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

ECE 150 *Fundamentals of Programming*

`class` data structures

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

© 2018 by Douglas Wilhelm Harder and Hiren Patel.
Some rights reserved.

Outline

- In this lesson, we will:
  - Introduce the 3-body problem and a simulation thereof
  - Introduce the `class` keyword and member variables
  - Show how to declare and initialize instances of classes or *objects*
  - Describe how to access and manipulate the member variables
  - Describe passing objects as arguments to functions
  - Discuss why using pass by reference is appropriate for objects

The three-body problem

- Suppose we are trying to simulate the three-body problem for stars:
  - Each star has:
    - A position x, y, and z
    - A velocity v_x, v_y, v_z
    - A mass m

Wikipedia User:Skatebiker

- We could store arrays for each of these:

```
double rigil_kentaurus_position[3]; // km
double rigil_kentaurus_velocity[3]; // km/h
double rigil_kentaurus_mass;        // kg
// Same for 'toliman' and 'proxima_centauri'
```

Classes

- The three local variables storing information about Rigel Kentaurus are all related
  - Just like an array and capacity are related
  - We would like to like to *group* this information together

```
// Class declaration
class Body;

// Class definition
class Body {
    public:
        // Member variables
        double position_[3];   // km
        double velocity_[3];   // km/h
        double mass_;          // kg
};
```

This class contains all information necessary to describe one star for the problem at hand...

Classes

- As a general rule,
  - Class names will be capitalized
  - Member variable will be suffixed with an underscore
- This is only a naming convention and is not required
  - Recall that *reserved identifiers* are prefixed with an underscore or contain a sequence of two underscores

```
// Class definition
class Body {
    public:
        // Member variables
        double position_[3];    // km
        double velocity_[3];    // km/h
        double mass_;           // kg
};
```

Classes

- With this declaration, `Body` now refers to a type like `int` or `double`
  - Rather than just one value associated with any instance
  - Each instance has two vectors of capacity three, and one `double`

```
// Class declaration
class Body;

// Class definition
class Body {
    public:
        // Member variables
        double position_[3];    // km
        double velocity_[3];    // km/h
        double mass_;           // kg
};
```

Accessing member variables

- We can declare a local variable to be an *instance* of this class:

```
// Class declarations
class Body;
// Function declarations
int main();

// Class definitions...

// Function definitions
int main() {
    Body earth{ {149.6e6, 0.0, 0.0}, {0.0, 107e3, 0.0}, 5.972e24 };

    return 0;
}
```

An instance of a class is also described as an *object*

```
1.496e8    position_   earth
0.0
0.0
0.0        velocity_
1.07e3
0.0
5.972e24   mass_
```

```
class Body {
    public:
        // Member variables
        double position_[3];    // km
        double velocity_[3];    // km/h
        double mass_;           // kg
};
```

Accessing member variables

- We access the member variables using the `.` operator
  - The left-hand operand must be an object
  - The right-hand operand must be the identifier of a member variable

```
int main() {
    Body earth{ {1.496e8, 0.0, 0.0}, {0.0, 107e3, 0.0}, 5.972e24 };

    std::cout << earth.mass_ << std::endl;
    std::cout << earth.position_[0] << std::endl;

    return 0;
}
```

Output:

```
5.972e+24
1.496e+08
```

```
1.496e8    position_   earth
0.0
0.0
0.0        velocity_
1.07e5
0.0
5.972e24   mass_
```

  - The memory for these seven doubles is deallocated with the function returns

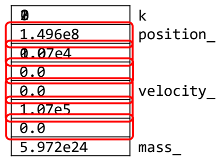
Assigning member variables

- You can also manipulate what is stored in the member variables:

```
int main() {
    Body earth{ {1.496e8, 0.0, 0.0}, {0.0, 107e3, 0.0}, 5.972e24 };

    // Approximate the distance the body moves in 0.1 h (= 6 min)
    for ( std::size_t k{0}; k < 3; ++k ) {
        earth.position_[k] += 0.1*earth.velocity_[k];
    }

    return 0;
}
```



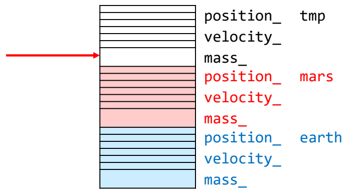
Objects as local variables

- The default initialization and assignment are the same:

```
int main() {
    Body earth{ {149.6e6, 0, 0}, {0, 107.0e3, 0}, 5.972e24 };
    Body  mars{ {0, 249.2e6, 0}, { 86.8e3, 0, 0}, 6.39e23  };

    Body tmp{ earth }; // All entries are copied over from 'earth'
    tmp = mars;        // All entries from 'mars' are copied over
    tmp.mass_ = 0.0;   // Sets only the mass of 'tmp' to 0.0

    return 0;
}
```



Objects as arguments

- Objects can be passes as arguments:

```
double speed( Body a );

double speed( Body a ) {
    return std::sqrt( a.velocity_[0]*a.velocity_[0]
                    + a.velocity_[1]*a.velocity_[1]
                    + a.velocity_[2]*a.velocity_[2] );
}
```

Objects as arguments

- Objects can be passes as arguments:

```
int main() {
    Body earth{ {149.6e6, 0, 0}, {0, 107.0e3, 0}, 5.972e24 };
    double earth_speed{ speed( earth ) };

    // 1 km/h = 0.000172603 mi/s
    double const KM_H_TO_MI_S{ 0.000172603 };

    std::cout << "That's orbiting at "
              << (KM_H_TO_MI_S*earth_speed)
              << " miles a second, so it's reckoned."
              << std::endl;  // Ref: Monty Python's "Galaxy Song"

    return 0;
}
```

