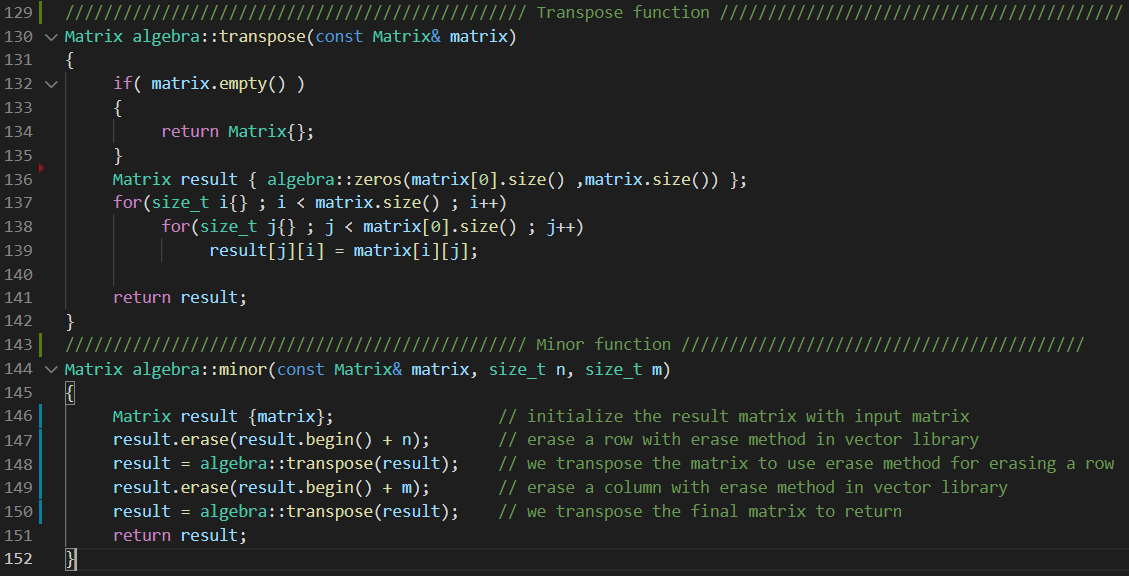
برای جمع 2 ماتریس نیز به همین شکل ابتدا خالی نبودن مانریس ها را بررسی می‌کنیم. مورد دیگری که بررسی کردیم یکسان بودن ابعاد ماتریس ها برای جمع پذیر بودن است و اگر اینطور نبود یک خطای منطقی برمی‌گردانیم. سپس همانند قبل با 2 حلقه تو در تو المان های متناظر هر ماتریس را با دیگری جمع می‌کنیم.

## 1-5-**تابع Transpose و Minor**

برای ماتریس ترنسپوز ابتدا خطای احتمالی را بررسی کردیم. اگر کاربر ماتریس خالی وارد کند یک ماتریس خالی برمی‌گردانیم. سپس یک ماتریس با مقادیر اولیه تماما صفر به کمک تابع zeros که قبلا ساخته ایم می‌سازیم. ابعاد این ماتریس را نیز **برعکس** ابعاد ماتریسی که در ورودی تابع گرفتیم در نظر می‌گیریم.

حال با 2 حلقه تو در تو هر المان از ماتریس ورودی را در المانی با ایندکس برعکس از ماتریس result قرار می‌دهیم.

برای تابع ماینور ابتدا یک ماتریس با همان ابعاد ماتریس ورودی مقداردهی اولیه می‌کنیم. سپس با متد erase می‌توانیم ردیف دلخواهی را از یک ماتریس حذف کنیم . سپس با عمل ترنسپوز توانستیم ستون مورد نظر ماتریس را نیز حذف کنیم.



