فاز دوم پروژه درس برنامه سازی پیشرفته

پروژه جراح

سینا عبداللهی یگانه

9712358030

Inheritance

به توسعه دهنده این امکان را می دهد تا کد مورد نظر را یکبار داخل کلاس پدر نوشته، سپس آن را هم در کلاس پدر و هم در کلاس یا کلاس های فرزند مورد استفاده قرار داد.

به توسعه دهنده این امکان را می دهد تا کد مورد نظر را Inheritance یکبار داخل کلاس پدر نوشته، سپس آن را هم در کلاس پدر و هم در کلاس یا کلاس های فرزند مورد استفاده قرار داد

مقدمه

جراحان از ابزار های مختلفی برای انجام عمل جراحی استفاده می کنند که دو تا از معروف ترین آن ها قیچی و انبر هستند، قیچی برای بریدن رگ ها و ق سمت های بدن ا ستفاده می شود و انبر برای باز نگه دا شتن بریدگی و همچنین بردا شتن ج سمی از داخل بدن استفاده می شوند

شرح : سه مورد از قیچی های جراحی که استفاده می شود در تصویر زیر در خط اول مشاهده می شود، نوع قیچی ازمواردی است که هر قیچی باید آن را داشته باشد و ویژگی بعدی طول قیچی به سانتی متر و راست یا چپ دست بودن آن می باشد. موارد دیگر نیز نوع بدنه قیچی که استیل یا

تنگستن کربید می باشد و اشیایی می باشد که می تواند ببرد می باشد.

ارث بری:

از کلاس (property) کلاس فرزند می تواند تمامی متدها و فیلد هایی تعریف نشده به ارث برده و مورد private پدر را که با سطح دسترسی استفاده قرار دهد. این امکان توسعه دهنده را قادر می سازد تا کد را یکبار در کلاس سطح پدر نوشته و سپس آن را بارها در کلاس هایی که از آن ارث بری دارند، مورد استفاده قرار دهد (هم در کلاس پدر و هم در کلاس های فرزند).

ارث بری دارد، بنابراین می Car از کلاس SportsCarدر مثال زیر، نشانه private هایی که با کلیدواژه ی property تواند به تمامی متدها و گذاری نشده اند، دسترسی داشته باشد. بدین وسیله توسعه دهنده متدهای تعریف شده اند را یکبار در public که به صورت ()hello و ()setModel کلاس پدر پیاده سازی کرده و سپس این متدها را هم در کلاس پدر و هم در کلاس های فرزند استفاده می کند.

```
//The parent class
class Car {
  // Private property inside the class
  private $model;
  //Public setter method
  public function setModel($model)
  {
    $this -> model = $model;
  }
  public function hello()
  {
```

```
    return "beep! I am a <i>" . $this ->
model . "</i><br>";
  }
}
//The child class inherits the code from
the parent class
class SportsCar : Car {
  //No code in the child class
}
//Create an instance from the child class
$sportsCar1 = new SportsCar();
// Set the value of the class' property.
// For this aim, we use a method that we
created in the parent
$sportsCar1 -> setModel('Mercedes Benz');
//Use another method that the child class
inherited from the parent class
echo $sportsCar1 -> hello();
```

.

خروجی:

?

1 beep! I am a Mercedes Benz

include <iostream>#

using namespace std;

class Person

}

public:

string profession;

```
int age;
{ } (16Person(): profession("unemployed"), age(
void display()
}
cout << "My profession is: " << profession <<
endl;
cout << "My length is: " << age << endl;
castroviejo scissors();
vannas scissors. ();
{
void walk() { cout << "castroviejo scissors" <<
endl; }
void talk() { cout << "vannas scissors." << endl; }
;{
Length class is derived from base class Person.  //
class Docter : public Person
}
public:
void Docter() { cout << "I can take length." <<
endl; }
;{
```

```
Docter class is derived from base class Person. //
class docter : public Person
}
public:
void search doctor() { cout << "I can play Docter."
<< endl; }
;{
int main()
}
Docter length;
Docter = "length";
;23Say length =
Docter.dis say();

Doctor,person();
Doctor, length;
Doctor.profession = "doctor";

;0return
}
```

Polymorphism Cplus

چندریختی

با **توابع مجازی** و **چند ریختی** میتوان سیستم هایی طراحی و پیاده سازی کرد که به آسانی قابل توسعه باشند. چندریختی هم در زمان ترجمه و هم در زمان اجرا در C++ امکان پذیر است. **چندریختی زمان ترجمه**با تعریف مجدد توابع و عملگرها صورت میگیرد و **چندریختی زمان اجرا**، با استفاده از وراثت و توابع مجازی انجام میشود.

تابع مجازی یک تابع عضو است که در یک کلاس **پایه** اعلان میشود و دوباره در کلاس **مشتق** تعریف میگردد. کلاس هایی از اشیا مانند circle، triangle، rectengle، squre و غیره را که همگی از کلاس shapeمشتق شده اند در نظر بگیرید. در برنامه نویسی شی گرا میتوان قابلیتی را به هر کدام از این کلاس ها اضافه کرد که خودشان را رسم کنند. تابعی که شکل را رسم میکند draw می نامیم. گرچه هر یک از این کلاس ها، تابع draw مخصوص به خود دارند، ولی این تابع برای هر شکل متفاوت است. هنگام رسم شکل اگر بتوان با همه آنها مثل اشیایی از کلاس shape برخورد کرد، بسیار عالی خواهد بود. در این صورت برای رسم شکل میتوان تابع draw از کلاس پایه shape را فراخوانی کرد و به برنامه اجازه داد خودش تشخیص دهد که از کدام تابع draw از کلاس مشتق استفاده کند. برای انجام چنین کاری تابع draw را در کلاس پایه بصورت **virtual**(**تابع مجازی**) تعریف کرده هر یک از توابع موجود در کلاس های مشتق را با اعضای جدیدی می نویسیم تا شکل مناسب را رسم کند. برای اعلان یک تابع مجازی کافیست در جلوی نام آن کلمه virtual را ذکر کرد.

