



گزارش تمرین سری اول (C++)

حمیدرضا دشت آبادی

۹۸۲۳۰۳۲

پیاده سازی کتابخانه جبر خطی در C++

—

استاد جهانشاهی

—

[h.dashtabadi@gmail.com](mailto:h.dashtabadi@gmail.com)

## خلاصه :

در این تمرین یک namespace با عنوان algebra تعریف می شود که درون آن توابعی کاربردی در حوزه جبر خطی تعریف شده است. declaration توابع داخل پوشه include فایل hw1.h تعریف شده و implementation مربوطه در پوشه src، فایل hw1.cpp موجود است. برای اجرا کردن برنامه ابتدا یک ایمپج داکر از محیط wsl2 و با dockerfile که در github تمرین موجود بود ساختیم. سپس از طریق ssh و با localhost ip (127.0.0.1) محیط ادیتور vscode را به pwd منتقل کردم. سپس در ترمینال ایجاد شده به مسیر build رفتم و از آنجا یک makefile با استفاده از cmake ساختم. همچنین یک شاخه git ایجاد کردم و یازده commit از مراحل مختلف کد زنی این برنامه حجیم ایجاد کردم و آن ها را به repository که ساختم push کردم :

[Commits · hamid-rd3/HamidrezaDashtabadi9823032 \(github.com\)](https://github.com/hamid-rd3/HamidrezaDashtabadi9823032)

## اقدامات اولیه برنامه :

در ابتدا یک namespace در hw1.h ایجاد کردم (بین #define AP\_HW1\_H و #endif // AP\_HW1\_H). این نوع syntax که #include guards نام دارد برای جلوگیری از double declaration یا مقدار دهی دوباره header مربوط به hw1.h می شود و از آن محافظت می کند. لذا در ادامه کتابخانه های مختلفی که نیاز بود و declaration مربوط به توابع را به algebra اضافه کردم.

همچنین در طول نوشتن برنامه ها اگر return در آخر تابع نوشته نمی شد کامپایلر warning می داد :

[Control reaches the end of a non-void function](#)

## بررسی توابع :

```
Matrix zeros(size_t n, size_t m)
Matrix ones(size_t n, size_t m)
```

توابع اولیه (zeros و ones) صرفاً برای یادگیری syntax مربوطه بود و نکته خاصی نداشت. صرفاً مقدار دهی (initialize) وکتور دو بعدی با n سطر m ستون که با عنوان Matrix تعریف کردیم و return کردن آن ماتریس.

```
Matrix random(size_t n, size_t m, double min, double max)
void show(const Matrix& matrix)
```

برای تست کردن تابع random نیاز بود که تابع show را نیز تعریف کنیم. لذا در تابع اول از std::random\_device دادن seed به موتور تولید اعداد تصادفی ; mt(rd()) std::mt19937 داده شد و سپس با استفاده از std::uniform\_real\_distribution<double> dist(min, max); اعداد تصادفی مرحله قبل را به صورت یک بازه تصادفی (min, max) در آوردیم و ادامه با استفاده از روش مقداردهی دو مرحله ای (تعریف یک وکتور یک بعدی خالی v1 و pushback آن به ماتریس) که در کد به طور کامل توضیح داده شده به ماتریس مقادیر تصادفی دادم. برای

`show()` نیز با استفاده از `std::setprecision(3) << std::setw(7)` به اندازه 7space به هر درایه ماتریس مکان تخصیص دادم که ماتریکسمان زیباتر نشان داده شود و دقت نمایش اعداد را 3decimal کردم.

```
Matrix multiply(const Matrix& matrix, double c)
Matrix multiply(const Matrix& matrix1, const Matrix& matrix2)
```

