برنامه‌سازی پیشرفته

تمرین شماره‌ی ۶

استاد: **رامتین خسروی**

طراحان: **محمّدرضا کیانی‌فر**، **کامران حسینی**، **محمّد مریدی**

مهلت تحویل: شنبه چهاردهم اردیبهشت ماه 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.

۱. اشکال هندسی سه بعدی

در این تمرین قصد داریم با الگو برداری از کد اشکال هندسی دوبعدی که در کلاس درس بررسی شد، به پیاده‌سازی چند ویژگی برای چند نوع شکل هندسی سه بعدی به شکل چندپرونده‌ای[3] بپردازیم.

شرح تمرین

شکل‌هایی که شما در این تمرین با آن‌ها کار خواهید کرد، به این شرح است:

نام	سازنده[4]	رابطه‌ی حجم[5]
کره (Sphere)	`Sphere(int x, int y, int z, int radius)`	$\frac{4}{3} \times \Pi \times radius^3$
مکعب مستطیل (Cuboid)	`Cuboid(int x, int y, int z, int width, int height, int depth)`	$width \times height \times depth$

[1] Inheritance
[2] Polymorphism
[3] Multi-File
[4] Constructor
[5] Volume

مخروط (Cone)	`Cone(int x, int y, int z, int radius, int height)`	$\frac{1}{3} \times \Pi \times radius^2 \times height$

هر یک از این اشکال، باید سه متد زیر را داشته باشند:

نام	امضا[6]	توضیحات
محاسبه‌ی حجم	`int volume()`	حجم شکل را محاسبه می‌کند و باز می‌گرداند.
حرکت	`void move(int dx, int dy, int dz)`	شکل را به اندازه‌های داده شده در هر بعد حرکت می‌دهد .
بزرگ‌نمایی	`void scale(int factor)`	ابعاد شکل را به نسبت ضریب داده شده بزرگ‌نمایی می‌کند.

همچنین شما باید عملگر[7] `>>` را نیز که مخصوص چاپ در خروجی استاندارد[8] است، برای این شکل‌ها سربارگذاری[9] کنید. با صدا زدن این عملگر روی یک شیء از کلاس `Shape`، باید خروجی به فرمت زیر در خروجی استاندارد چاپ شود:

```
type: <t>, center: (<x>, <y>, <z>), volume: <v>
```

که در آن مقدار اول نوع شکل است که یکی از سه مقدار `Sphere` یا `Cuboid` یا `Cone` است. سه مقدار بَعدی مختصات مرکز شکل و مقدار آخر حجم شکل است.

نحوه‌ی آزمون

برای آزمودن برنامه‌ی شما نیازی به دریافت ورودی از جریان ورودی استاندارد[10] نیست؛ بلکه در بدنه‌ی تابع `main`، شکل‌ها ساخته می‌شوند و با استفاده از عملگر سربارگذاری شده برای چاپ در خروجی استاندارد، تست خواهند شد.

کد صفحه‌ی بعد یک مورد آزمون نمونه برای این مسئله است:

[6] Signature
[7] Operator
[8] Standard Output
[9] Overload
[10] Standard Input Stream

```cpp
int main() {
	Shape* shape = new Cuboid(10, 10, -10, 5, 10, 2);
	cout << shape << endl;
	shape->move(-10, 0, 5);
	shape->scale(2);
	cout << shape << endl;
	return 0;
}
```

که خروجی آن به شکل زیر است:

```
type: Cuboid, center: (10, 10, -10), volume: 100
type: Cuboid, center: (0, 10, -5), volume: 800
```

نکات تکمیلی

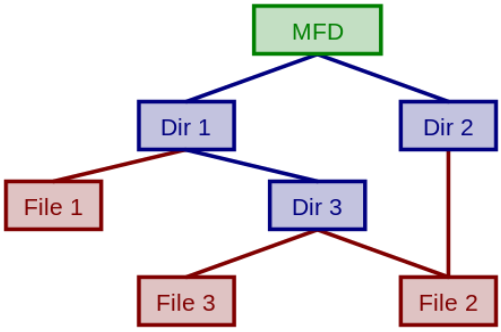
- نمونه‌ای دیگر از ورودی و خروجی این مسئله را می‌توانید در این لینک مشاهده کنید.
- به ثابت[11] بودن شیء پاس داده شده برای سربارگذاری عملگر خواسته شده، دقت کنید.

