Structures

<u>Structure</u>: C++ construct that allows multiple variables to be grouped together.

General Format:

```
   struct <struct Name>
   {
      type1 field1;
      type2 field2;
      . . .
   };
```

Don't forget the semicolon!

Example `struct` declaration

In C++11, we *can* provide default values for struct fields

```
struct Student
{
    int studentID;
    string name;
    short yearInSchool;
    double gpa = 4.0;
};
```

structure tag

structure members (data members)

- This declaration creates a new data type, `Student`.
- `struct` declaration does **not allocate** memory or create variables
- Once `Student` is declared, it is used just **like any other data type**.

Instantiating objects from structures

```
Student john;
Student jack;
```

- Two objects (or variables) are instantiated from the Student type: `john` and `jack`. Objects occupy memory locations once instantiated.
  - Memory is allocated for each variable (object).

Alternative way to instantiate struct objects

```
struct{
        int studentID;
        string name;
        short yearInSchool;
        double gpa;
} john, jack;
```

- The lines above create a structure (without naming it) and instantiate two objects from that structure.
  - No need to declare the structure type if it is not used again for instantiating more objects.

Accessing struct data members

- Structures data members may be accessed like any other variable, for example:

```
    cout << “The GPA for John is ‟ << John.gpa << endl;
    cout << ‟ The ID for John is ‟ << John.StudentID;
```

Structure initialization

```
Student John = {000123,”John”,3, 3.5}; // inits all members

Student John{000123,”John”,3, 3.5};  // inits all members

Student John = {.name=“John”,3,3.5}; // inits name,year, gpa

Student Michael(John);                 // inits fields from John

Student John = {}; // default init, all members are zeros for
//primitive fields, default inits for non-primitives.
```

Nesting structures

```
struct Seniors{
   Student class_student;
   int graduation_data;
   int cumulative_gpa;
};
Seniors class_of_2019;
```

- In this example, the `struct seniors` has a data member that is of type `Student`.

Structs

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

struct Motorcycle {
    string brand;
    string model;
    string color;
    int cc;
};

// Emphasis on the pass by constant reference
void printMotorcycle(const Motorcycle& aBike) {
    cout << aBike.brand << ' ' << aBike.model << ' '
        << aBike.color << ' ' << aBike.cc << endl;
}

int main() {
    Motorcycle myBike;

    cout << myBike << endl; // Won't compile

    myBike.brand = "Suzuki";
    myBike.model = "Vstrom";
    myBike.color = "white";
    myBike.cc = 650;

    cout << myBike << endl; // Won't compile

}
```

Compilation error

Structs

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

struct Motorcycle {
    string brand;
    string model;
    string color;
    int cc;
};
// Emphasis on the pass by constant reference
void printMotorcycle(const Motorcycle& aBike) {
    cout << aBike.brand << ' ' << aBike.model << ' '
        << aBike.color << ' ' << aBike.cc << endl;
}
int main() {
    Motorcycle myBike;

    // The strings are "empty" and the int is whatever is sitting in
    // memory
    cout << myBike.brand << ", " << myBike.model << ", "
        << myBike.color << ", " << myBike.cc << endl;

    myBike.brand = "Suzuki";
    myBike.model = "Vstrom";
    myBike.color = "white";
    myBike.cc = 650;

    cout << myBike.brand << ' ' << myBike.model << ' '
        << myBike.color << ' ' << myBike.cc << endl;

    printMotorcycle(myBike);

    Motorcycle bike2{ "Harley Davidson", "Softtail", "Black", 1746 };
    printMotorcycle(bike2);


    myBike = bike2;
    printMotorcycle(myBike);

    Motorcycle bike3(bike2);
    printMotorcycle(bike3);

}
```

```
, , , -858993460
Suzuki Vstrom white 650
Suzuki Vstrom white 650
Harley Davidson Softtail Black 1746
Harley Davidson Softtail Black 1746
Harley Davidson Softtail Black 1746
```

Vectors of structs, reading from file

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

struct Motorcycle {
    string brand;
    string model;
    string color;
    int cc;
};

