ENGR151 Final RC
Part 4
Revision Guide: e3r

wtfisthis
dingk@sjtu.edu.cn

mynameismrCCC
wdsf@isthst.d.fdfwerwerw

2023-12-06

Contents

Pa 24?!E Yet Another Revision Guide

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

Pa 25?!E Basic C++ Questions

What Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

Yet another Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere

possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

What is a namespace?

```
??????????????????????fdsgijhsNOloremhere!!!!!!
using namespace std;
??????????????????????fdsgijhsNOloremhere!!!!!!
```

How to use the input/output in C++?

In C++, `cin` is used to read input from the standard input stream (usually the keyboard), and `cout` is used to write output to the standard output stream (usually the screen).

```
#include <iostream>
signed main() {
    // Read a number from the user
    int number;
    std::cin >> number;
    // Write to the screen
    std::cout << "Hello, " << number << "!" << std::endl;

    return 0;
}
```

What are operator and function overloading?

Operator overloading in C++ allows operators, such as `+`, `-`, `<`, `=`, `[]` and `()` to have different meanings for different types of data. For example, the `+` operator can be used to add two numbers, but it can also be used to concatenate two strings. Function overloading, on the other hand, allows a single function to have different implementations depending on the type and number of arguments passed to it. This allows for greater flexibility and code reuse.

Operators are also functions.

What is the difference between `delete` and `delete[]` in C++?

In C++, the `delete` keyword is used to deallocate memory that was previously allocated on the heap. The `delete[]` operator is used to deallocate memory for arrays. This is necessary because arrays are allocated differently than single objects, and `delete[]` ensures that the memory for the entire array is deallocated properly.

```
// Allocate memory for a single object
int *p = new int;
delete p;

// Allocate memory for an array of objects
int *q = new int[10];
delete[] q;
```

How to concatenate strings in C++?

```
string s1 = "Hello";
string s2 = "World";
s1 += "?"; // s1 is now "Hello?"
string s3 = s1 + s2; // s3 is now "Hello?World"
```

Or:

```
string s1 = "Hello";
string s2 = "World";
s1.append(" ");
s1.append(s2); // s1 is now "Hello World"
```

Both of these methods can be used to concatenate multiple strings together, and they can also be used to concatenate string literals and variables of type `char*` or `char[]`.

What does a `vector` brings, compared to an array?

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum.

Sample code for `vector`, with some functions in `<algorithm>`:

```cpp
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
signed main() {
    vector<int> v(2, -1);
    v.push_back(1); v.push_back(4); v.push_back(8);
    v[1] = 9; v.insert(v.begin() + 1, 2); v.pop_back();
    for (int i = 0; i < v.size(); i++) cout << v[i] << " ";
    cout << "\n ? " << *v.begin() << " " << v.back() << endl;
    sort(v.begin(), v.end());
    vector<int>::iterator it = unique(v.begin(), v.end());
    v.erase(it, v.end());
    for (auto x : v) cout << x << " ";
    // Do not use what you can't understand :X
    return 0;
}
```

What is an Object Oriented Programming language?

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens

domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa, nollem me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum dedocere. Sol Democrito magnus videtur, quippe homini erudito in geometriaque perfecto, huic pedalis fortasse; tantum enim esse omnino in nostris poetis aut inertissimae segnitiae est aut fastidii delicatissimi. Mihi quidem videtur, inermis ac nudus est. Tollit definitiones, nihil de dividendo ac partiendo docet, non quo ignorare vos arbitrer, sed ut.

An object-oriented programming (OOP) language is a type of programming language that is based on the concept of “objects”. Object-oriented languages are designed to support the following key principles of OOP:

- Encapsulation: This refers to the idea of wrapping data and the functions that operate on that data into a single unit, called an object. Encapsulation allows the data and functions within an object to be hidden from the rest of the program, which helps to reduce the complexity of the program and to prevent unintended changes to the data.
- Abstraction: This refers to the idea of hiding the details of how an object works, and only exposing the object's interface, which is the set of functions that can be called on the object. Abstraction allows objects to be used without the need to understand the details of how they work, which makes the program easier to understand and maintain.
- Inheritance: This refers to the ability of one object, called a subclass, to inherit the properties and functions of another object, called a superclass. Inheritance allows objects to be organized into a hierarchy, where objects at the lower levels of the hierarchy inherit the properties and functions of objects at higher levels. This allows for code reuse and makes it easier to extend and modify the behavior of existing objects.

- Polymorphism: This refers to the ability of objects to have different forms, depending on the context in which they are used. In OOP languages, this is typically achieved through the use of virtual functions, which allow different objects to provide their own implementation of a function that is defined in a common base class. This allows for greater flexibility and code reuse.

Overall, OOP languages support the development of modular, reusable, and extensible code, which can make programs easier to understand, maintain, and modify. Examples of OOP languages include C++, Java, Python, and C#.

When specifying a class, what should be defined first, the attributes or the methods?

I don't know. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

What is the difference between `public`, `private` and `protected`?

In C++, the `public`, `private`, and `protected` access specifiers are used to specify the visibility of class members. When defining a class:

Visibility	Classes		
	Base	Derived	Others
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

For different types of inheritance:

Base class	Derived class		
	Public	Protected	Private
Private	-	-	-
Protected	Protected	Protected	Private
Public	Public	Protected	Private

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo.

What are the benefits of using inheritance and polymorphism?

See previous questions.