[11] Const

۲. مدیر پرونده

فایل‌سیستم[12]

در علم کامپیوتر، فایل‌سیستم وظیفه‌ی کنترل کردن نحوه ذخیره‌سازی و بازیابی اطلاعات ذخیره‌شده در حافظه[13] را برعهده دارد. بدون استفاده از فایل‌سیستم، اطلاعات بدون هیچ ساختار خاصی در حافظه ذخیره می‌شدند و امکان بازیابی اطلاعات ذخیره شده وجود نداشت. با جداسازی اطلاعات و اختصاص دادن نام و شناسه‌ای یکتا به هر مجموعه‌ی مرتبط از اطلاعات می‌توان اطلاعات را از یکدیگر تشخیص داد و به‌سادگی بازیابی کرد. در ادامه با برخی مفاهیم اولیه‌ی فایل‌سیستم‌ها آشنا می‌شوید.



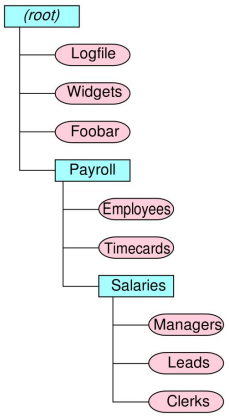
پرونده[14]

پرونده‌ی کامپیوتری[15] یک منبع کامپیوتری[16] برای ذخیره‌سازی اطلاعات به صورت مجزا از یکدیگر در حافظه‌ی دستگاه است. پرونده‌های کامپیوتری را می‌توان برای ذخیره‌سازی انواع تصاویر، ویدئوها، نسخه‌های متنی و انواع بسیار متنوعی از اطلاعات طراحی کرد. با استفاده از برنامه‌های کامپیوتری، می‌توان انواع مختلفی از پرونده‌ها را باز کرد، خواند، تغییر داد، ذخیره کرد و در انتها پرونده را بست.

[12] File System
[13] Storage
[14] File
[15] Computer File
[16] Computer Resource

پوشه[17]

پوشه ساختاری در فایل‌سیستم است که شامل ارجاعاتی[18] به سایر پرونده‌ها یا پوشه‌ها است. در یک فایل‌سیستم سلسله‌مراتبی[19] به پوشه‌ای که داخل پوشه‌ای دیگر قرار دارد، یک *زیرپوشه*[20] گفته می‌شود. اصطلاحات *والد*[21] و *فرزند*[22] برای توصیف رابطه‌ی بین یک پوشه و زیرپوشه‌هایش استفاده می‌شود. یک پوشه ممکن است والد هیچ، یک یا چند زیرپوشه باشد. به بالاترین پوشه‌ای که والد ندارد *ریشه*[23] گفته می‌شود.

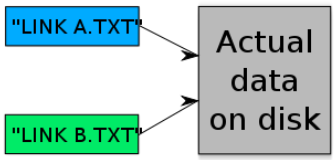


لینک[24]

در یک فایل‌سیستم، می‌توان برای یک پرونده یا پوشه چندین لینک تعریف کرد. از طریق لینک‌های مختلف می‌توان به محتوای یک پرونده یا پوشه دسترسی پیدا کرد و عملیات‌هایی را که برای هر کدام از آن‌ها تعریف می‌شود بر رویشان اعمال کرد. با استفاده از لینک‌ها می‌توان به یک پرونده و یا پوشه، چندین نام نسبت داد که تمام آن‌ها به داده‌ای یکسان درون حافظه اشاره می‌کنند و هیچ‌کدام از لینک‌ها به دیگری وابسته نیستند؛ برای مثال اگر با استفاده از یک لینک در محتوای یک پرونده تغییری داده شود، این تغییرات از طریق لینک دیگری که به این پرونده اشاره دارد و آن را باز کرده است نیز قابل رؤیت خواهد بود.

---

[17] Directory
[18] References
[19] Hierarchical File System
[20] Subdirectory
[21] Parent
[22] Child
[23] Root
[24] Link



شرح تمرین

در این تمرین شما به شبیه‌سازی یک فایل‌سیستم می‌پردازید. در این فایل‌سیستم، امکان اضافه کردن و مشاهده محتوای یک پرونده، پوشه یا لینک وجود دارد. جزئیات توابعی که انتظار می‌رود شما پیاده‌سازی کنید، در ادامه آمده است.

