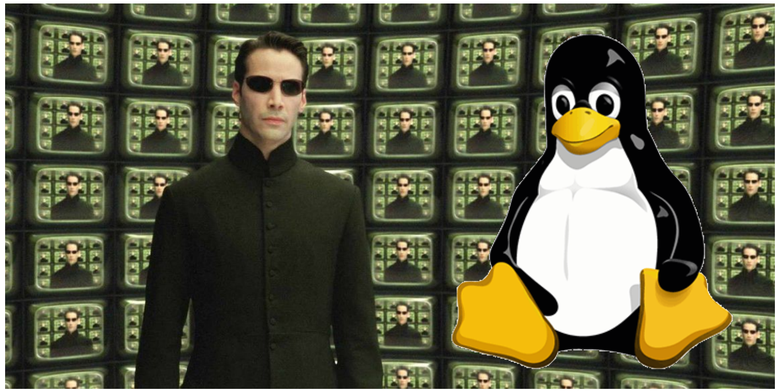
09.07.20 17:30

CSCI 125
Introduction to Computer Science and Programming II
Lecture 6: User Class I



Jetic Gū

2020 Summer Semester (S2)

Overview

- Focus: Basic C/C++ Syntax
- Architecture: Linux/Unix OS
- Core Ideas:
    1. Basic declaration
    2. Constructor
    3. Tutorial: compiling class using headers

P0
Review

Data Types

- Primary √
  - Integers, Characters, Boolean √
  - Floating point √
  - Void√
- Derived
  - Function√, Array√, Pointer, Reference√
- User Defined
  - **Struct, Class**, Enumerate√, Typedef√

Review

P1
Declar.

Introduction to Class

Declarations, Basic Member functions

Summary

P1
Declar.

Object-Oriented Programming

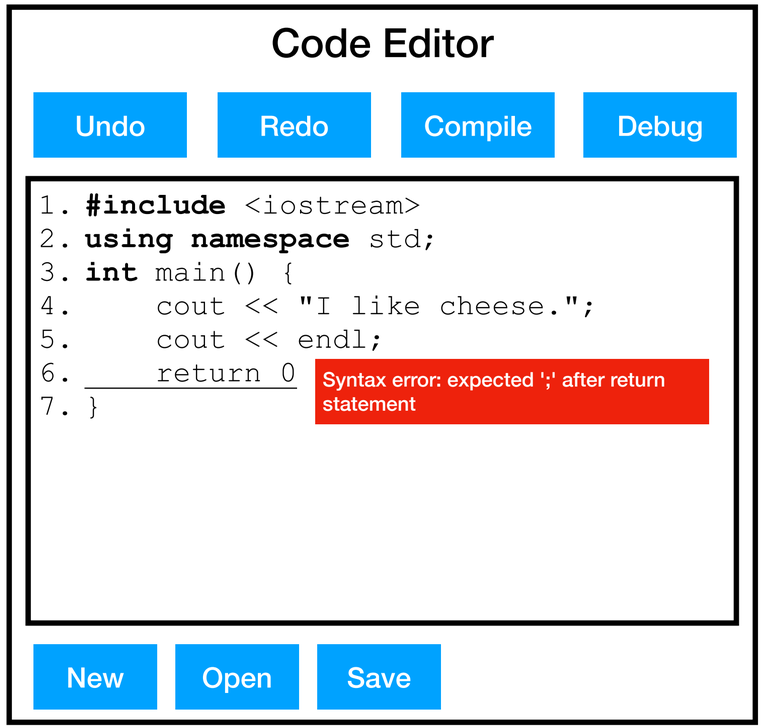
- Data-centric software engineering approach
- Design all components of the software based on Data Structures
- Most modern software systems uses OOP
- An object
  - An object is an instance of a user defined datatype
    e.g. `string str;  // str is an` **`instance`** `of class string`
  - An object has member variables and functions: e.g. `str.c_str()`

Concept

P1
Declar.