دو تابع `multiply` تعریف می شود با یک اسم (`overload function`) اما چون متغیرهای ورودی این دو تابع با هم تفاوت دارد `compiler` هیچ مشکلی برخورد نمی کند. در ابتدا همانطور که در کد تست این بخش آمده برای `empty` بودن ماتریس ها از `if else` ها استفاده کردم تا اگر ماتریس یا ماتریس ها خالی بودند یک ماتریس خالی را برگرداند تا عمل ضرب روی آن ها انجام نشود و کامپایلر ارور `segmentation error` یا خطای محاسباتی ندهد. همچنین برای ضرب دو ماتریس باید چک کنیم که تعداد ستون ماتریس اول باید با تعداد سطر ماتریس دوم برابر باشد در غیر اینصورت با تابع `throw` یک `logic_error` یا خطای منطقی به کامپایلر داده می شود واز ادامه کامپایل برنامه جلو گیر می شود.

در الگوریتم این برنامه از سه تا `loop` استفاده شد که اولی در سطرهای ماتریس ۱ می گردد و دومی در ستونهای ماتریس ۲ و سومی در ستونهای ماتریس ۱ و سطرهای ماتریس ۲ تا متصل کننده دو لوپ اول باشد .

همچنین در این برنامه از نوع `range based loop` برای جمع کردن درایه های ماتریس با یک عدد ثابت استفاده شد که حجم کمتر و زیبایی بیشتری به بدنه کد داد.

```
Matrix sum(const Matrix& matrix, double c)
Matrix sum(const Matrix& matrix1, const Matrix& matrix2)
```

این قسمت هم شباهت زیادی به دو تابع قبلی دارد منتها با این تفاوت که اگر یکی از ماتریس ها خالی بود و دیگری مقدار اولیه داشت می بایست خطای منطقی می دادم. همچنین سطر ها و ستون های ماتریس اول با دوم باید برابر باشد وگرنه خطای منطقی بدهد.

```
Matrix transpose(const Matrix& matrix)
Matrix minor(const Matrix& matrix, size_t n, size_t m)
```

در تابع ترانپزاده صرفا جای `i` و `j` عوض شد برای اینکه ماتریس در ستون ها بگردد نه سطر ها و علاوه بر سادگی کد کاربرد فراوانی دارد در محاسبه کهاد و معکوس از آن استفاده می شود. برای محاسبه `minor` ماتریس باید مربعی باشد یعنی تعداد سطر و ستون های آن برابر باشد لذا یک خطای منطقی برای نقض آن شرط قرار دادم. کهاد زیرمجموعه ای از ماتریس با توجه به درایه `(n,m)` ماتریس می سازد که سطر و ستون درایه را نادیده می گیرد و مابقی را به صورت یک ماتریس با ابعاد کوچک تر (یک واحد) عینا بر می گرداند. این تابع هم در محاسبه دترمینان و معکوس ماتریس کاربرد دارد .

```
Matrix transpose(const Matrix& matrix)
Matrix minor(const Matrix& matrix, size_t n, size_t m)
```

در محاسبه دترمینان از تابع بازگشتی استفاده شده است که در مسیر رفت دترمینان ماتریس بزرگتر از ضرب درایه ها های سطر اول با علامت مثبت منفی یک در میان در دترمینان کهاد درایه مربوطه (cofactor) و در نهایت جمع همه آن ها می باشد . همین طور اندازه ماتریس های کوچک تر می شود تا در نهایت به ماتریس یک در یک می رسیم که دترمینانش با خودش برابر است حالا مسیر بازگشت طی می شود و همینطور دترمینان ماتریس های بزرگ تر محاسبه می شود تا در نهایت دترمینان ماتریس اصلی محاسبه می شود. برای معکوس گرفتن از تابع باید حواسمان باشد که رنک ماتریس کامل باشد یا دترمینان آن مخالف صفر باشد. برای محاسبه معکوس دترمینان کهاد یک درایه گرفته می شود و در منفی یک به توان جمع سطر و ستون آن درایه ضرب می شود و در همان محل ریخته می شود. سپس ترانواده معکوس اولیه را بدست می آوریم و در نهایت تقسیم بر دترمینان خود ماتریس می کنیم (با استفاده از تابع multiply) (به همین خاطر است که نباید دترمینانش صفر باشد).

```
Matrix concatenate(const Matrix& matrix1, const
Matrix& matrix2, int axis=0)
```