اضافه کردن یک پرونده

در برنامه‌ی شما باید بتوان با فراخوانی[25] تابع زیر، یک پرونده با محتوایی مشخص را به فایل‌سیستم اضافه کرد:

```
void add_file(int id, std::string title, std::string content, int parent_id);
```

پس از فراخوانی تابع فوق، باید یک پرونده مجازی با نامی مشخص و شماره‌ای یکتا به سیستم شما افزوده شود. محتوای پرونده نیز از طریق آرگومان `content` به برنامه شما داده می‌شود.

اضافه کردن یک پوشه

در برنامه‌ی شما باید بتوان با فراخوانی تابع زیر، یک پوشه را به فایل‌سیستم اضافه کرد:

```
void add_directory(int id, std::string title, int parent_id);
```

پس از فراخوانی تابع فوق، یک پوشه‌ی مجازی با نامی مشخص و شماره‌ای یکتا به سیستم شما افزوده می‌شود.

اضافه کردن یک لینک

در برنامه‌ی شما باید بتوان با فراخوانی تابع زیر، لینکی جدید برای یک عنصر بوجود آورد:

```
void add_link(int id, std::string title, int element_id, int parent_id);
```

پس از فراخوانی تابع فوق، یک لینک با شماره‌ای یکتا به عنصری با شناسه‌ی `element_id` بوجود می‌آید. دقت شود که `element_id` باید معرف یک پرونده و یا پوشه باشد.

[25] Call

مشاهده‌ی محتویات یک عنصر

کاربر باید بتواند با فراخوانی تابع زیر، محتویات مربوط به یک عنصر را مشاهده کند:

```
void view(int id);
```

محتویات یک عنصر شامل عنوان عنصر و متناسب با نوع عنصر، شامل موارد زیر باشد:

- **پرونده** محتویاتی که در هنگام اضافه کردن پرونده وارد شده است.

```
Title: <title>
Content:
<content>
```

- **پوشه** نام تمام عناصری که داخل این پوشه قرار دارند.

```
Title: <title>
(For each element in the directory:)
Title: <title>, Type: <type>
```

- **لینک** یکی از موارد فوق، متناسب با نوع عنصری که به آن اشاره می‌کند.

```
Title: <title>
<linked content>
```

مدیریت استثناها[26]

در طول اجرای برنامه، ممکن است انواع خطا در برنامه شما رخ دهد. در این تمرین، انتظار می‌رود شما با پرتاب[27]کردن نمونه[28]هایی از کلاس‌هایی که ادامه ذکر خواهد شد و گرفتن[29] 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[30]ها، خروجی تابع `what` را در جریان خطای استاندارد[31] چاپ کنید.

خطاهایی که انتظار می‌رود که شما در برنامه‌تان به آن‌ها رسیدگی کرده باشید، در ادامه آمده‌اند:

[26] Exception Handling
[27] Throw
[28] Instance
[29] Catch
[30] Exception
[31] Standard Error Stream

- در طول اجرای برنامه اگر تابعی قصد دسترسی به شناسه‌ای غیرمعتبر را داشت، باید نمونه‌ای از کلاس `BadElementId` پرتاب شود و خروجی تابع `what` نیز عبارت زیر باشد:

```
Invalid element ID requested!
```

- اگر هنگام فراخوانی یک تابع، یک عنصر با **شناسه‌ای یکسان** با عنصری دیگر که در حال حاضر در سیستم وجود دارد به وجود آورد، باید نمونه‌ای از کلاس `IdAlreadyExists` پرتاب شود و خروجی تابع `what` نیز عبارت زیر باشد:

```
Requested ID already exists!
```

- نام‌های اختصاص داده شده به عناصر در یک پوشه باید یکتا باشد. بدین ترتیب اگر هنگام فراخوانی یک تابع، یک عنصر با **نامی یکسان** با عنصری دیگر که در حال حاضر در این پوشه وجود دارد به وجود آورد، باید باید نمونه‌ای از کلاس `BadTitle` پرتاب شود و خروجی تابع `what` نیز عبارت زیر باشد:

```
Parent directory has the same child!
```

- اگر در هنگام بوجود آوردن یک عنصر شناسه‌ی والد معرف یک پوشه نبود، باید نمونه‌ای از کلاس `BadParentId` پرتاب شود و خروجی تابع `what` نیز عبارت زیر باشد:

```
Parent ID is not referring to a directory!
```

- اگر در هنگام بوجود آوردن یک لینک، شناسه عنصر معرف یک پوشه و یا پرونده نبود، باید نمونه‌ای از کلاس `BadLinkedElement` پرتاب شود و خروجی تابع `what` نیز عبارت زیر باشد:

```
Invalid element type requested!
```

