HY-150a Programming Assignment 4

Classes:

In C++, classes are very much like structs, except that classes provide much more power and flexibility. In fact, the following struct and class are effectively identical.

Struct : public by default but Class: private by default

```
ClassPoint {
Public:
    Double x;
    Double y;
};
```

```
Struct Point {
    Double x;
    Double y;
};
```

HY-150a Programming Assignment 4

Inline:

- Increase the execution time of a program.
- Compiler replace those function definition wherever those are being called at compile time instead of referring function definition at runtime.
- NOTE- This is a suggestion to compiler to make the function inline, if function is big, compiler can ignore.

```
inline int square(int x) { return x*x; }
class Vector {
    float x, y, z;
public:
   Vector(float x, float y, float z) :
   x(x), y(y), z(z) {}
   float GetX(void) { return x; }  // by
   default treated inline
};
```

```
int x = square(2);//
   function code copied
   here so this is
   equal to: int x =
   2*2;
Vector v(0, 1, 2);  x =
   v.GetX(); // code
   from Vector::GetX
   also is copied here
```

HY-150a Programming Assignment 4

Overloading:

- You can have multiple definitions for the same function name in the same scope.
- The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.
- You can not overload function declarations that differ only by return type.

HY-150a Programming Assignment 4

Overloading:

```
class printData {
public:
void print(int i) { cout << "Printing int: " << i << endl; }
void print(double f) {
cout << "Printing float: " << f << endl;
}
void print(char* c) {
cout << "Printing character: " << c << endl;
}
};
int main(void) {
printData pd;
pd.print(5);
pd.print(500.263);
pd.print("Hello C++");
return 0;
}
```

HY-150a Programming Assignment 4

Constructor:

- A class constructor is a special member function of a class that is executed whenever we create new objects of that class.
- A constructor will have exact same name as the class and it does not have any return type at all, not even void.
- Constructors can be very useful for setting initial values for certain member variables.

HY-150a Programming Assignment 4

```
class Vehicle {
protected:
      string license;
      int year;
public:
      Vehicle(const string &myLicense, const int myYear)
          : license(myLicense), year(myYear) {}
      const string getDesc() const
          {return license + " from " + stringify(year);}
      const string &getLicense() const {return license;}
      const int getYear() const {return year;}
};
```

Constructor

<<Get-ers>> for be able to
call protected, private members
But without changing var. values

FORTH
Institute of Computer Science

CSD University of Crete

HY-150a Programming Assignment 4

Object-Oriented Programming OOP:

- Encapsulation: grouping related data and functions together as objects and defining an interface to those objects.
- Inheritance: allowing code to be reused between related types.
- Polymorphism: allowing a value to be one of several types, and determining at runtime which functions to call on it based on its type.

HY-150a Programming Assignment 4

Encapsulation just refers to packaging related stuff together : with classes.

If someone hands us a class, we do not need to know how it actually works to use it; all we need to know about is its public methods/data – its interface.

This is often compared to operating a car: when you drive, you don't care how the steering wheel makes the wheels turn; you just care that the interface the car presents (the steering wheel) allows you to accomplish your goal.

HY-150a Programming Assignment 4

Encapsulation just refers to packaging related stuff together : with classes.

If someone hands us a class, we do not need to know how it actually works to use it; all we need to know about is its public methods/data – its interface.

This is often compared to operating a car: when you drive, you don't care how the steering wheel makes the wheels turn; you just care that the interface the car presents (the steering wheel) allows you to accomplish your goal.

HY-150a Programming Assignment 4

Objects being boxes with buttons you can push, you can also think of the interface of a class as the set of buttons each instance of that class makes available.

Interfaces abstract away the details of how all the operations are actually performed, allowing the programmer to focus on how objects will use each other’s interfaces – how they interact.

This is why C++ makes you specify public and private access specifiers: by default, it assumes that the things you define in a class are internal details which someone using your code should not have to worry about.

HY-150a Programming Assignment 4

The practice of hiding away these details from client code is called “data hiding,” or making your class a “black box.”

One way to think about what happens in an object-oriented program is that we define what objects exist and what each one knows, and then the objects send messages to each other (by calling each other’s methods) to exchange information and tell each other what to do.

HY-150a Programming Assignment 4

Inheritance:

- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

- The idea of inheritance implements a relationship between classes.

