نام و نام خانوادگی دانشجو: محمدامین احمدی رشته :کامپیوتر شماره دانشجویی: 9912358001

نام استاد : خانم بطحائیان

موضوع : فاز دوم پروژه کشاورز


بسم الله الرحمن الرحيم

فهرست مطالب

شرح پروژه

با اینکه اطلاعات کمی برای پروژه داده شده بود ولی خب به برخی موارد از قبیل :

پیاده سازی دو کلاس

استفاده از:

const

set

get

constructor

operator overloading

برای فاز اول پروژه اشاره شده بود.هرچند که درباره ی مورد آخر من شخصا از استاد پرسیدم و ایشان گفتند که به علت این که این مبحث رو تدریس نکردند احتیاجی به پیاده سازی این بخش نیست.

برای فاز دوم پروژه به تکمیل فاز اول پرداخته شده: اضافه کردن یک کلاس و به کارگیری وراثت و پولیمورفیسم درآن.

لازم به ذکر است که از operator overloading در این فاز پیاده سازی شده.

مقدمه

در این فاز از پروژه من صرفا دوکلاس را با موارد خواسته شده پیاده سازی کردم .

این برنامه شامل کلاس هاییست که در آن یک کلاس به اطلاعات شخص کشاورز اعم از نام و نام خوانوادگی و سن او، و یک کلاس به نوع وسیله ی کشاورزی و ونوع بذر پرداخته اند.

کلاس اول:

Farmer

کلاس بعدی:

AgricultureTools

کلاس سوم:

Tractor

و همچنین یک ماژول برای تست کلاس که به ترتیب موارد پیاده سازی شده در کلاس ها و همچنین operator overloading را که پیاده سازی آن به صورت غیر عضو از کلاس ها صورت گرفته را تست میکند.

با پیاده سازی فاز های بعدی و مرج شدن آن ها با این قسمت خب بالطبع تغییراتی در تست کلاس و همچنین توابع کلاس ها حاصل میشود.

class Farmer

کلاس فارمر شامل توابعیست که هریک از این توابع به شکل خاصی به بهبود برنامه کمک میکنند که در ادامه به آنها میپردازیم:

اما قبل از آن به سراغ متغیر های پرایوت این کلاس میرویم یعنی:

1) دو متغیر از نوع اشاره گر کاراکتری جهت ذخیره سازی نام و نام خانوادگی کشاورز

```
string name
string lastName
```

2) یک متغیر جهت ذخیره سازی سن کشاورز

```
int age
```

3) متغیری از نوع کلاس AgricultureTools که در ادامه با کاربرد آن آشنا میشویم

```
AgricultureTools agricultureTools
```

حال میرویم به سراغ توابع این کلاس:

1) دو تابع جهت برگشت دادن نام و نام خانوادگی که همانطور که میبینیم از دو کانست برای آن ها استفاده شده که اولی برای نوع مقدار برگشتی است و دومی برای تعریف تابع که باعث میشود که که دستکاری متغیر های پرایوت از طریق آن امکان پذیر نباشد.

```
const string getName() const;
const string getLastName() const;
```

2) کانستراکتور کلاس فارمر که هر یک از متغیر های پرایوت را مقداردهی می کند.

```
Farmer(string name, string lastName, int age, const AgricultureTools &agricultureTools);
```

3) وجود دوتابع اولی برای مقداردهی و دومی برای برگشت دادن مقدار از متغیر پرایوت age :

توجه کنید وجود const در تابع get به این معنیست که این تابع حق ایجاد تغییر در متغیرهای پرایوت را ندارد.

```
void setAge(int age);
```

int getAge() const;

4) دوتابع یکی برای مقداردهی و دیگر برگشت دادن اطلاعات از متغیری از جنس کلاس AgricultureTools :

درواقع ما با پارامتر قرار دادن یک رفرنس از کلاس AgricultureTools برای تابع setAgricultureTools که مقداری را برگشت نمیدهد و در کلاس Farmer تعریف شده است، میتوانیم به اطلاعات موجود در متغیرهای کلاس AgricultureTools از طریق متغیر agricultureTools در کلاس Farmer دسترسی داشته باشیم .

در تابع getAgricultureTools که درواقع برا برگرداندن اطلاعات نوشته شده ما با استفاده ی درست از این تابع در تست کلاس میتوانیم اطلاعات هریک از متغیر های کلاس AgricultureTools را نمایش دهیم که در ادامه با هم میبینیم.

void setAgricultureTools(const AgricultureTools &agricultureTools);

const AgricultureTools &getAgricultureTools() const;

class AgricultureTools

در کلاس AgricultureTools همانطور که پیش تر اشاره شد بیشتر با مسائل مربوط به کشاورزی سر و کار داریم باهم میبینیم.

