Lab # 9: Structures

EC-102 – Computer Systems and Programming

Usman Ayub Sheikh

School of Mechanical and Manufacturing Engineering (SMME),
National University of Sciences and Technology (NUST)

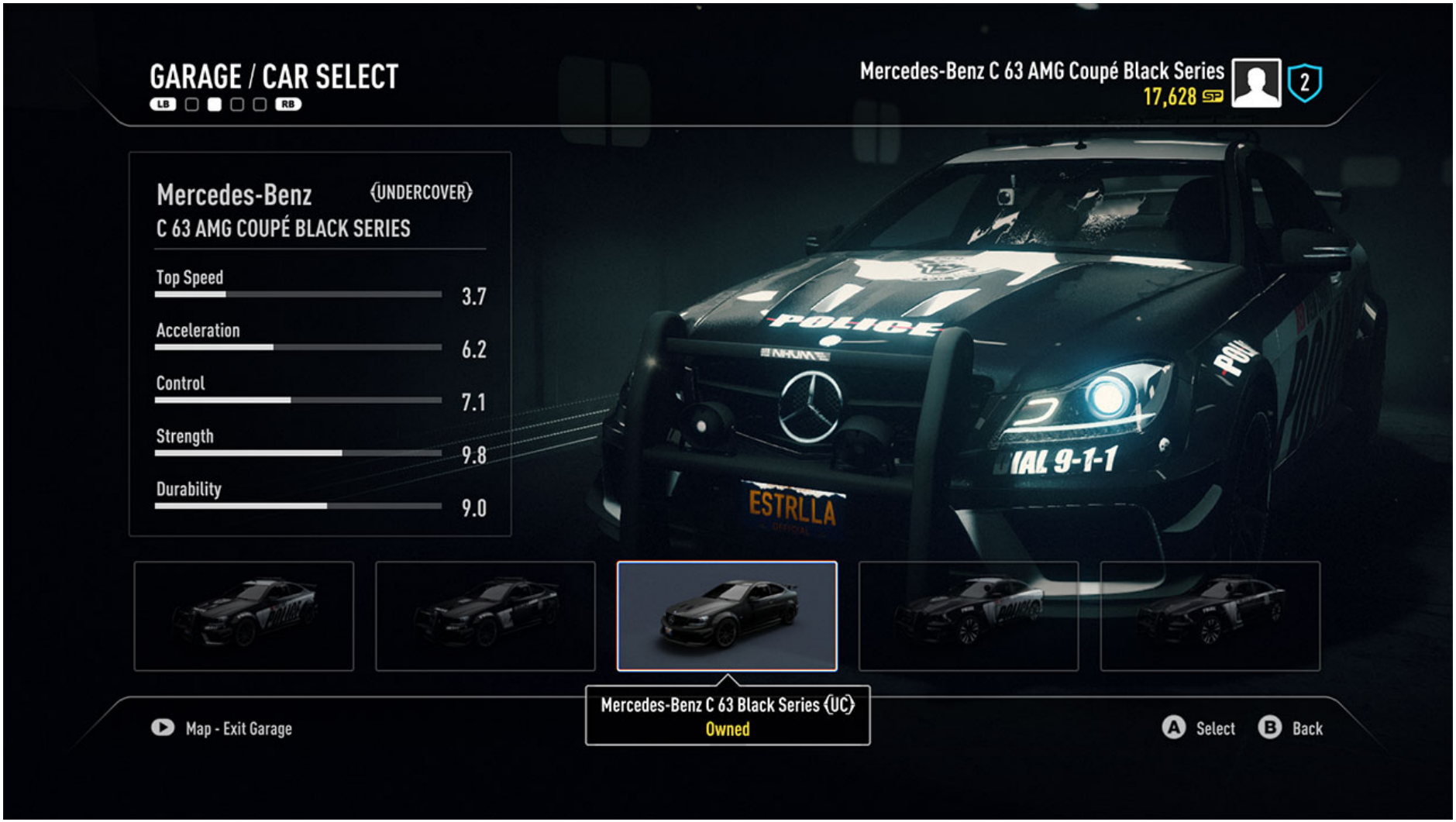
October 25, 2015

Outline

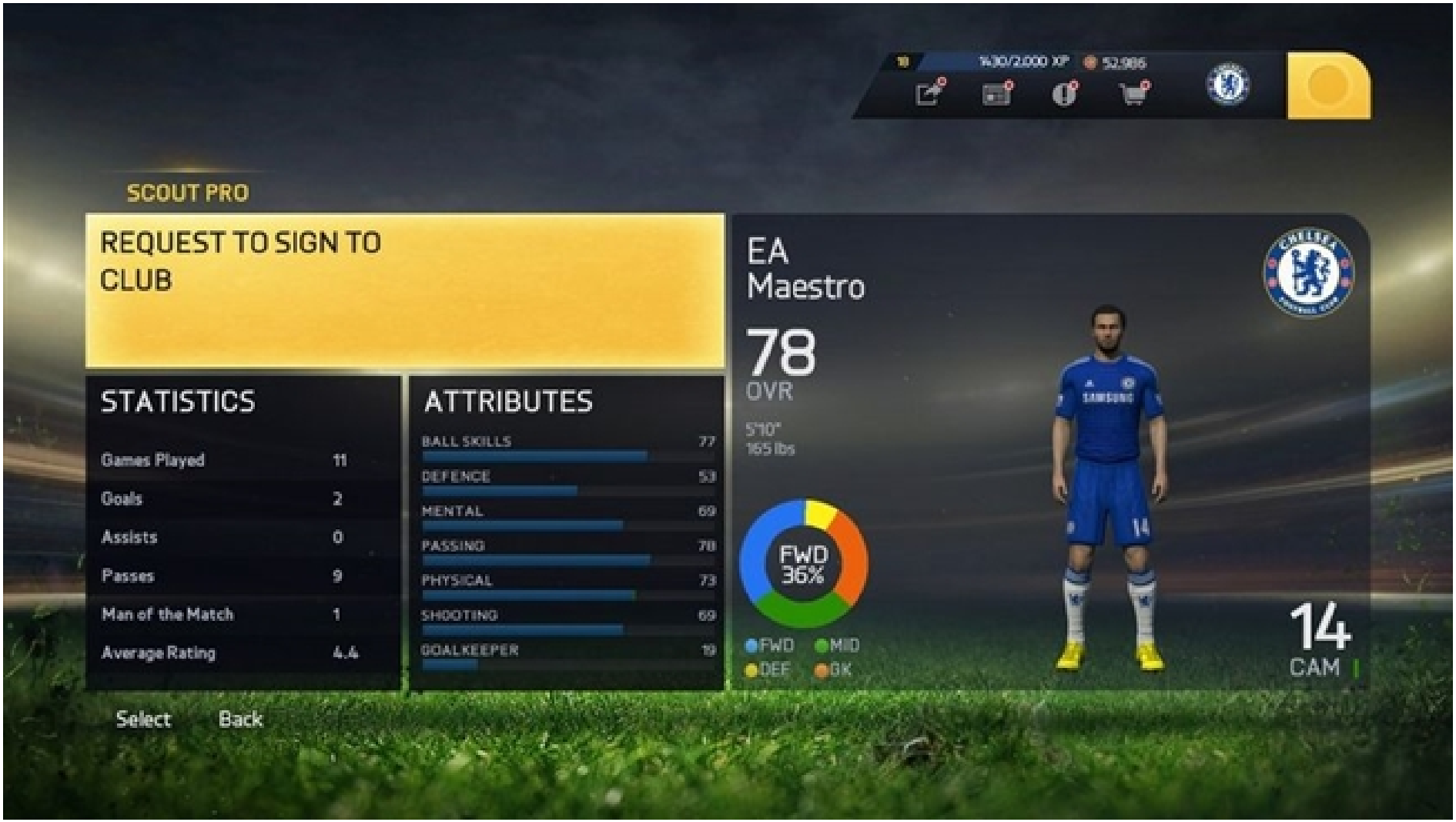
Structures – Why?

- Variables of such data types as `int`, `float` and `char` can represent, at most, one item of information e.g. a length or a width
- In the physical world, we deal with entities such as people and cars, all with their own set of attributes/characteristics such as name, size and weight etc.
- In order to model such things, we want to be able to not only
  - group all the relevant data of an entity into a single variable, but also
  - work with that variable as we work with variables of type `int` or `float`

Structures – Why?



Structures – Why?



Structures – What?

- A **collection** of simple variables
- The variables can be of **different** types, and
- Are known as the **members** of the structure
- Examples:
  - An item in a widget company's parts inventory
    - Model number
    - ID
    - Cost
  - A student of BS Mechanical Engineering
    - Name
    - Reg. No.
    - Section

Structures – How?

```
1  #include <iostream>
2  using namespace std;
3
4  struct part
5  {
6      int id;
7      float cost;
8  };
9
10 int main()
11 {
12     part part1;
13
14     part1.id = 12;
15     part1.cost = 22.57;
16
17     cout << "ID " << part1.id << " Cost " << part1.cost;
18     return 0;
19 }
```

Defining the Structure

```
1 struct part
2 {
3     int id;
4     float cost;
5 };
```

- The keyword `struct` introduces the structure definition
- Next comes the structure name
- The declarations of the structure members – id and cost – are enclosed in braces
- A semicolon follows the closing brace, terminating the entire structure

Defining the Structure

```
1 struct part
2 {
3     int id;
4     float cost;
5 };
```