HY-150a Programming Assignment 4

Inheritance example 1: (Shape-Rectangle):

```
// Base class
class Shape {
public: void setWidth(int w) { width = w;
}
void setHeight(int h) { height = h; }
protected: int width; int height;
};

// Derived class
class Rectangle: public Shape
{
public: int getArea() { return (width *
height); }
};
```

```
int main(void) {
Rectangle Rect; Rect.setWidth(5);
Rect.setHeight(7);
cout << "Total area: " <<
Rect.getArea() << endl;
return 0;
}
```

HY-150a Programming Assignment 4

Inheritance example 2: (Vehicle-Car):

```
public:
    Vehicle(const string &myLicense, const int myYear)
        : license(myLicense), year(myYear) {}
    const string getDesc() const
        {return license + " from " + stringify(year);}
    const string &getLicense() const {return license;}
    const int getYear() const {return year;}
};
```

```
class Car : public Vehicle { // Makes Car inherit from Vehicle
    string style;

public:
    Car(const string &myLicense, const int myYear, const string
        &myStyle)
        : Vehicle(myLicense, myYear), style(myStyle) {}
    const string &getStyle() {return style;}
};
```

HY-150a Programming Assignment 4

Inheritance example 2: (Animal-Cat-Dog):

```
class Animal {
public:
   Animal(char* name) :
       name(name) {}
   void Walk(void);
private:
   char* name;
};


class Cat : public Animal
{
public:
  Cat(char*
name):Animal(name){}
  void Climb(void);
};
```

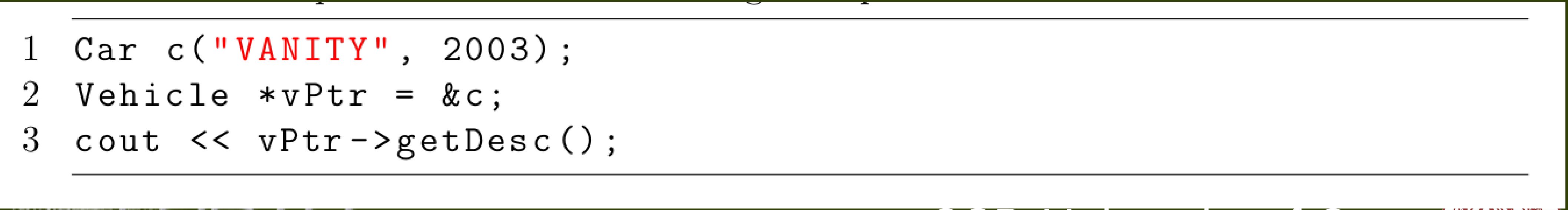
```
class Dog : public
Animal {
public:
  Dog(char*
name):Animal(name){}
   void Bark(void);
};
```

HY-150a Programming Assignment 4

Polymorphism:

- Polymorphism means “many shapes.” It refers to the ability of one object to have many types. If we have a function that expects a Vehicle object, we can safely pass it a Car object, because every Car is also a Vehicle.
- Likewise for references and pointers: anywhere you can use a Vehicle *, you can use a Car *.
- The idea of inheritance implements a relationship between classes.

There is still a problem. Take the following example:



```
Car c("VANITY", 2003);
Vehicle *vPtr = &c;
cout << vPtr->getDesc();
```

HY-150a Programming Assignment 4

There is still a problem. Take the following example:

```
1 Car c("VANITY", 2003);
2 Vehicle *vPtr = &c;
3 cout << vPtr->getDesc();
```

(The -> notation on line 3 just dereferences and gets a member. ptr-> member is equivalent to (*ptr).member.).

Because vPtr is declared as a Vehicle *, this will call the Vehicle version of getDesc, even though the object pointed to is actually a Car.

Usually we'd want the program to select the correct function at runtime based on which kind of object is pointed to.

HY-150a Programming Assignment 4

We can get this behavior by adding the keyword virtual before the method definition:

```
class Vehicle {
    ...
    virtual const string getDesc() {...}
};
```

Compiler does not statically link the code of the function but produce code that will decide what function must be called at runtime (also known as ***late (ή dynamic) binding***).