// Emphasis on the pass by constant reference
void printMotorcycle(const Motorcycle& aBike) {
    cout << aBike.brand << ' ' << aBike.model << ' '
         << aBike.color << ' ' << aBike.cc << endl;
}

int main() {
    //
    // Filling a vector from a file
    //
    cout << "========\n";
    ifstream bikeStream("bikes.txt");

    vector<Motorcycle> vm;
    string brand, model, color;
    int cc;
    while(bikeStream >> brand >> model >> color >> cc) {
        Motorcycle mot;
        mot.brand = brand;
        mot.model = model;
        mot.color = color;
        mot.cc = cc;
        vm.push_back(mot);
    }

    //
    // looping over the collection.
    // If not modifying remember to use constant reference
    //
    for (const Motorcycle& m : vm) {
        printMotorcycle(m);
    }
    cout << "========\n";
}
```

```
========
Honda CB500F Red 500
Triumph Bonneville Black 865
Yamaha FJ-09 White 847
HarleyDavidson Iron883 DeadwoodGreen 883
========
```

Vectors of structs, reading from file

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

struct Motorcycle {
    string brand;
    string model;
    string color;
    int cc;
};

// Emphasis on the pass by constant reference
void printMotorcycle(const Motorcycle& aBike) {
    cout << aBike.brand << ' ' << aBike.model << ' '
         << aBike.color << ' ' << aBike.cc << endl;
}

int main() {
    //
    // Filling a vector from a file
    //
    cout << "========\n";
    ifstream bikeStream("bikes.txt");

    vector<Motorcycle> vm;
    string brand, model, color;
    int cc;
    while(bikeStream >> brand >> model >> color >> cc) {
         Motorcycle mot{brand, model, color, cc};
        vm.push_back(mot);
    }

    //
    // looping over the collection.
    // If not modifying remember to use constant reference
    //
    for (const Motorcycle& m : vm) {
        printMotorcycle(m);
    }
    cout << "========\n";
}
```

```
========
Honda CB500F Red 500
Triumph Bonneville Black 865
Yamaha FJ-09 White 847
HarleyDavidson Iron883 DeadwoodGreen 883
========
```

Reading directly into struct members

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

struct Motorcycle {
    string brand;
    string model;
    string color;
    int cc;
};

// Emphasis on the pass by constant reference
void printMotorcycle(const Motorcycle& aBike) {
    cout << aBike.brand << ' ' << aBike.model << ' '
        << aBike.color << ' ' << aBike.cc << endl;
}

int main() {
    vector<Motorcycle> vm;
    cout << "========\n";
    ifstream bikeStream2("bikes.txt");
    Motorcycle mot;
    while (bikeStream2 >> mot.brand >> mot.model
        >> mot.color >> mot.cc) {
        vm.push_back(mot);
    }

    //
    // looping over the collection.
    // If not modifying remember to use constant reference
    //
    for (const Motorcycle& m : vm) {
        printMotorcycle(m);
    }
    cout << "========\n";
}
```

```
========
Honda CB500F Red 500
Triumph Bonneville Black 865
Yamaha FJ-09 White 847
HarleyDavidson Iron883 DeadwoodGreen 883
========
```

Vectors of structs – cont.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

struct Motorcycle {
    string brand;
    string model;
    string color;
    int cc;
};

// Emphasis on the pass by constant reference
void printMotorcycle(const Motorcycle& aBike) {
    cout << aBike.brand << ' ' << aBike.model << ' '
        << aBike.color << ' ' << aBike.cc << endl;
}

int main() {
    vector<Motorcycle&> vm;
    cout << "========\n";
    ifstream bikeStream2("bikes.txt");
    Motorcycle mot;
    while (bikeStream2 >> mot.brand >> mot.model
        >> mot.color >> mot.cc) {
        vm.push_back(mot);
    }

    //
    // looping over the collection.
    // If not modifying remember to use constant reference
    //
    for (const Motorcycle& m : vm) {
        printMotorcycle(m);
    }
    cout << "========\n";
}
```

Compilation error – you cannot have a vector of references.

- We will also learn in future lectures:
    - You can't have an "array" of references
    - You can't have "pointers" to references