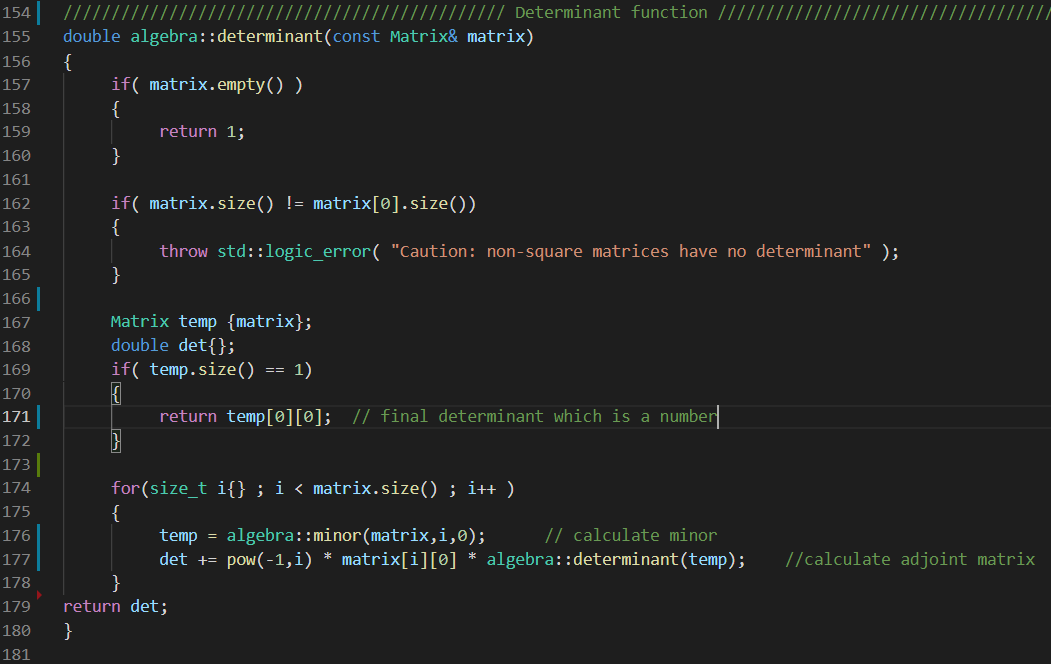
## 1-6-**تابع determinant**

**برای تابع دترمینان ابتدا چک می‌کنیم که کاربر ماتریس خالی وارد نکرده باشد در غیر اینصورت مقدار 1 را برمی‌گردانیم.(در خود توضیحات گیت هاب گفته شده بود)**

**سپس این خطای احتمالی را چک می‌کنیم که ماتریس ورودی مربعی باشد در غیر اینصورت یک اخطار منطقی برمی‌گردانیم.**

برای محاسبه دترمینان به صورت بازگشتی از خود تابع دترمینان در درون خودش استفاده می‌کنیم و در نهایت وقتی به یک عدد اسکالر رسیدیم آن را بر‌می‌گردانیم.

در درون حلقه نیز ابتدا ماینور را محاسبه کرده و سپس ماتریس الحاقی را محاسبه می‌کنیم.



## 1-7-**تابع Inverse**

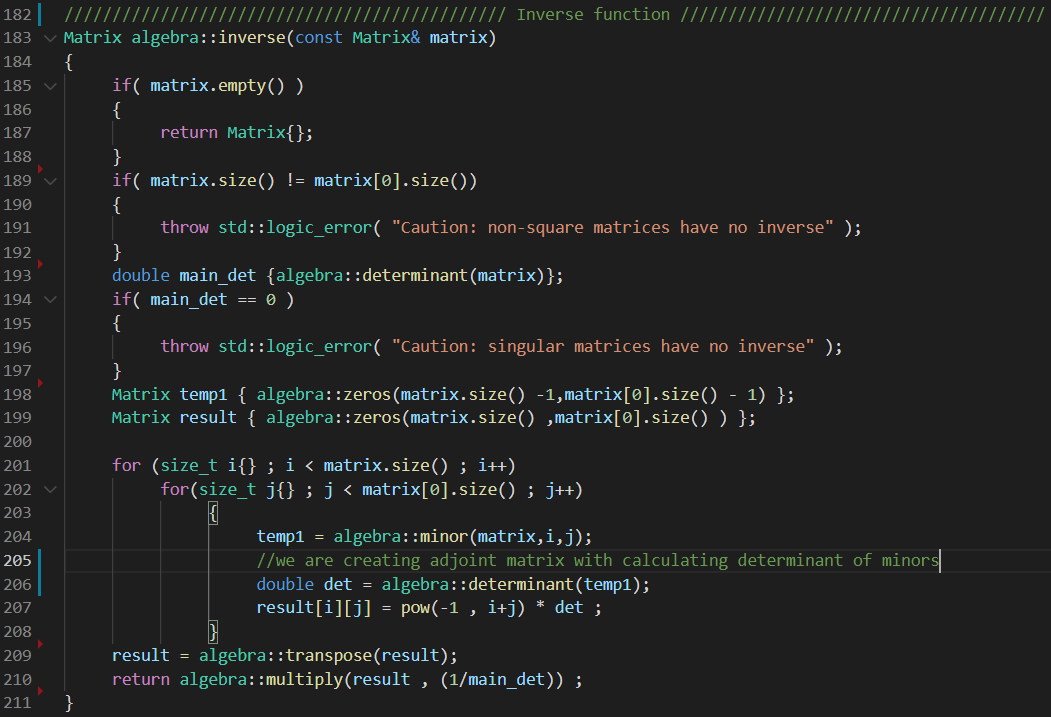
مانند قبل همه ملاحظات گفته شده برای خطاهای احتمالی را بررسی کردیم.