HY-150a Programming Assignment 4

Polymorphism example:

```
struct Base {
void foo(void)
   { printf("base");}
};
struct Derived : public Base {
void foo(void)
   {printf("derived");}
};
Derived derived;
Base base, *b = new Derived;
base.foo();   //prints base
derived.foo();//prints derived
b->foo();
```

Problem!!!
prints base but intended derived

```
struct Base {
   virtual void foo(void) {
       printf("base");
   }
};
struct Derived : public Base{
   void foo(void) {
       printf("derived");
   }
};
Derived derived;
Base base, *b = new Derived;
base.foo();   //prints base
derived.foo();//prints derived
b->foo();
```

Solve!!!
Prints derived

HY-150a Programming Assignment 4

Pure virtual συναρτήσεις λέγονται οι virtual συναρτήσεις που δεν έχουν υλοποίηση

Δηλώνονται κανονικά ως virtual συναρτήσεις προσθέτοντας ένα ***‘=0’*** στο τέλος τους

Δε μπορούμε να δημιουργήσουμε στιγμιότυπα από κλάσεις που έχουν έστω και μια pure virtual συνάρτηση

Αυτές οι κλάσεις ονομάζονται ***abstract***

Χρησιμοποιούνται για να ορίσουν γενική λειτουργικότητα που θα υλοποιηθεί αργότερα από τα derived classes

Αν ένα derived class δεν υλοποιεί pure virtual συναρτήσεις ενός base, τότε είναι και αυτό *abstract*

Μπορούμε να έχουμε pointers και references σε abstract classes, όχι όμως αντικείμενα

HY-150a Programming Assignment 4

Ενδέχεται να έχουμε δεσμεύσει δυναμικά μνήμη για πολυμορφικά αντικείμενα.
Κατά τη διαγραφή τους με delete θέλουμε πάντα να κληθεί ο κατάλληλος (πιο derived) destructor.
Για αυτό το λόγο ορίζουμε το destructor της base class ***virtual.*** Πάντα όταν σχεδιάζουμε μια κλάση που πιθανό να κληροδοτήσει σε κάποια άλλη ορίζουμε τον destructor της ως virtual.
Στο επόμενο παράδειγμα, αν ο destructor της Shape δεν ήταν virtual, κατά το *delete shape* θα καλούνταν μόνο ο destructor της Base (Shape) και θα είχαμε memory leak αν είχαμε κάνει new ένα *Object* της Derived κλάσης Circle.

HY-150a Programming Assignment 4

Στο παράδειγμα δίπλα, αν ο destructor της Base δεν ήταν virtual, κατά το *delete b* θα καλούνταν μόνο ο destructor της Base και θα είχαμε memory leak για το *Object o* της κλάσης Derived

```
struct Base {
   Base(void) { }
   virtual ~Base(void) { foo(); }
   virtual void foo(void)
      { printf("base"); }
};
struct Derived : public Base {
   Derived(void) { o = new Object; }
   ~Derived(void){ foo(); delete o;}
   void foo(void)
      { printf("derived"); }
private:
   Object* o;
};
Base* b = new Derived;
delete b; //prints derived base
```

Στο επόμενο παράδειγμα, αν ο destructor της Shape δεν ήταν virtual, κατά το *delete shape* θα καλούνταν μόνο ο destructor της Base (Shape) και θα είχαμε memory leak αν είχαμε κάνει new ένα *Object* της Circle.

HY-150a Programming Assignment 4

```
#include <stdio.h>
class Shape {
  public:
  virtual ~Shape();
  virtual void draw() = 0;
};
```

Virtual Destructor

Pure virtual

```
class Circle : public Shape {
public:
   ~Circle();

  virtual void draw();
};
```

simple Destructor

```
Shape::~Shape() { printf("shape
destructor\n"); }

// void Shape::draw() {
//     printf("Shape::draw\n");}

Circle::~Circle() { printf("circle destructor\n"); }
void Circle::draw() { printf("Circle::draw\n"); }
```

```
int main() {
  Shape *shape = new Circle;
  shape->draw();
  delete shape;
  return 0;
}
```

If you compile it and run it as:

```
($ g++ -Wall xxx.cpp -o xxx
$ ./xxx Circle::draw circle destructor shape
destructor
```

Will shows: Circle::draw

```
circle destructor
shape destructor
```

HY-150a Programming Assignment 4

Verify your understanding of how the virtual keyword and method overriding work by performing a few experiments:

Remove the virtual keyword from each location individually, recompiling and running each time to see how the output changes. Can you predict what will and will not work?

Try making Shape::draw non-pure by removing = 0 from its declaration.

Try changing shape (in main()) from a pointer to a stack-allocated variable.