- The structure definition serves **only** as a blueprint for the creation of variables of type `part`
- Unlike the definition of a simple variable, it does not set aside any memory or even name any variables
- It is merely a specification of how structure variables will look when they are defined

Defining Structure Variables

```
1 part part1;
2 part part2;
3 part part3;
```

- Line 1 defines a variable `part1` of type structure `part`, Line 2 and 3 define a few more variables of the same type
- These definitions reserve space in the memory for `part1`, `part2` and `part3` respectively
- How much space for `part1`? Enough to hold all the members i.e. `id` and `cost`
- 4 bytes for the `id` and 4 bytes for the `cost` = 8 bytes for one `part` variable

Accessing Structure Members

```
1 part1.id = 22;
2 part2.id = 23;
3 part3.id = 24;
4
5 part1.cost = 45.55;
6 part2.cost = 34.64;
7 part3.cost = 24.55;
```

- Once a structure has been defined, its members can be accessed using a **dot operator**
- The structure member is written in three parts:
  1. the name of the structure variable e.g. `part1`,
  2. the dot operator, and
  3. the member name e.g. `id`
- `part3.id` means the `id` member of `part3`

Accessing Structure Members

```
1 cout << "IDs: " << endl;
2 cout << part1.id << endl;
3 cout << part2.id << endl;
4 cout << part3.id << endl;
```

- Structure members are treated just like other variables
- In the assignment statement `part1.id = 22`, the `id` member of `part1` has been assigned a value of 22
- Similarly, `cout` statements can be used to display the `id` of each of the three parts

Other Structure Features

- Structure members can be **initialized** when the structure variable is defined

```
1     part part1 = {22, 45.55};
2     part part2 = {23, 34.64};
3     part part3 = {24, 24.55};
```

- One structure variable can be **assigned** to another variable of the same type as follows:

```
1     part1 = part2;
```

The value of each member of `part2` is assigned to the corresponding member of `part1`

Solved Example 1

```
1  // demonstrates some additional features of structures
2  #include <iostream>
3  using namespace std;
4
5  struct part
6  {
7      int modelnumber;
8      int partnumber;
9      float cost;
10 };
11
12 int main()
13 {
14     part part1 = {6244, 373, 217.55F};
15     part part2;
16
17     cout << "Model " << part1.modelnumber;
18     cout << ", part " << part1.partnumber;
19     cout << ", cost $" << part1.cost << endl;
```

Solved Example 1

```
20      part2 = part1;
21
22      cout << "Model " << part2.modelnumber;
23      cout << ", part " << part2.partnumber;
24      cout << ", cost $" << part2.cost << endl;
25
26      return 0;
27  }
```

Solved Example 2

```
1  // demonstrates structures using English measurements
2  #include <iostream>
3  using namespace std;
4
5  struct Distance
6  {
7      int feet;
8      float inches;
9  };
10
11 int main()
12 {
13     Distance d1, d3;
14     Distance d2 = {11, 6.25};
15
16     cout << "\nEnter feet: ";
17     cin >> d1.feet;
18     cout << "Enter inches: ";
19     cin >> d1.inches;
```

Solved Example 2

```
20      d3.inches = d1.inches + d2.inches;
21      d3.feet = 0;
22
23      if(d3.inches >= 12.0)
24      {
25          d3.inches -= 12.0;
26          d3.feet++;
27      }
28      d3.feet += d1.feet + d2.feet;
29
30      cout << d1.feet << "\'-" << d1.inches << "\" + ";
31      cout << d2.feet << "\'-" << d2.inches << "\" = ";
32      cout << d3.feet << "\'-" << d3.inches << "\"\n";
33
34      return 0;
35  }
```

Solved Example 3

```
1  // demonstrates structures within structures
2  #include <iostream>
3  using namespace std;
4
5  struct Distance
6  {
7      int feet;
8      float inches;
9  };
10
11 struct Room
12 {
13     Distance length;
14     Distance width;
15 };
```

Solved Example 3

```
16 int main()
17 {
18     Room dining;
19
20     dining.length.feet = 13; // nested structure member
21     dining.length.inches = 6.5;
22     dining.width.feet = 10;
23     dining.width.inches = 0.0;
24
25     float l = dining.length.feet + dining.length.inches / 12;
26     float w = dining.width.feet + dining.width.inches / 12;
27
28     cout << "Dining room area is: " << l * w << " sq ft\n";
29     return 0;
30 }
```

Exercise 1

- Create a structure called `employee` that contains two members:
  - an employee number (type `int`), and
  - the employee's compensation (in dollars, type `float`)
- Ask the user to fill in this data for three employees
- Store it in three variables of type `struct employee`, and then
- Display the information for each employee

Exercise 2

- Create a structure called `Volume` that uses three variables of type `Distance` to model the volume of a room.
- Initialize a variable of type `Volume` to specific dimensions, then,
- Calculate the volume it represents, and
- Print out the result