نکات تکمیلی

- در تمام توابعی که در بالا ذکر شد، `parent_id` معرف یک پوشه می‌باشد.
- در ابتدای برنامه، به صورت پیش‌فرض باید پوشه‌ی ریشه با **شناسه‌ی صفر** در فایل‌سیستم شما وجود داشته باشد و والد آن نیز دارای شناسه‌ای برابر با **صفر** باشد.
- کلاس رابطی که شما در برنامه‌تان به پیاده‌سازی توابع آن می‌پردازید و همچنین نمونه‌هایی از تابع اصلی که در کنار پرونده‌های برنامه شما قرار می‌گیرد و برنامه‌ی شما با این تابع ترجمه[32] می‌شود از لینک قابل دسترس است.

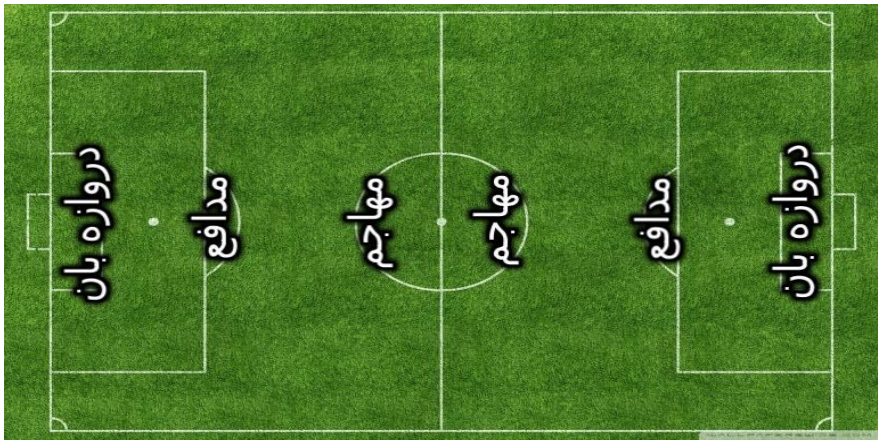
[32] Compile

۳. مسابقه‌ی فوتبال (امتیازی)

در این تمرین شما به شبیه‌سازی یک مسابقه‌ی فوتبال ساده‌شده می‌پردازید.

توصیف بازیکنان و زمین بازی

در این بازی دو تیم در مقابل یکدیگر قرار می‌گیرند که هر تیم را سه بازیکنِ دروازه‌بان[33]، مدافع[34] و مهاجم[35] تشکیل می‌دهند. حالت شروع بازی در شکل زیر آمده است:



بازیکنان توانایی[36]های پاس[37]، دفاع[38] و دریبل[39] را دارند. علاوه بر این هر بازیکن یک معیار استقامت[40] نیز دارد که بر توانایی‌هایش تأثیر می‌گذارد. هر کدام از توانایی‌ها و استقامت بازیکنان با یک عدد صحیح[41] بین ۰ تا ۱۰۰ مشخص می‌شود.

[33] Goalkeeper
[34] Defender
[35] Striker
[36] Ability
[37] Pass
[38] Defend
[39] Dribble
[40] Stamina
[41] Integer

حرکات بازیکنان

در ابتدا توپ دست بازیکن مهاجم تیم میزبان است. سپس در هر مرحله بازیکنی که توپ را در اختیار دارد یکی از حرکت‌های زیر را انجام می‌دهد:

- اگر بازیکن دروازه‌بان باشد:
  - توپ را به مدافع پاس می‌دهد.
- اگر بازیکن مدافع باشد:
  - توپ را به مهاجم پاس می‌دهد.
- اگر بازیکن مهاجم باشد:
  - باید از بازیکن حریفی که در مقابلش قرار دارد عبور کند. عبور از دروازه‌بان به منزله‌ی زدن گل است.

تقابل

هر دو بازیکن رقیب را که مقابل هم قرار می‌گیرند یک تقابل می‌نامیم. در واقع در هر مرحله از بازی یک تقابل رخ داده و با توجه به نتیجه آن تقابل بعدی رخ می‌دهد و....