HY-150a Programming Assignment 4

Assignment 4

HY-150a Programming Assignment 4

Implement a class called Tool.

- It should have an int field called strength and a char field called type.
- You may make them either private or protected.
- The Tool class should also contain the function void setStrength (int), which sets the strength for the Tool.

HY-150a Programming Assignment 4

- Create 3 classes called Rock, Paper, and Scissors, which inherit from Tool.
- Each of these classes will need a constructor which will take in an int that is used to initialize the strength field.
- The constructor should also initialize the type field using 'r' for Rock, 'p' for Paper, and 's' for Scissors.

Create a virtual function in class Tool, the Tool::play, to compare Rock paper and Scissors strength.

HY-150a Programming Assignment 4

The strength field shouldn't change in the tool::play , which returns true if the original class wins in strength and false otherwise.

Class game will be responsible for playing the game and will contain 2 tool * (pointers) one for player and one for PC as it is unknown which type class each one will choose in each round of the game.

HY-150a Programming Assignment 4

H Game::draw να καλεί τη Shape::draw κάθε σχήματος με παράμετρο την τοποθεσία που θα απεικονίζει το σχήμα.

H Game, θα μπορεί να γνωρίζει τις διαστάσεις του καμβά.

HY-150a Programming Assignment 4

Paper ή square: η οποία ζωγραφίζει πάνω στον καμβά ένα τετράγωνο με περίμετρο από ‘.’, που με πλευρά μήκους side και με με x, y το κέντρο του σχήματος που αντιστοιχεί σε σημείο πάνω στον κεντρικό άξονα του καμβά και απέχει τουλάχιστον x χαρακτήρες από το κέντρο του καμβά.

HY-150a Programming Assignment 4

Rock ή circle: η οποία θα ζωγραφίζει πάνω στον καμβά ένα κύκλο με περίμετρο από αστεράκια (.), με x, y το κέντρο του σχήματος που αντιστοιχεί σε χαρακτήρα πάνω στον κεντρικό άξονα του καμβά και απέχει τουλάχιστον δέκα χαρακτήρες από το κέντρο του καμβά.

Scissors ή ένα χιαστί σχήμα: η οποία θα ζωγραφίζει πάνω στον καμβά δυο χιαστί με γραμμές από '.'.

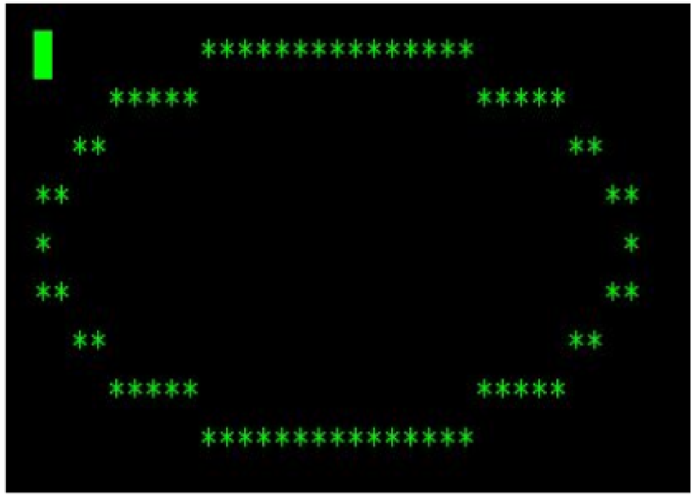
HY-150a Programming Assignment 4

Draw the scene to canvas /Canvas save

```
For each shape in the scene:
    Draw(canvas,shape);
end loop
```

If a **target-pixel** is out of bounds, do nothing about it.

Origin of an shape must be in bounds!



```
         ***************
    *****               *****
  **                         **
**                             **
*                               *
**                             **
  **                         **
    *****               *****
         ***************
```

ascii circle (without fill)

Write a text file to save Canvas! (assign1 conventions)

Don’t waster memory

- Only canvas can be an array of static size.
- Deleting shapes means also deallocating their memory.
- Before quitting deallocate all dynamically allocated memory.

❖ use malloc() & free() (C style)
❖ use new & delete (C++ style)
❖ **Do not mix** malloc()/delete , new/free() (*undefined behavior)*
❖ Use .clear() on STL containers

HY-150a Programming Assignment 4

ΑΠΟΡΙΕΣ ???

FORTH Institute of Computer Science

CSD University of Crete