این تابع در واقع دو تابع است که با ورودی سوم متمایز از یکدیگر می شوند. در نوع اول ابتدا دو بردار خالی تعریف می کنیم و آن ها را با مقادیر سطر iام دو ماتریس پر می کنیم. سپس با استفاده از تابع insert بردار دوم را در طول بردار اول قرار می دهیم و همانند قبل در for خارجی به ماتریس pushback می کنیم. در نوع دوم ابتدا ماتریس را با درایه های ماتریس اول پر می کنیم و سپس با درایه های ماتریس دوم .

```
Matrix ero_swap(const Matrix& matrix, size_t r1, size_t r2)
```

در این تابع ابتدا بررسی می شود که آیا r1 یا r2 از تعداد سطر ماتریس بیشتر نباشند. سپس مقادیر ماتریس را به ماتریس resultat می دهیم مگر اینکه در سطر r1 باشیم در این صورت مقادیر سطر r2 را به ماتریس resultat اضافه می کنیم و برعکس .

```
Matrix ero_multiply(const Matrix& matrix, size_t r, double c)
```

در این تابع هم همانند قبل در ابتدا بررسی می شود که مبادا سطری که می خواهیم با C جمع زده شود از تعداد سطر های خود ماتریس بیشتر نباشد. در اینجا اگر در سطر rام نباشیم مقادیر قبلی را به ماتریس resultat می دهیم و اگر در سطر rام باشیم با استفاده از  $matrix[i][j] * c$  درایه آن سطر و ستون را در C ضرب می کنیم و با استفاده از loop همین کار را برای ستون های بعدی نیز تکرار می کنیم .

```
Matrix ero_sum(const Matrix& matrix, size_t r1, double c, size_t r2)
```

چون نمی توانیم مقادیر یک **vector** را تغییر دهیم یعنی فقط می توانیم به آن عضو اضافه کنیم یا کل درایه های آن را با هم تغییر دهیم مثلاً یک ماتریس را درون ماتریس دیگری بریزیم ، مجبوریم یک ماتریس جدید **newmatrix** تعریف کنیم و مقادیر ماتریس را درون آن ریخته و درایه **r1** آن را در **C** ضرب کنیم (با استفاده از **ero\_multiply**) و سپس **resultat** را با مقادیر ماتریس ورودی پر می کنیم مگر در سطر **r2** که سطر ماتریس ورودی با سطر **r1** **newmatrix** جمع زده می شود و در خروجی ریخته می شود.

```
Matrix upper_triangular(const Matrix& matrix)
```

در اینجا هم ابتدا چک می شود که ماتریس خالی نباشد یا غیر مربعی نباشد سپس عملیات محورگیری را انجام می دهیم تا در صورت امکان عناصر روی قطر اصلی صفر نباشد به طوریکه در هر درایه قطر اصلی که صفر است درایه های زیری آن چک می شود که اگر مخالف صفر بود ، سطر مربوط به درایه غیر صفر با سطر درایه قطر اصلی عوض شود. در مرحله دوم اگر درایه روی قطر اصلی صفر بود یعنی تمام درایه های زیری آن صفر بوده ولذا کار خاصی نمی توانیم انجام دهیم و صرفاً سراغ درایه بعدی قطر اصلی می رویم. ذکر این نکته حائز اهمیت است که ماتریس بالامثلثی یعنی درایه های زیر قطر اصلی باید صفر باشند و صفر بودن یا نبودن بقیه درایه ها تعیین نگردیده است. سپس با استفاده از روش حذفی گوسی درایه های زیر قطر اصلی را صفر می کنیم با روش (**Initializing from another vector**).