Object-Oriented Programming

- Example: Simple Code Editor
  - Central data structure: source-file
    - Sub-structure: edit history manager
    - Sub-structure: syntax-checker
    - Sub-structure: API to compiler
    - Sub-structure: API to debugger

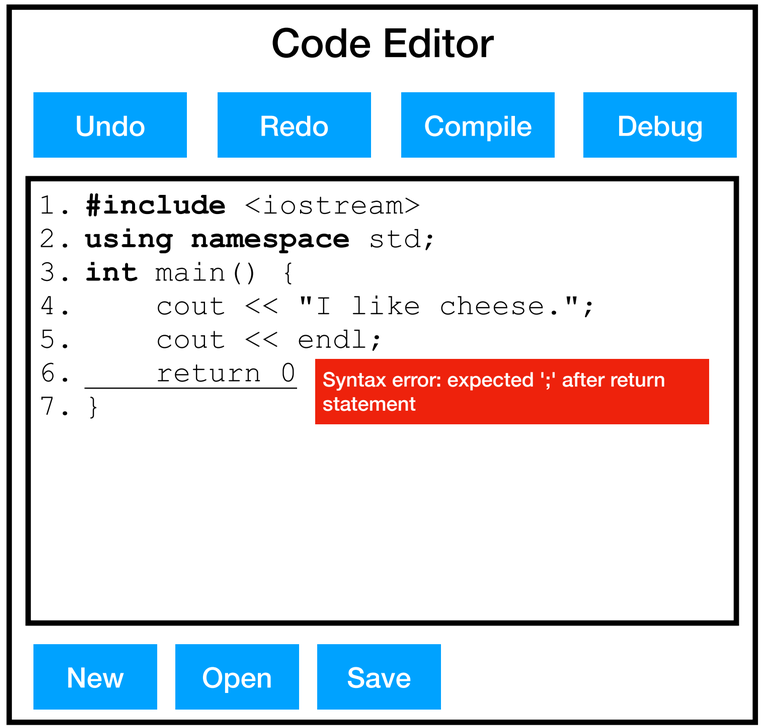


Concept

P1
Declar.

Object-Oriented Programming

- Example: Simple Code Editor
  - Object: source-file
    - Sub-component (object)
      edit history manager
    - Sub-component (object)
      syntax-checker
    - Sub-component (object)
      API to compiler
    - Sub-component (object)
      API to debugger



Concept

P1
Declar.

Declaration

- Declaration starts with `class`
  `struct` also works, but don't use it
- Declaration of member functions and variables
  - access specifiers: kinda like scope
- `objectNames`: optional

```
1. class ClassName {
2.   accessSpecifier1:
3.     member variable1...
4.   accessSpecifier2:
5.     member function1...
6. } objectNames;
```

Concept

1. `struct` was from C, in practice do not use `struct`, always write `class` in C++

Access Specifiers

- Access specifier: `public`
  - Can be accessed anywhere the object is visible

```
1. class Employee {
2.   public:
3.     int id;
4.     std::string name;
5.     int salary;
6. } john;
7. Employee smith;
8. john.name = "John Cooper";
9. std::cin >> smith.id;
```

Concept

P1
Declar.

Access Specifiers

- Access specifier: `public`
  - Can be accessed anywhere the object is visible
- Access specifier: `private`
  - Can be accessed **ONLY** by other members of this object

```
1.  class Employee {
2.    private:
3.      int salary;
4.    public:
5.      void setSalary(int val) {
6.          salary = val;
7.      }
8.  } john;
9.  john.salary = 12; // doesn't work
10. john.setSalary(12); // works!
```

Concept

1. C++ class does not use anything like `self` in python class

P1
Declar.

Access Specifiers

- Access specifier: `public`
  - Can be accessed anywhere the object is visible
- Access specifier: `private`
  - Can be accessed **ONLY** by other members of this object
- Access specifier: `protected`
  - Covered in *Lecture 6: User Class III*[1]

```
1.  class Employee {
2.    private:
3.      int salary;
4.    public:
5.      void setSalary(int val) {
6.          salary = val;
7.      }
8.  } john;
9.  john.salary = 12; // doesn't work
10. john.setSalary(12); // works!
```

Concept

1. Next week

P1
Declar.

Access Specifiers

- Access specifiers
  - The default one is `private`
  - Everything under the current specifier belongs to that access type

```
1.  class Employee {
2.      std::string name;  // private
3.    public:
4.      void setSalary(int val);
5.      void setName(std::string val);
6.    private:
7.      int salary;
8.  };
```

Concept

P1
Declar.

Member Variables

- Variable declaration just like normal variables
- Can have initial values, the same as normal variables

```
1.  class Employee {
2.      std::string name;
3.      int salary=12;
4.      int age=-1;
5.  };
```

Concept

P1
Declar.

Member Functions

- Usually, declare as function prototypes first
  - Same as normal function prototypes
  - Can have overrides
- Complete the declaration later
  `type Class::Function ...`

```
1.  class Employee {
2.        int id;
3.    public:
4.      void setId(int val);
5.  };
```

Prototype

```
7.  void Employee::setId(int val) {
8.        id = val;
9.  }
```

Complete definition

Concept

Example (demo1)

- Circle area calculation (`demo1.cpp`)
  - Downloadable as Handout `H804`
- 4: `radius` is a `private` member variable
- 6, 7: `public` member function prototype

```
3.  class Circle {
4.      double radius;
5.      public:
6.          void setRadius(double val);
7.          double area();
8.  };
```

Example (demo1)

- 10: accessing private member variable radius in member function
- 25: calling member function, this changes the value for `x.radius`

```
10. void Circle::setRadius(double val) {
11.     radius = val;
12. }


23.     double val;
24.     cin >> val;
25.     x.setRadius(val);
```

Example (demo2)

- Circle area calculation (`demo2.cpp`)
  - Downloadable as Handout `H804`
- Implementing member functions inside the class declaration
- Usually, you do this when the function is very simple

```
3.  class Circle {
4.    double radius;
5.    public:
6.      void setRadius(double val)
    {radius = val;}
7.      double area() {return
    radius*radius*3.14}
8.  };
```

Constructor of Class

Summary

What are Constructors

- Constructors are member functions
- Constructors are called immediately when a class object is declared and initialised
  - Used to initialise a class object
- Similar to `def __init__(self, ...)` in python classes

Concept

Constructor of a Class (demo3)

- 7: Prototype declaration
  - Must be `public`
  - Syntax
    `ClassName(...);`
    `// do not add type`
- 11: Prototype implementation
  - Syntax
    `ClassName::ClassName(...){...}`

```
4.  class CountDown {
5.    int count;
6.    public:
7.      CountDown(int val);
8.      void do();
9.  };

11. CountDown::CountDown(int val) {
12.     count = val;
13. }

23. CountDown c(12);
```

1. Handout H804

Constructor of a Class (demo3)

- 23: declare a new variable, which is an instance/object of class `CountDown`
  - Constructor called, `c.count` initialised to 12

```
4.  class CountDown {
5.    int count;
6.    public:
7.      CountDown(int val);
8.      void do();
9.  };

11. CountDown::CountDown(int val) {
12.     count = val;
13. }

23. CountDown c(12);
```

Constructor of a Class (demo3)

- Constructors cannot be called explicitly as if they were regular member functions
- Constructors are only executed once, when a new object of that class is created

```
4.  class CountDown {
5.    int count;
6.    public:
7.      CountDown(int val);
8.      void doCount();
9.  };

11. CountDown::CountDown(int val) {
12.     count = val;
13. }

23. CountDown c(12);
```

Constructor of a Class

- Constructors do support overloading, just like any other member function or regular function

```
1.  class MyClass {
2.   public:
3.     MyClass(int val);
4.     MyClass(int val1, int val2);
5.     MyClass(int val1, int val2, int val3);
6.  };
```

Class in Headers

Also, practice question p022

Summary

Downloads

- Handout `H804`
  - p022.h
  - demo4.cpp
- Task: write p022.cpp

Concept

P3
Tutorial: Class

Instructions

- Step 1: finish the function declarations in p022.cpp

p022.cpp

```
1. #include "p022.h"
2.
3. void Circle::setRadius(double
   val) {
4.     radius = val;
5. }
6.
7. double Circle::area() {
8.     return radius * radius * pi;
9. }
```

Demo

Instructions

- Step 2: Compile the object file for p022.cpp
  `$ g++ p022.cpp -c -o p022.o`

p022.cpp

```
1. #include "p022.h"
2.
3. void Circle::setRadius(double
   val) {
4.      radius = val;
5. }
6.
7. double Circle::area() {
8.      return radius * radius * pi;
9. }
```

Demo

P3
Tutorial: Class

Instructions

- Step 3: Compile the main demo4 file with object `p022.o`
  `$ g++ demo4.cpp p022.o -o demo4`

- Run the programme to test your code
  `$ ./demo4`

demo4.cpp

```
1. #include <iostream>
2. #include "p022.h"
3. using namespace std;

5. int main() {
6.     Circle x, y;  // declare 2 new objects of class circle

8.     // cin >> x.radius;  // this will not work

10.    double val;
11.    cin >> val;
12.    x.setRadius(val);
13.    cin >> val;
14.    y.setRadius(val);
15.    cout << "area of circle 1 is: " << x.area() << endl;
16.    cout << "area of circle 2 is: " << y.area() << endl;
17.    return 0;
18.}
```

Demo