متغیر های پرایوت کلاس:

دو متغیر از نوع اشاره گر کاراکتری جهت ذخیره سازی نوع ابزار کشاورزی و نوع بذر مورد نظر کشاورز.

```
string tools;
string bazr ;
```

توابع عضو کلاس:

1) دوتابع جهت تنظیم اطلاعات متغیر پرایوت tools، اولی برای گرفتن مقدار و دومی برای برگرداندن مقدار جهت نمایش.

```
void setTools(string tools)
char *getTools() const;
```

2) دوتابع جهت تنظیم اطلاعات متغیر پرایوت bazr، اولی برای گرفتن مقدار و دومی برای برگرداندن مقدار جهت نمایش.

```
void setBazr(string bazr);
char *getBazr() const;
```

3) و اما نوبتی هم که باشد نوبت کانستراکتور این کلاس است که به طور مستقیم متغیرهای پرایوت کلاس را مقدار دهی اولیه می کند.

```
AgricultureTools(string tools, string  bazr);
```

4) یک تابع برای پرینت اطلاعات به طور دقیق:

```
virtual void print() const;
```

کلمه ی virtual در اینجا به این خاطر است که کلاس فرزند بتواند از از این تابع در بحث پلیمورفیسم استفاده کند.

class Tractor

میرسیم به کلاس تراکتور که درواقع فرزند کلاس AgricultureTools محسوب می شود. این تراکتور از ابزار و امور مربوط به کشاورزی ارث میبرد در عین حال ویژگی های به خصوص خود را نیز دارا میباشد.

این نوع از تراکتور با تراکتور های معمول متفاوت بوده و کار آن شخم زدن نیست بلکه هدفش استفاده از المان های کلاس AgricultureTools به نحو احسن میباشد و در این راه به کشاورز کمک می کند.

این تراکتور با توجه به نوع وسیله ای که در اختیار دارد اقدام به کاشت بذری میکند که در اختیارش قرار داده شده.

متغیر های پرایوت کلاس:

```
float weight;

float power;
```

و همچنین قابل توجه است متغیر های پرایوت کلاس پدر در فرزند پنهان است و فرزند اجازه ی دسترسی مستقیم به آن را ندارد.

توابع عضو کلاس:

1) دو تابع جهت تنظیم قدرت تکراکتور اولی برای تعیین و دومی برای نمایش :

```
float getPower() const;

void setPower(float power);
```

2) دو تابع جهت تنظیم وزن تکراکتور اولی برای تعیین و دومی برای نمایش :

```
float getWeight() const;

void setWeight(float weight);
```

3) کانستراکتور کلاسTractor که میتوانیم در پروتوتایپ آن دو متغیر bazr و tools را هم ببینم. همانطور که بالاتر گفته شد کلاس فرزند اجازه ی دسترسی مستقیم به متغیر های پرایوت کلاس پدر را ندارد پس تکلیف چیست؟

قضیه در پیاده سازی این کانستراکتور طور دیگریست :

```
Tractor::Tractor(string tools, string bazr, float weigth, float power) : AgricultureTools(tools, bazr){

  this->weight = weigth;

  this->power = power;}
```

همانطور که میبینید ما برای دستکاری متغیر های tools و bazr کانستراکتور کلاس AgricultureTools (پدر) را در کلاس Tractor فراخوانی کردیم.

4) یک تابع برای پرینت اطلاعات :

```
void print() const;
```

که به صورت کاملا مشخص به چاپ اطلاعات کلاس Tractor به صورت مستقیم و همچنین اطلاعات کلاس پدر از طریق دستور AgricultureTools::print(); می پردازد.

ماژول تست کلاس ها

هدر:

```
#include <iostream>
#include "Farmer.h"
#include "AgricultureTools.h"
#include "Tractor.h"
```

:input operator overloading

```
istream &operator>>(istream & input, Farmer & f)
{   cout << "say my name" << endl;
    input >> f.name;
    cout << "say my Lastname" << endl;
    input >> f.lastName;
    cout << "say my age" << endl;
    input >> f.age;
    return input;
}
```

با پیاده سازی این کد ما در واقع عملگر << را سربار گذاری میکنیم به نحوی که باعث میشویم که کامپایلر بعد از دیدن دستور:

```
cin >> *farmer;
```

رفتار مخصوصی را از خود نشان دهد که نتیجه ی آن گرفتن اطلاعات شخص کشاورز از ورودی میباشد.

همچنین این تابع بع عنوان دوست کلاس Farmer تعریف شده که مشکلی از لحاظ دسترسی مستقیم به متغیر های پرایوت کلاس Farmer وجود نداشته باشد:

```
friend std::istream &operator>>(std::istream & , Farmer & );
```

:output operator overloading

```
ostream &operator<<(ostream & out, Farmer & f)
{     cout <<"Name: " << f.name << endl;
    cout <<"last name: " << f.lastName << endl;
    cout <<"age: " << f.age << endl;
    return out;
}
```

با پیاده سازی این کد ما در واقع عملگر >> را سربار گذاری میکنیم به نحوی که باعث میشویم که کامپایلر بعد از دیدن دستور:

```
cout << *farmer;
```

رفتار مخصوصی را از خود نشان دهد که نتیجه ی آن نمایش شخص کشاورز در خروجی میباشد.