سپس دترمینان ماتریس اصلی را در متغییرmain\_det مقداردهی اولیه کردیم.

سپس دو ماتریس با مقادیر اولیه تماما صفر به کمک تابع zeros که قبلا ساخته ایم می‌سازیم. ابعاد ماتریس اول یکی کمتر از ماتریس اصلی برای ماینور و دومی هم ابعاد ماتریس اصلی می‌باشد.

سپس با 2 حلقه تو در تو ابتدا ماینور را محاسبه می‌کنیم و سپس با محاسبه ی دترمینان ماینور ها   
ماتریس الحاقی را می‌سازیم و سپس منفی یک به توان جمع اندیس درایه ها را اعمال می‌کنیم.

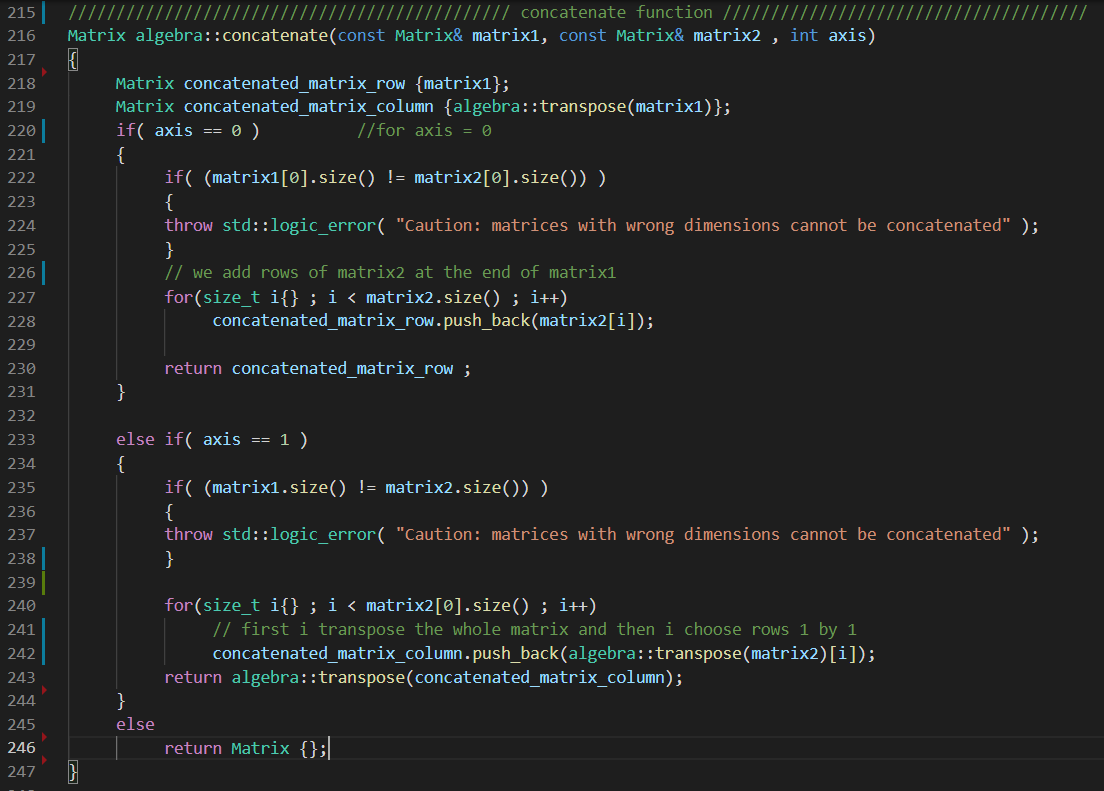
در نهایت ماتریس را ترنسپوز کرده و بر دترمینان ماتریس اصلی تقسیم می‌کنیم.



## 1-8-**تابع concatenate**

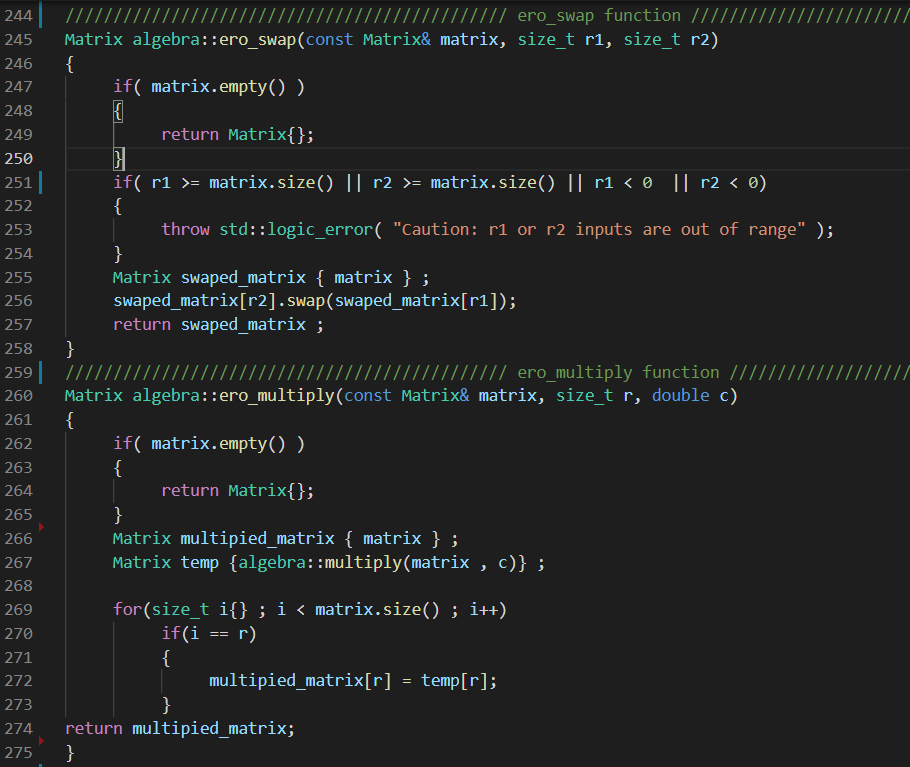
با توجه به ورودی axis که صفر است یا یک به 2 قسمت تقسیم می‌کنیم.

در هر حالت خطا های احتمالی را بررسی کردیم و هم چنین با توجه به نوع axis با دستور push\_back سطر به سطر ردیف های ماتریس دومی را به ماتریس اولی اضافه می‌کنیم.



## 1-9-**تابع ero\_swap و ero\_multiply**

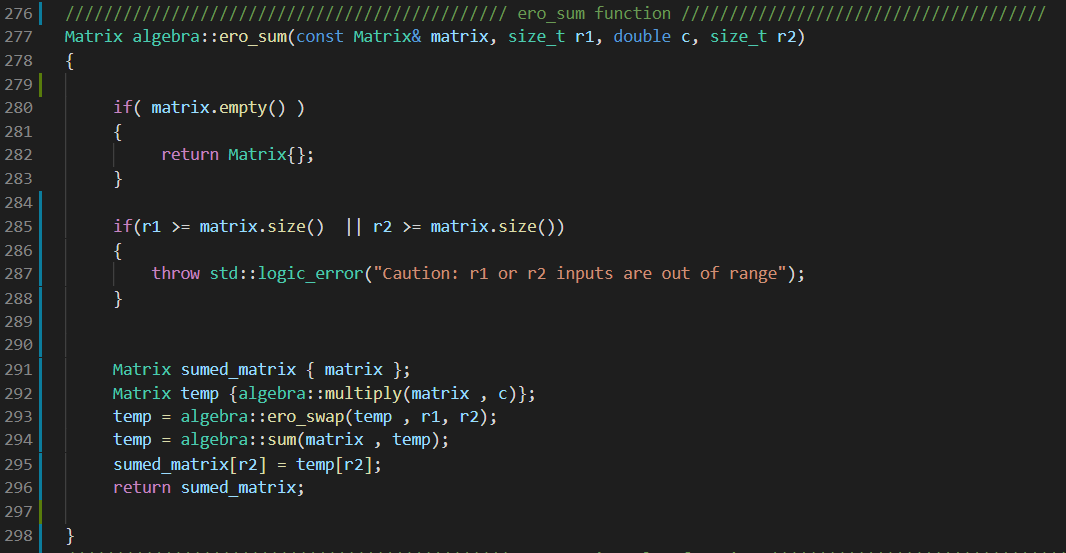
بررسی ملاحظات خطا های احتمالی را انجام دادیم. سپس با دستور swap که در کتابخانه vector است این عمل را انجا می‌دهیم. 😊