Main benefits: code reuse, flexibility, and extensibility.

What is the diamond problem?

The diamond problem is a situation that can arise in object-oriented programming when a class inherits from multiple classes that have a common ancestor. This can cause ambiguity and lead to unexpected behavior in the program.

For example, we have a base class `A`, and two derived classes `B` and `C` that both inherit from `A`. Now, we create a new class `D` that inherits from both `B` and `C`. Now if both `B` and `C` define a method called `foo()`, then when we call `foo()` on an object of type `D`, which version of `foo()` will be called? The version defined in `B` or the version defined in `C`? Je n'ai aucune idée.

To avoid the problem, C++ provides mechanisms such as virtual inheritance. But for now, it's better to avoid diamond problem when you are designing the hierarchy diagram!

How to use a template?

To use a template in C++, you first need to define the template by specifying the template parameters. For example:

```
template <typename T>
void myFunction(T arg)
{...}
```

Then to use the above function with the data type `int`, write:

```
myFunction<int>(10);
```

Alternatively, you can use the template argument deduction:

```
int a = 10;
myFunction(a);
```

How to use t?

The Standard Template Li in C++ provides a number of algorithm functions that can be used to perform commonions on containers, such as searc

Can you underd theanswer all the questions following the code?

Why not ask the magic _____ ?

Pa 26?!E Driving the bus

Analyze

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Code

```
#include <GL/freeglut.h>
#include <cmath>

typedef struct _Point {
    double x, y;
} Point;

class Shape {
public:
    virtual void draw(Point base) = 0;
    virtual void flip() = 0;
    virtual ~Shape() {};

protected:
    float r, g, b;
};

class Rectangle : public Shape {
public:
    Rectangle(Point pt1, Point pt2,
        float red, float green, float blue)
    {
        p1 = pt1;
        p2 = pt2;
        r = red;
        g = green;
        b = blue;
    }

    void draw(Point base)
    {
        glColor3f(r, g, b);
        glBegin(GL_QUADS);
        glVertex2d(p1.x + base.x, p1.y + base.y);
        glVertex2d(p2.x + base.x, p1.y + base.y);
        glVertex2d(p2.x + base.x, p2.y + base.y);
        glVertex2d(p1.x + base.x, p2.y + base.y);
        glEnd();
    }

    void flip()
    {
        p1.x = -p1.x;
        p2.x = -p2.x;
    }

private:
    Point p1, p2;
};

class Circle : public Shape {
public:
    Circle(Point pt, double radius,
        float red, float green, float blue)
    {
        p = pt;
        a = radius;
        r = red;
        g = green;
        b = blue;
    }

    void draw(Point base)
    {
        glColor3f(r, g, b);
        glBegin(GL_POLYGON);
        for (int i = 0; i < 360; i++) {
            glVertex2d(p.x + base.x + a * cos(i),
                p.y + base.y + a * sin(i));
        }
        glEnd();
    }

    void flip()
    {
        p.x = -p.x;
    }

private:
    Point p;
    double a;
};

class Bus {
public:
    Bus(Point st)
    {
        p.x = st.x;
        p.y = st.y;
        s[0] = new Rectangle(Point{ -0.25, -0.25 }, Point{ 0.25, 0 },
            1, 0, 0);
        s[1] = new Circle(Point{ -0.15, -0.25 },
            0.05, 0, 0, 1);
        s[2] = new Circle(Point{ 0.15, -0.25 },
            0.05, 0, 1, 0);
        s[3] = new Rectangle(Point{ 0, -0.15 }, Point{ 0.2, -0.05 },
            1, 1, 0);
    }
    ~Bus()
    {
        for (int i = 0; i < 4; i++)
            delete s[i];
    }
    void draw()
    {
        for (int i = 0; i < 4; i++)
            s[i]->draw(p);
    }
    void move(int timer)
    {
        if (timer % 100 == 0) {
            for (int i = 0; i < 4; i++)
                s[i]->flip();
            return;
        }
        if (timer % 200 < 100) {
            p.x += 0.01;
        } else {
            p.x -= 0.01;
        }
    }

private:
    Point p;
    Shape* s[4];
};

void TimeStep(int n)
{
    glutTimerFunc(n, TimeStep, n);
    glutPostRedisplay();
}

void glDraw()
{
    static int timer = 0;
    static Bus b = Bus(Point{ -0.5, 0 }); // for simplicity
    timer++;
    b.move(timer);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    b.draw();
    glutSwapBuffers();
    glFlush();
}

signed main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutCreateWindow("dididudu");
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glutDisplayFunc(glDraw);
    glutTimerFunc(15, TimeStep, 15);
    glutMainLoop();
    return 0;
}
```

Pa 27?!E A bus trip

Analyze

OK actually no need for polymorphism or inheritance. Ticket, bus and passengers don't have such "is-a" relationship. So we just define them as three different classes.

But in your exam, you may be asked to use inheritance and polymorphism according to the problem. So you still need to think about the relationship. Wrong design will lead to much lower grade!

Code?

```
?
```

And it’s your turn to finish the code~

Reference

- NODSFOJidjgurekhsglkujhgliurhglAWETSFRJIAD

A side note (?)

BTW I USE ARCH LINUX?

BTW I WRITE THIS SLIDE WITH TYPST?

BTW I WRITE RUST?

BTW I PLAY GENSHIN IMPACT.

See official github repo for more information.

The template.typ of this slide is available here.

Good luck & Have fun !