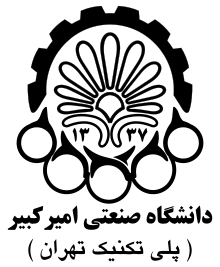
**بسم الله الرحمن الرحیم**



**گزارش تمرین سری چهارم**

**برنامه سازی پیشرفته**

**سیده معصومه سجادی**

**ش.د:۹۳۲۳۰۸۷**

اردیبهشت 1398

سوالات C++

سوال 1:

* 1. Dive deeper into C++ with move semantics, a key component of C++ resource management. Move semantics allow you to optimize the transfer of data and reallocate memory without extra copy operations. In this course, Bill Weinman helps you understand rvalues (temporary values that are the key to move semantics); create move constructors and operations; and use the copy-and-swap idiom to "swap in" new data while destroying the old.
  2. The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.
  3. pure abstract . کلاس ارث دهنده ای با وضعیت pure abstract ، دو ویژگی خواهد داشت:

**ویژگی اول:** امکان تعریف تابع pure abstract آن داخل خودش نیست.

**ویژگی دوم:** تمامی کلاس های ارث برنده، موظف به تعریف آن تابع می باشند.

* 1. override . با قرار دادن این عبارت جلوی توابع مشترک بین کلاس ارث دهنده و کلاس ارث برنده، کامپایلر کاربر را موظف می کند که آن تابع را در کلاس ارث برنده بازتعریف کند. در نتیجه کاربرد این عبارت، در ارث بری کلاس ها می باشد.
  2. inline . با نوشتن عبارت inline در ابتدای تابع، کامپایلر کد تابع را عینا در محل فراخوانی شدن آن قرار می دهد، بدون اینکه نیازی به انجام مراحل واقعی فراخوانی تابع و پروسه های مربوط به آن باشد. این کار منجر به افزایش سرعت کد می شود، ولی در عین حال، حجم فایل اجرایی را نیز افزایش می دهد. به همین دلیل استفاده از inline در توابع کوچک بسیار مفید و کاربردی می باشد، ولی استفاده از آن در توابع بزرگتر ممکن است غیرمفید و یا حتی امکان ناپذیر باشد.
  3. . explicit قرار دادن عبارت explicit در ابتدای تعریف تابع، منجر به جلوگیری از فراخوانی تابع به صورت ضمنی می گردد. به عبارت دیگر، تابع تنها زمانی فراخوانی می شود که ورودی تعریف شده برای آن، با ورودی اعمال شده به آن عینا یکسان باشد و تابع به طور صریح فراخوانی شده باشد.

سوال ۲:

سوال ۳ و سوال ۴:

در این سوال قصد داریم کلاس Shape را به صورت خلاصه و به عنوان کلاس پایه تعریف کنیم. در مرحله بعد دو کلاس TwoDimensionalShape و ThreeDimensionalShape را به ترتیب برای اشکال دو بعدی و اشکال سه بعدی که از کلاس پایه Shape ارث می برند تعریف می نماییم. در مرحله بعد دو کلاس Circle و Square را به عنوان ارث برنده از کلاس TwoDimensionalShape و دو کلاس Sphere و Cube را به عنوان ارث برنده از کلاس ThreeDimensionalShape پیاده سازی می کنیم. در نهایت همه کلاس های پیاده سازی شده باید تابع main داده شده را کاملا دقیق اجرا کنند و خروجی مورد نظر باید مثل خروجی داده شده در صورت سوال باشد.

در کلاس هایی که از کلاس مبنا ارث می برند توابع area() و volume() که محاسبات مساحت و حجم را برای اشکال انجام می دهند تعریف می کنیم و توجه می کنیم که برای اشکال دو بعدی حجم بی معنی خواهد بود. این توابع را به صورت virtual در کلاس ها استفاده می نماییم. در واقع در prototype توابعی که قرار است روی آنها late binding انجام شود عبارت virtual قرار می دهیم. اصطلاح late binding به این معناست که فرض کنید چند تابع با یک نام در کلاس های مختلف وجود دارد و در تابع اصلی main این تابع با نام مورد نظر روی یک شیء می خواهد اعمال شود. کامپایلر هنگامی که به این خط از کد می رسد باید تشخیص دهد که شیء مورد نظر مربوط به کدام یک از کلاس های ارث دهنده یا ارث برنده است، سپس تابع اعمال شده را از همان کلاس برای اجرا روی شیء فراخوانی کند نه توابع با همان نام از کلاس های دیگر که این حالت در زمان run time توسط کامپایلر انجام می شود. این دقیقا یکی از مصداق های کلاس های polymorphism می باشد. طبق همین تعریف چون مساحت و حجم هر یک از اشکال با روابط مخصوص به خود به دست می آید بنابراین باید این توابع را در همه کلاس ها تعریف کرده و از عبارت virtual در prototype آنها در فایل های header استفاده کنیم. همچنین چون خروجی های مختلفی برای هر یک از اشکال داریم روی توابع print هم late binding انجام داده و عبارت virtual را در آنها به کار می بریم.

در قسمت پایانی تابع main یک حلقه for داده شده که برای 4 شیء ساخته شده باید 4 خروجی مربوط به مشخصات اشکال را چاپ کند. همه چیز باید با یک خط کد cout << \*( ptr[ x ] ) << '\n'; چاپ شود. بنابراین باید اپراتور << را به گونه ای تعریف کنیم که دو ورودی cout به صورت رفرنسی از کلاس std::ostram و دیگری \*( ptr[ x ] ) به صورت رفرنسی از کلاس Shape ( یک شیء) بگیرد و رفرنس گرفته شده از کلاس std::ostram را به عنوان ورودی تابع print در هر یک از کلاس ها قرار دهد و درون خود اپراتور این تابع را روی همان شیء گرفته شده اعمال کند و در نهایت یک رفرنس از کلاس std::ostram برگرداند که منجر به خروجی های مورد نظر شود. این اپراتور را فقط در کلاس پایه Shape تعریف می نماییم.

در تمرین قبلی کلاس نقطه را تعریف کرده و پیاده سازی نمودیم. در مرحله آخر اپراتور جمع را برای کلاس های ارث برنده طوری به کار می بریم که با جمع عنصری از کلاس نقطه و کلاس های مشتق شده بتواند مرکز اشکال را به اندازه مختصات داده شده برای تست جابجا نماید. این بخش به صورت یک حلقه for در آخر تابع main اضافه شده است که یکی یکی مرکز اولیه اشکال ساخته شده را می گیرد با مختصات یک نقطه که برای تست داده شده (3 و 2) جمع می کند و مراکز جدید اشکال را در خروجی نمایش می دهد.

سوال ۵:

سوال ۶:

سوال ۷: