UNSW SYDNEY

MTRN2500

Tutorial 2 – Classes I & Coding Practices

Classes I

UNSW
SYDNEY

Namespaces

- A means of providing scope to code that relates to a particular feature
- Reduces the likelihood of naming conflicts
- Use the scope-resolution operator (::) to access something within a namespace

```
1.namespace name {
2.     void fun();
3.}

4.name::fun();  // Accessing inside namespace
```

What are Classes

- A user-definable data structure
- A means of housing related information together
- Containing both data members (variables) and functions that relate together
- An extension of a struct (something you would have seen in C)
- When a class is instantiated, the variable you get back is referred to as an object of the class

Class definition and instantiation *(from lectures)*


```
1. class Circle {          // Classname
2. public:
3.    double getRadius(); // Member functions
4.    double getArea();
5. private:
6.    double mRadius;     // Data members
7.    std::string mColor;
8. };
9. Circle circle;          // Define an object

10.class SoccerPlayer {    // Classname
11.public:
12.   void run();          // Member functions
13.   void jump();
14.   void kickBall();
15.private:
16.   std::string mName;   // Data members
17.   int mNumber;
18.   int mXLocation;
19.   int mYLocation;
20.};
21.SoccerPlayer soccerPlayer; // Define an object
```

Use PascalCase for classname

public and private are Access Control Modifiers

- public: member accessible both inside and outside the class
- private: member only accessible inside the class
- The order of public and private can be arbitrary
- We recommend putting public first in this course (class users usually care about this part most)

Use prefix “m” to indicate data members

Class definition must end with a semicolon

Use camelCase for instance name

Object-oriented Programming (OOP) in C++ (ntu.edu.sg)
https://onlinegdb.com/A93pHtYna
#SytleGuide 301, 302, 303, 335, 388, 389

Defining class functions

- Must give member functions a definition
- Definition should generally be done in your .cpp file
- To denote defining function in class use '::' operator
- All data members (public and private) can be accessed when implementing member functions

```
1.#include <iostream>
2.#include <string>

3.class Circle {          // classname
4.public:
5.    double getRadius(); // Member functions
6.    double getArea();
7.private:
8.    double mRadius;     // Data members
9.    std::string mColor;
10.};

11.double Circle::getRadius() {
12.    return mRadius;
13.}
```

Accessing class variables

- Use the ‘dot’ operator to access members of the class
- Must first have an instance of the class to access this
- Can only access public members outside of class functions

```
1.// Using the class from the previous slide

2.Circle c1{}; // create an instance
3.double a {c1.mRadius}; // reference a data member of circle
4.double b {c1.getRadius()}; // reference a member function of circle
```

Constructors

- A means of initialising the class when it is first created
- Automatically called on object creation
- Must have the same name as the class itself
- Can be multiple constructors (overloaded constructors) for a given class depending on the input
- Complete two main tasks:
  - Setup all variables
  - Perform any initialisation tasks

Constructors -- continued

```
1. class Circle {                // Classname
2. public:
3.     Circle() = default;       // explicitly asking compiler to provide a default constructor
4.     // Circle();              // default constructor
5.     Circle(double radius, std::string color)
6.         : mRadius{radius}, mColor{color} {} // Constructor using member initialiser list

7.     double getRadius();       // Member functions
8.     double getArea();
9. private:
10.    double mRadius {};        // Data members
11.    std::string mColor {};
12.};
```

Coding practices

Paradigms

UNSW
SYDNEY

Writing good code

When writing code, more than just completing the task we have to focus on writing good code

- Easy to read (for yourself or someone else)
- Maintainable
- Extensible

Coding practices

- DRY (don’t repeat yourself)
- KISS (keep it simple)
- Commenting
- Consistency
- Use modern/C++ coding practices

Lab tasks

UNSW
SYDNEY