Output:

```
That's orbiting at 18.4685 miles a second, so it's reckoned,
```

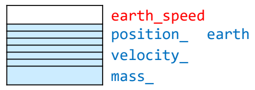
Objects as arguments

- The function `main()` has two local variables:

```
int main() {
    Body earth{ {149.6e6, 0, 0}, {0, 107.0e3, 0}, 5.972e24 };
    double earth_speed{ speed( earth ) };

    // 1 km/h = 0.000172603 mi/s
    std::cout << "That's orbiting at "
              << (0.000172603*earth_speed)
              << " miles a second, so it's reckoned."
              << std::endl;  // Ref: Monty Python's "Galaxy Song"

    return 0;
}
```

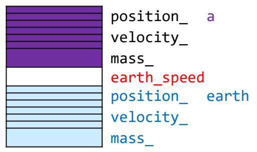


Objects as arguments

- When `speed(…)` is called, the argument is copied to the parameter

```
int main() {
    Body earth{ {149.6e6, 0, 0}, {0, 107.0e3, 0}, 5.972e24 };
    double earth_speed{ speed( earth ) };

double speed( Body a ) {
    return std::sqrt( a.velocity_[0]*a.velocity_[0]
                    + a.velocity_[1]*a.velocity_[1]
                    + a.velocity_[2]*a.velocity_[2] );
}
```

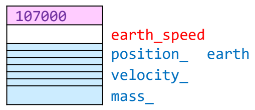


Objects as arguments

- The result is calculated put back onto the call stack:

```
int main() {
    Body earth{ {149.6e6, 0, 0}, {0, 107.0e3, 0}, 5.972e24 };
    double earth_speed{ speed( earth ) };

double speed( Body a ) {
    return std::sqrt( a.velocity_[0]*a.velocity_[0]
                    + a.velocity_[1]*a.velocity_[1]
                    + a.velocity_[2]*a.velocity_[2] );
}
```



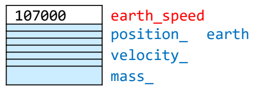
Objects as arguments

- The returned value is copied to `earth_speed` and subsequently used

```
int main() {
    Body earth{ {149.6e6, 0, 0}, {0, 107.0e3, 0}, 5.972e24 };
    double earth_speed{ speed( earth ) };

    // 1 km/h = 0.000172603 mi/s
    std::cout << "That's orbiting at "
              << (0.000172603*earth_speed)
              << " miles a second, so it's reckoned."
              << std::endl; // Ref: Monty Python's "Galaxy Song"

    return 0;
}
```



Return values

- Return values behave similarly:

```
Body center_of_mass( Body a, Body b );

Body center_of_mass( Body a, Body b ) {
    Body com{ {0.0,0.0,0.0}, {0.0,0.0,0.0}, a.mass_ + b.mass_ };

    for ( std::size_k{0}; k < 3; ++k ) {
        com.position_[k] = (a.mass_/com.mass_)*a.position_[k]
                         + (b.mass_/com.mass_)*b.position_[k];

        com.velocity_[k] = (a.mass_/com.mass_)*a.velocity_[k]
                         + (b.mass_/com.mass_)*b.velocity_[k];
    }

    return com;
}
```

Return values

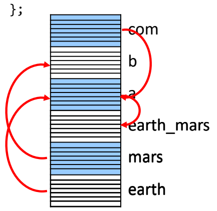
- The returned value is copied to `earth_mars` and subsequently used

```
int main() {
    Body earth{ {149.6e6, 0, 0}, {0, 107.0e3, 0}, 5.972e24 };
    Body  mars{ {0, 249.2e6, 0}, { 86.8e3, 0, 0}, 6.39e23 };

    Body earth_mars{ center_of_mass( earth, mars ) };

    std::cout << earth_mars.mass_ << std::cout;
    std::cout << "("  << earth_mars.position_[0]
              << ", " << earth_mars.position_[1]
              << ", " << earth_mars.position_[2]
              << ")" << std::endl;
    std::cout << speed( earth_mars ) << std::endl;

    return 0;
}
```



Output:

```
6.611e+24
(1.3514e+08, 2.40869e+07, 0)
97021.1
```

Passing by reference and not value

- Problem:
  - Every time an object is passed as an argument, there is potentially a significant amount of copying involved
- Solution:
  - Use pass by reference or pass by constant reference

Passing by reference and not value

- For example:

```
Body center_of_mass( Body const &a, Body const &b );

Body center_of_mass( Body const &a, Body const &b ) {
    Body com{{0,0,0}, {0,0,0}, a.mass_ + b.mass_};

    for ( std::size_k{0}; k < 3; ++k ) {
        com.position_[k] = (a.mass_/com.mass_)*a.position_[k]
                         + (b.mass_/com.mass_)*b.position_[k];

        com.velocity_[k] = (a.mass_/com.mass_)*a.velocity_[k]
                         + (b.mass_/com.mass_)*b.velocity_[k];
    }

    return com;
}
```

These refer to the actual arguments – no copies are made

The return value is still returned by value

Summary

- Following this lesson, you now
  - Are aware of the `class` keyword
  - Know how to declare and initialize objects
    - Instances of classes are also called *objects*
  - You know how to access and manipulate the member variables
  - Know how to pass objects to functions and how pass by value works
  - Understand that pass by reference works the same way as for primitive data types

References

[1] cplusplus.com
http://www.cplusplus.com/doc/tutorial/classes/

[2] Wikipedia
https://en.wikipedia.org/wiki/Class_(computer_programming)

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.