همچنین این تابع به عنوان دوست کلاس Farmer تعریف شده که مشکلی از لحاظ دسترسی مستقیم به متغیر های پرایوت کلاس Farmer وجود نداشته باشد:

```
friend std::ostream &operator<<(std::ostream & , Farmer & );
```

استفاده از دستور : using namespace std; برای راحتی کار.

تابع main:

در ابتدا ما باید برای هر کدام از این کلاس ها شی تعریف کنیم تا به ویژگی های آن موجودیت ببخشیم:

```
AgricultureTools *agricultureTools = new AgricultureTools("bil" , "wheat");
Farmer *farmer = new Farmer("AMIN" , "AHMADI " , 32,*agricultureTools);
```

این اشیا به صورت پوینتری تعریف شده اند که بتوان برای آن ها با استفاده از دستور new حافظه ی پویا در نظر گرفت.

لازم به ذکر است که عدم تخصیص حافظه ی پویا در این قسمت مشکلی به وجود نمی آورد (حداقل در این فاز).

برای هر کانستراکتور از این کلاس مقادیری به صورت دستی در نظر گرفته شده تا از صحت کارکرد برنامه اطمینان حاصل شود.

توجه داشته باشید که شی agricultureTools زود تر ساخته میشود تا بتوان برای ساخت شی farmer آن را به کانستراکتور کلاس Farmer پاس داد.

سپس به نمایش اطلاعات از طریق فراخوانی درست توابع میپردازیم:

1) نمایش نام و نام خانوادگی و سن کشاورز به ترتیب.

```
cout<<farmer->getName()<<"\n";
cout<<farmer->getLastName()<<"\n";
cout<<farmer->getAge()<<"\n";
```

نکته: بدیهی است که چون شی farmer به صورت پوینتری تعریف شده برای دسترسی به توابع عضو آن میبایست از ’->‘ به جای ’ . ‘ استفاده کنیم.

2) نمایش نوع ابزار و نوع بذر مورد استفاده کشاورز.

```
cout<<farmer->getAgricultureTools().getTools()<<"\n";
cout<<farmer->getAgricultureTools().getBazr()<<"\n";
agricultureTools->print();
```

نکته : میدانیم که نوع ابزار و نوع بذر در کلاس AgricultureTools ذخیره شده . همانطور که توضیح داده شد ما از طریق کلاس Farmer میتوانیم به این اطلاعات از طریق فرا خوانی تابع ()getAgricultureTools دسترسی داشته باشیم اما لازم است که مشخص کنیم که دقیقا به کدام یک از دیتا ها احتیاج داریم پس ما بعد از تابع نام برده شده از یک ' . ' استفاده میکنیم (چون درواقع یک رفرنس از کلاس اِگریکالچر به کلاس فارمر بازگردانده شده) و بعد از آن نام تابع مناسبی را که دیتای مورد نظر مارا باز میگرداند را فراخوانی میکنیم که دراینجا یا ()getTools است و یا ()getBazr .

تست کلاس Tractor و ... :

در ابتدا ما باید برای این کلاس شی تعریف کنیم تا به ویژگی های آن موجودیت ببخشیم:

```
Tractor *tractor = new Tractor("kolang", "jo", 800.5, 15.6);
```

سپس به نمایش اطلاعات از طریق فراخوانی درست توابع میپردازیم:

```
cout<<tractor->getTools()<<"\n";
cout<<tractor->getBazr()<<"\n";
cout<<tractor->getWeight()<<"\n";
cout<<tractor->getPower()<<"\n";
tractor->print();
```

حال به سراغ استفاده از تابع print برای تست پولیمورفیسم میرویم:

یک اسم مستعار برای agricculturetools به نام vasayel تعریف میکنیم و تابع print را برای آن فراخوانی میکنیم.طبیعتا مشاهده میکنیم که همان خروجی دستور ;()agricultureTools->print نمایش داده میشود.

همین کار ار برای tractor انجام میدهیم با اسم مستعار tirakhtoor که همان خروجی دستور ;()tractor->print نمایان میشود.

اگر ما بیایم vasayel را این بار اسم مستعاری برای tractor قرار دهیم انتظار داریم که خروجی مربوط به دستور

tractor->print(); را دوباره مشاهده کنیم اما اینطور نیست بلکه خروجی دستور vasayel->print(); که به صورت پیش فرض در vasayel ذخیره شده.

برای جلوگیری از این امر و دست یافتن به خروجی مناسب ما از پولیمورفیسم کمک میگیریم و کلمه ی virtual را قبل از تعریف تابع print پدر قرار میدهیم تا تابع پرینت کلاس فرزند هم بتواند از آن به شکل درست بهره ببرد:

```
virtual void print() const;
```

ذکر یک نکته:

- agriculturetools شاید به صورت لغوی به معنای ابزار کشاورزی باشد اما به صورت منطقی میتواند به تمامی امور،اشیا و لوازمی که در عمل کشاورزی تاثیر دارند اشاره کند.

در پناه خدا

شاد و پیروز باشید