تقابل ممکن است در حالت های زیر رخ دهد:

- بین بازیکنی که پاس می‌دهد (دروازه‌بان یا مدافع) و بازیکنی که پاس را دریافت می‌کند، بازیکنی از تیم حریف وجود داشته باشد.
  - در این حالت طبق جدول تقابل، بازیکنی که پاس می‌دهد اگر
    - برنده شود، پاس موفقیت آمیز خواهد بود و توپ به بازیکن هم تیمی می‌رسد.
    - برنده نشود، توپ به دست بازیکن حریفی که به بین این دو بازیکن بوده می‌افتد. در هر حال بازیکنی که توپ را در اختیار می گیرد به سراغ تقابل بعدی خود می‌رود.
- مهاجم بخواهد از مهاجم، مدافع یا دروازه‌بان تیم حریف عبور کند.

این حالت نیز مانند حالت قبل است. با این تفاوت که در صورت برنده شدن خود مهاجم همچنان توپ را در اختیار دارد و به سراغ تقابل بعدی می‌رود. (در حالت شکست نیز همانند حالت قبل است.

جدول تقابل ها را در ادامه مشاهده می‌کنید. فرض کنید توپ در اختیار بازیکن تیم A است.

	دروازه‌بان تیم A	مدافع تیم A	مهاجم تیم A
دروازه‌بان تیم B	✖	✖	$A_{dribble} > \frac{4 \times B_{defence}}{3}$
مدافع تیم B	✖	$A_{pass} > B_{defence} + 10$	$A_{dribble} > B_{defence}$
مهاجم تیم B	$A_{pass} > \frac{B_{defence}}{2}$	$A_{pass} > B_{defence} + 15$	$A_{dribble} > B_{defence} + 5$

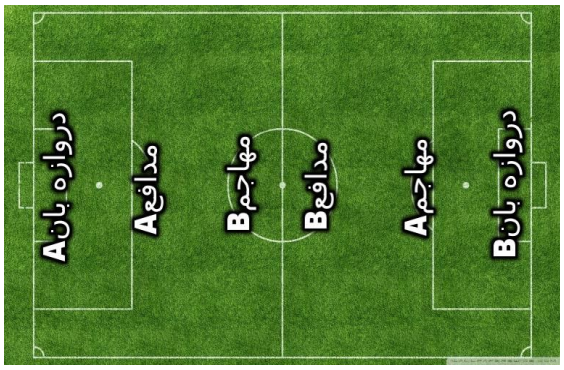
دقت کنید در هنگام پاس دادن اگر تقابل رخ ندهد (بازیکنی از تیم حریف بین بازیکنان وجود نداشته باشد) پاس به‌درستی به مقصد خواهد رسید.

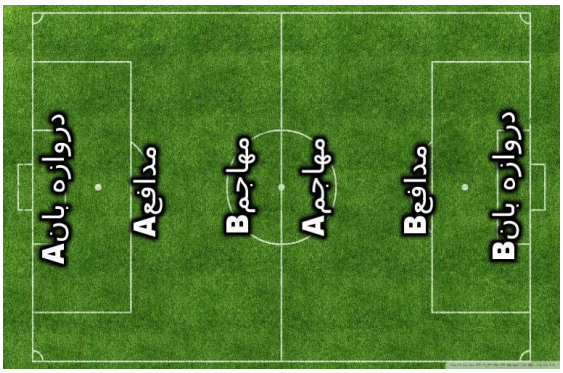
نحوه‌ی محاسبه‌ی استقامت

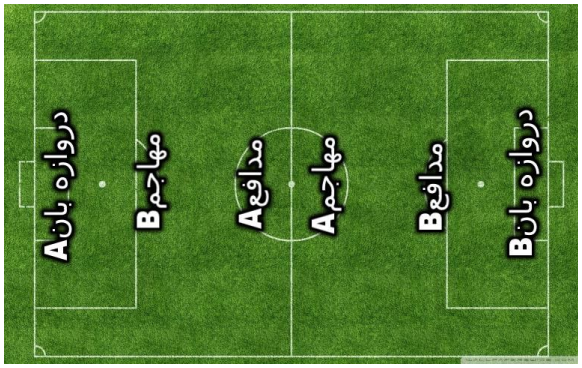
در ابتدا استقامت همه‌ی بازیکنان ۱۰۰ واحد است. در هر مرحله تقابل از استقامت بازیکنی که در تقابل پیروز شده به دلیل خستگی ۱۰ واحد کم می‌شود. میزان استقامت بازیکنان تأثیر مستقیمی بر توانایی های آنان دارد و به هر میزان که از استقامت آنان کم شود، توانایی‌های آنان تقلیل می‌یابد؛ یعنی بعد از هر تقابل باید هر ۳ توانایی بازیکن برنده را ۱۰ واحد کمتر در نظر بگیرید. توانایی بازیکنان از صفر کمتر نخواهد بود؛ در نتیجه، اگر به واسطه‌ی استقامت یکی از توانایی های بازیکنان کمتر از صفر شد آن را صفر در نظر بگیرید.