وقتی کلاسی از یک کلاس پایه با تابع مجازی مشتق میشود، کلاس مشتق، آن تابع را طوری در خودش تعریف میکند که

نیازهای خودش را برطرف کند. بطور کلی توابع مجازی فلسفه ” **یک واسط چند متد**” را که مربوط به چندریختی است پیاده سازی میکند.

هنگامیکه تابع مجازی بطور عادی دستیابی میشود، همچون سایر توابع عضو عمل میکند، اما آنچه تابع مجازی را با اهمیت جلوه میدهد و موجب پشتیبانی چندریختی در زمان اجرا میشود، چگونگی رفتار آنها در هنگام دسترسی از طریق اشاره گر است. اشاره گر کلاس پایه میتواند برای اشاره کردن به هر شیءی از هر کلاسی که از آن مشتق میشود، به کار گرفته شود. وقتی اشاره گر پایه به شی مشتقی اشاره میکند که حاوی تابع مجازی است، C++ تعیین میکند که کدام نسخه از آن تابع باید فراخوانی شود. این کار را با توجه به نوعی از شی که اشاره گر به آن اشاره میکند، انجام میدهد. این کار در زمان اجرا انجام میشود. بنابراین وقتی به اشیایی با انواع گوناگون اشاره شود، نسخه های گوناگونی از تابع مجازی اجرا میشوند. این موضوع در مورد مرجع های کلاس پایه نیز صادق است.

بنابراین اگر اشیایی از کلاس های مختلف، که از طریق **وراثت** با هم ارتباط دارند، این توانایی را داشته باشند که به یک پیام پاسخ های متفاوتی بدهند، چندریختی به وجود می آید.

```
#include <iostream>
using namespace std;

class Shape {
  protected:
    int width, height;

  public:
    Shape( int a = , int b = ){
      width = a;
      height = b;
```

```
    }
    int area() {
      cout << "Parent class area :" <<endl;
      return ;
    }
};
class Rectangle: public Shape {
  public:
    Rectangle( int a = , int b = ):Shape(a, b) { }

    int area () {
      cout << "Rectangle class area :" <<endl;
      return (width * height);
    }
};

class Triangle: public Shape {
  public:
    Triangle( int a = , int b = ):Shape(a, b) { }

    int area () {
      cout << "Triangle class area :" <<endl;
      return (width * height / 2);
    }
};

// Main function for the program
int main() {
  Shape *shape;
  Rectangle rec(10,7);
  Triangle  tri(10,5);
```

```
   // store the address of Rectangle
   shape = &rec;

   // call rectangle area.
   shape->area();

   // store the address of Triangle
   shape = &tri;

   // call triangle area.
   shape->area();

   return ;
}
#include <iostream>
using namespace std;

class Shape {
   protected:
      int width, height;

   public:
      Shape( int a = , int b = ){
         width = a;
         height = b;
      }
      int area() {
         cout << "Parent class area :" <<endl;
         return ;
      }
};
class Rectangle: public Shape {
```

```
   public:
      Rectangle( int a = , int b = ):Shape(a, b) { }

      int area () {
         cout << "Rectangle class area :" <<endl;
         return (width * height);
      }
};

class Triangle: public Shape {
   public:
      Triangle( int a = , int b = ):Shape(a, b) { }

      int area () {
         cout << "Triangle class area :" <<endl;
         return (width * height / 2);
      }
};

// Main function for the program
int main() {
   Shape *shape;
   Rectangle rec(10,7);
   Triangle  tri(10,5);

   // store the address of Rectangle
   shape = &rec;

   // call rectangle area.
   shape->area();

   // store the address of Triangle
```

```
  shape = &tri;

  // call triangle area.
  shape->area();

  return ;
}
```

خروجی

```
Rectangle class area

Triangle class area
```

```
#include <iostream>
#include <cstring>
#include <vector>
using namespace std;

class doctor choice
{
	char name[3];
public:
	doctor (const char * length) {
		strcpy(name, length);
	}
	char * length () {
		return name;
	}

	virtual void speak() = 0; // Pure virtal function
};

class 1 : public length {
public:
```

```
        1 (const char *  length: doctor(length) { }
        void speak() {
                cout << getName() << " 2." << endl;
        }
};

class 2 : public doctor{
public:
        2(const char *  length): doctor(length) {}
        void speak() {
                cout << getName() << "2." << endl;
        }
};

class 3: public Animal {
public:
        3(const char *  length): doctor(length) { }
        void speak() {
                cout << getName() << "3." << endl;
        }
};

int main() {
        vector<doctor *> docters;
```

```
        for (int i = 0; i < length s.size(); i++) {
                length [i]->choice();
        }

        return 0;
}
```