Composite Data Types (C/C++)

- **Arrays**: ordered sequence of values of the same type
- **Structures**: named components of various types
  - Used to represent a relationship between values of different types
  - Example: student
    - ID Number
    - Name
    - Age
    - Major
    - Address
    
    The values are related because they belong to the same student.

Structures

- Define the student as a struct in C++:
  ```cpp
  struct Student {
    int idNumber;
    string name;
    int age;
    string major;
  };
  ```
  - Defines a new data type
  - The components are called “members” (or “fields”).
- To define a variable of type Student:
  ```cpp
  Student csStudent;
  ```

Initializing, Accessing Structures

- Struct variable can be initialized when it is defined:
  - values must be in order of struct declaration
  ```cpp
  Student student1 = {123456, "John Smith", 22, "Math"};
  ```

- Use dot notation to access members of a struct variable:
  ```cpp
  student1.age = 18;
  student2.idNumber = 123456;
  cin >> gradStudent.name;
  gradStudent.major = "Rocket Science";
  ```
Structures: operations

- **Valid operations over entire structs:**
  - assignment: `student1 = student2;`
  - function call: `showStudent(gradStudent);`
- **Invalid operations over structs:**
  - comparison: `student1 == student2`
  - output: `cout << student1;`
  - input: `cin >> student2;`
  - Must do these member by member

Arrays of Structures

- You can store values of structure types in arrays.
  ```cpp
  Student roster[40];  //holds 40 Student structs
  ```
- Each student is accessible via the subscript notation.
  ```cpp
  roster[0] = student1;
  ```
- Members of structure accessible via dot notation
  ```cpp
  cout << roster[0].name << endl;
  ```

Nested Structures

- You can nest one structure inside another.

```cpp
struct Address {
  string street;
  string city;
  string state;
  int zip;
};
struct Student {
  int idNumber;
  string name;
  Address homeAddress;
};
```
- Use dot operator multiple times to get into the nested structure:
  ```cpp
  Student student1;
  student1.name = "Bob Lambert";
  student1.homeAddress.city = "San Angelo";
  student1.homeAddress.state = "TX";
  ```

Structures as function arguments

- Structure variables may be passed as arguments to functions.

```cpp
void showStudent(Student x) {
  cout << x.idNumber << endl;
  cout << x.name << endl;
  cout << x.age << endl;
  cout << x.major << endl;
}
```
- Like regular variables:
  - structure variables are passed by value by default.
  - pass by reference can be used to change the value of a member in the function.