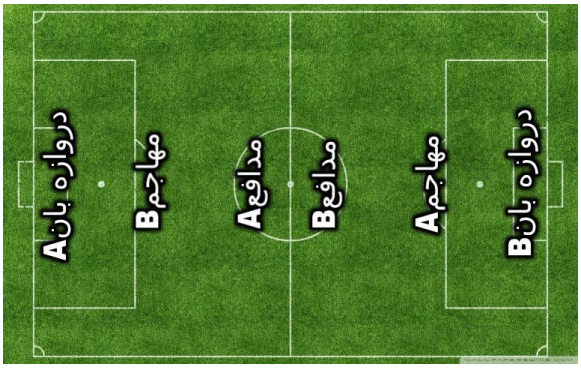
جابجایی بازیکنان

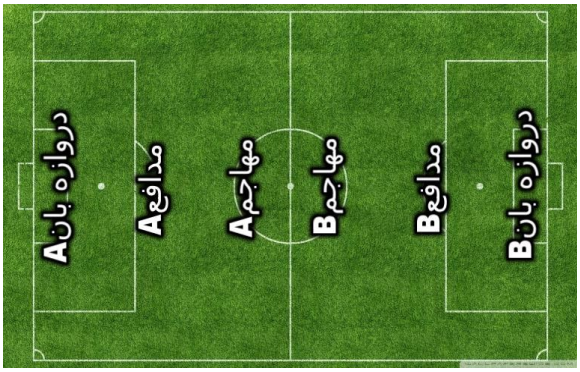
آرایش اولیه‌ی بازیکنان همان‌طور است که در بخش‌های قبلی گفته شد. وقتی مهاجمی به مدافع حریف می‌رسد و موفق می‌شود از آن عبور کند (دریبل بزند) از مدافع عبور کرده و مکان آن با مکان مدافع عوض می شود و تا پایان بازی نیز در همانجا قرار می‌گیرد.
در مورد رویارویی دو مهاجم حریف نیز همین نکته صادق است.
برای توضیح بیشتر، دقت کنید که آرایش بازیکنان همواره به یکی از ۵ حالت صفحه‌ی بعد است:











در واقع، مکان دروازه‌بان‌ها همیشه ثابت است و مهاجم ها پس از هربار دریبل از بازیکن روبرویی خود عبور می کنند و هیچگاه هم رو به عقب بر نمی‌گردند.

اتمام بازی

وقتی که مهاجم یکی از تیم ها موفق به زدن گل شود (از دروازه‌بان عبور کند) آن تیم به عنوان برنده مسابقه اعلام می‌شود.

در حالتی که توپ ۵ بار میان بازیکنان دو تیم جابه‌جا شود و گلی حاصل نشود مساوی اعلام می‌شود.

نحوه‌ی آزمون

برای این تمرین، یک رابط[42] در اختیار شما قرار داده شده است که درستی برنامه‌ی شما از طریق فراخوانی تابع

```
void get_result();
```

این رابط سنجیده می‌شود. خروجی این تابع باید یکی از خروجی‌های زیر باشد:

```
result: team B wins
result: team A wins
result: draw!
```

به فرمت خروجی دقت کنید و دقیقاً همانند آن را برگردانید.

برای افزودن بازیکن نیز از متدهایی که در رابط تعریف شده‌اند استفاده می‌شود. این متدها عبارتند از:

```
void add_team_A_goalkeeper();
void add_team_A_defender();
void add_team_A_striker();
void add_team_B_goalkeeper();
void add_team_B_defender();
void add_team_B_striker();
```

[42] Interface

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.
- 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 میک‌فایل 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[43] برنامه‌ی شما با توابع اصلی آزمون[44] 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[45]، می‌توانید از این لینک استفاده کنید.
- **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 شکل در بخش اوّل، نوع عنصر در بخش دوم و نوع بازیکنان در بخش سوم 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.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت کشف تقلب مطابق قوانین درس با آن برخورد خواهد شد.
- 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.

[43] Main
[44] Test
[45] Application Programming Interface