ECE 150 *Fundamentals of Programming*

Introduction to classes

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

Douglas Wilhelm Harder, M.Math. LEL
Prof. Hiren Patel, Ph.D., P.Eng.
Prof. Werner Dietl

Outline

- In this lesson, we will:
  - Describe issues with using only primitive data types
  - Look at examples where simple local variables are insufficient
  - Introduce the idea of classes to solve this problem

Limitations of primitive data types

- To this point, we have discussed primitive data types:

```
char
short  int   long
float  double
bool
```

  - In reality, most complex objects require more parameters to describe them
- Even now, every time we send an array as an argument,
      we must also send the capacity as a separate parameter

Limitations of primitive data types

- Suppose you create a triangle element for a tessellation
  - Such a triangle has:
    - Three corners
      - Each corner has three coordinates
    - A color
      - A color has three values: red, green and blue
  - This is just 12 variables

Limitations of primitive data types

- Suppose we author functions to render or manipulate such triangles
  - Each function could have up to twelve parameters
- Suppose we now add a transparency
  - We may have to update every function,
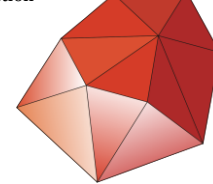   and consequently, every updated function call

Limitations of primitive data types

- Now, suppose we wanted to create a simple surface:
  - Here we have eleven triangles
  - This is 99 coordinates, 11 colors and 11 transparency levels
- We could store this as a number of arrays
  - However, we must still pass a number of arrays
- Suppose after a while, you want to add a *fountain fill*
  - Each triangle now requires direction and a second color
  - Now we must update every function
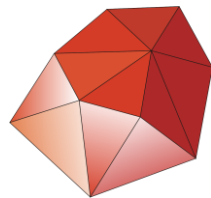   and every function call again...

Limitations of primitive data types

- Such tessellations are the basis of most rendered images
  - This one was created by Gilles Tran (see http://www.oyonale.com/)

Limitations of primitive data types

- Also, notice that *color* is something independent of a triangle
  - Does every object that has a color need to deal with colors independently?

Student records

- Suppose you want to create a *marks manager*
  - Every student has:
    - A given name or names   `std::string`   Syed
    - A surname   `std::string`   Inzamam-ul-Haq
    - A uWaterloo student ID number   `std::string`   20071005
    - A uWaterloo user ID   `std::string`   inzi
    - A program   `std::string`   COMPE
    - A list of zero or more courses, each of which has:
      - A subject ID   `std::string`   ECE
      - A subject number   `std::string`   150
      - A term in which it was taken   `std::string`   0979
      - A grade   `int`   71

Student records

- All this data could be stored in arrays and arrays of arrays...
  - We could author a number of functions to calculate:
    - Term averages
    - Overall averages
  - Much of this data would have to be passed to these functions
- Once again, suppose we want to add *grading designations*
  - Only some courses count towards the term and cumulative averages
  - Once again, we'd have to modify all functions that require this information, and all function calls to these functions
- Engineers cannot have more than three failed courses
  - While updating grading designations, one may forget to update such code, as three failed courses only count those that count towards averages

Student records

- Additionally, suppose someone makes a mistake:
  - They use these arrays but accidentally assign a grade of 110
  - What if the course was listed as ECE 155, instead?

- There is no way to check this, as any user can modify any entry in any of the arrays

- Suppose that a user accidentally modifies the capacity of an array:

```
if ( courses_taken[n] = 0 ) {
    // The 'n'th student has not taken any
    // courses yet
} else {
    // Calculate the overall average
}
```

Classes to the rescue

- Classes are a means of aggregating and also protecting data
  - Definition: aggregate (noun)
    A whole formed by combining several (typically disparate) elements.
  - Any such type is a collection of member variables, each of which is either a primitive data type, or even other classes
    - Unlike arrays, different member variables can have different types
  - Any local variable of that type allows each member variable to have different values
  - Like variables of type `int` and `double` that can be modified and passed to functions, instances of classes can have their member variables modified and yet they can be passed as a whole to functions

Summary

- Following this lesson, you now
  - Understand the issues with primitive data types
  - Understand the need for user-defined aggregate data types
  - Have an idea as to what we will be covering in this section

References

[1]   No references?

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using Consolas.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.

4