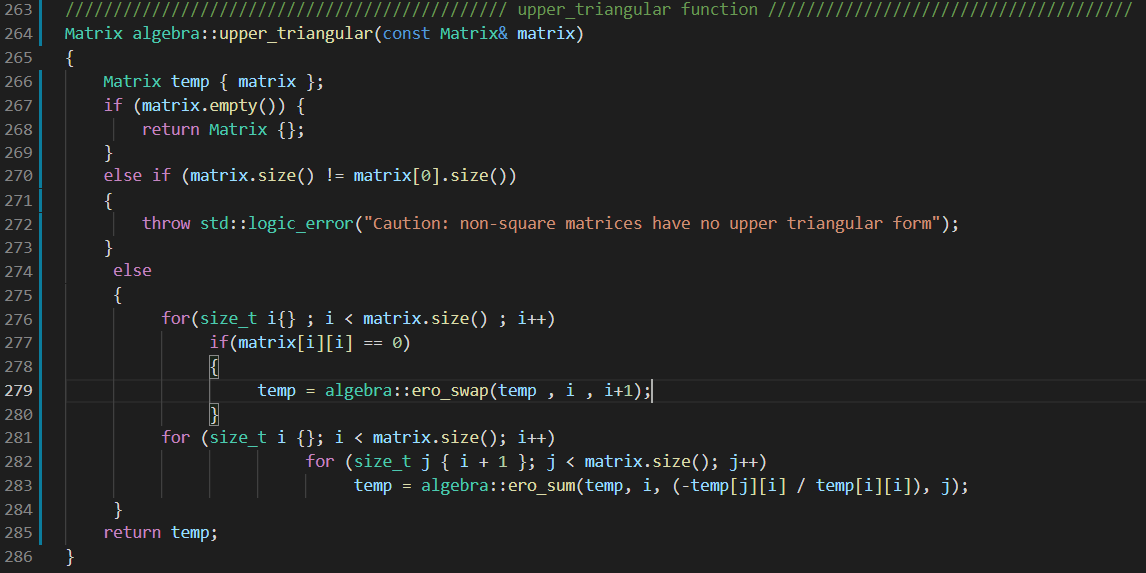
## 1-10-**تابع ero\_sum**

در این تابع همه ملاحظات قبلی انجام شده و بررسی می‌شود. سپس با استفاده از توابع multiply و

Ero\_swap که در قبل تعریف و مشخص کردیم ، یک سطر از ماتریس ورودی را در یک اسکالر که از ورودی دریافت کردی ضرب و سپس با یک سطر دیگر جمع می‌کنیم و در همان سطر دومی میریزیم.



## 1-11-تابع upper\_triangular

در این تابع ابتدا ملاحظات قبلی را بررسی می‌کنیم .

سپس چک می‌کنیم که اگر درایه صفری روی قطر اصلی باشد آن را با استفاده از تابع ero\_swap سطر ها را جابهجا می‌کنیم تا به حالت عادی بازگردیم . سپس با 2 حلقه تو در تو و شروع حلقه داخلی از درایه ای که در رو روی قطر قرار دارد ، با استفاده از تابع ero\_sum ضریبی از سطر اصلی را با سطر های پایین اش جمع می‌کنیم تا ماتریس را به صورت مثلثی در بیاوریم.