Data Types

- Data Type:
  - set of values
  - set of operations over those values
- example: Integer
  - whole numbers, -32768 to 32767
  - +, -, *, /, %, ==, !=, <, >, <=, >=, ...
- Which operation is not valid for float?
Data Types (C/C++)

- Scalar (or Basic) Data Types (atomic values)
  - Arithmetic types
    - Integers
      - short, int, long
      - char, bool
    - Floating points
      - float, double, long double

- Composite (or Aggregate) Types:
  - Arrays: ordered sequence of values of the same type
  - Structures: named components of various types

Structures

- 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;
};
```

- **NOTE:** semicolon after last brace!
- A struct is a data type, by convention the name is capitalized.
- The components are called “members” (or “fields”).

Structures

- So far we have defined a new data type, but we haven’t defined any variables of that type.
- To define a variable of type Student:

```cpp
    Student csStudent;
```

- Can define multiple variables of type Student:

```cpp
    Student student1, student2, gradStudent;
```
- Each one has its own set of the member variables in the Student data type
Structures

• Each variable of type student has its own set of the member variables from the Student data type

```cpp
Student student1, student2;

student1
idNumber
name
age
major

student2
idNumber
name
age
major
```

Accessing Structure Members

• Use dot notation to access members of a struct variable:

```cpp
student1.age = 18;
student2.idNumber = 123456;
cin >> gradStudent.name;
gradStudent.major = "Rocket Science";
```

• Member variables of structures can be used just like regular variables of the same type.

```cpp
student1.age++;     //happy birthday
myFunc(student2.idNumber);
if (student1.age==student2.age) {
    ...
}
```
Structures: operations

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

Structures: output

- **Output the members one at a time:**
  ```
  cout << student1.idNumber << " ";
  cout << student1.name << " ";
  cout << student1.age << " ";
  cout << student1.major << endl;
  ```
  Output:
  ```
  11122 Chris Johnson 19 Football
  ```
- **Comparing two structs:**
  ```
  if (student1.idNumber == student2.idNumber &&
      student1.name == student2.name &&
      student1.age == student2.age &&
      student1.major == student2.major)
  ...
  ```
Initializing structures

- Struct variable can be initialized when it is defined:
  
  ```
  Student student1 = {123456,"John Smith",22, "Math"};
  ```

- Must give values in order of the struct declaration.

- Can NOT initialize members in structure declaration, only variable definition:

  ```
  struct StudentA {
    int id = 123456; //ILLEGAL
    string name = "John Smith"; //ILLEGAL
  }
  ```

Arrays of Structures

- You can store values of structure types in arrays.
  
  ```
  Student roster[40]; //holds 40 Student structs
  ```

- Each student is accessible via the subscript notation.
  
  ```
  roster[0] = student1;
  ```

- Members of structure accessible via dot notation

  ```
  cout << roster[0].name << endl;
  ```
Arrays of Structures

**Arrays processed in loops:**

Student roster[40];

//input
for (int i=0; i<40; i++) {
    cout << "Enter the name, age, idNumber and " << "major of the next student: \n";
    cin >> roster[i].name >> roster[i].age >> roster[i].idNumber >> roster[i].major;
}

//output all the id numbers and names
for (int i=0; i<40; i++) {
    cout << roster[i].idNumber << endl;
    cout << roster[i].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;
};
```
Nested Structures

- Use dot operator multiple times to get into the nested structure:

  ```
  Student student1;
  student1.name = "Bob Lambert";
  student1.homeAddress.city = "San Angelo";
  student1.homeAddress.state = "TX";
  ```

- Or set up address structure separately:

  ```
  Address a1;
  a1.street = "101 Main St.";
  a1.city = "San Angelo";
  a1.state = "TX";
  a1.zip = 76903;

  student1.name = "Bob Lambert";
  student1.homeAddress = a1;
  ```

Structures as function arguments

- Structure variables may be passed as arguments to functions.

  ```
  void showStudent(Student x) {
    cout << x.idNumber << endl;
    cout << x.name << endl;
    cout << x.age << endl;
    cout << x.major << endl;
  }

  // in main:
  Student student1;

  //input information about student1 here
  showStudent(student1);
  ```
Structures as function arguments

- By default, structure variables are passed by value (like most variables).
- If the function needs to change the value of a member, the structure variable should be passed by reference.

```c
void happyBirthday(Student &s) {
    s.age++;
}
```

Returning Structure from Function

- A function may return a structure.

```c
Student inputStudent(ifstream &fin) {
    Student result;
    fin >> result.idNumber;
    fin >> result.name;
    fin >> result.age;
    fin >> result.major;
    return result;
}
```

// in main:
ifstream inFile;
inFile.open("students.dat");

Student student1 = inputStudent(inFile);
for (int i=0; i<40; i++)
    roster[i] = inputStudent(inFile);

inFile.close();

Always pass input/output streams by reference!!
Example: nested Structures

- Could have multiple structs using Address:

```cpp
struct Student {
    int idNumber;
    string name;
    float gpa;
    Address homeAddress;
    Address campusAddr;
};

struct Faculty {
    int idNumber;
    string name;
    string officeLocation;
    Address address;
};

struct GradStudent {
    int idNumber;
    string name;
    int yearGraduated;
    Address homeAddress;
    Address campusAddr;
};
```

- Could have **one** function to process Addresses

```cpp
void showAddress(Address x) {
    cout << x.street << endl;
    cout << x.city << " ";
    cout << x.state << " ";
    cout << x.zip << endl;
}
```

- Call it for different structure types with Address:

```cpp
Student st;
Faculty fac;
GradStudent gs;
//...
showAddress(st.homeAddress);
showAddress(fac.address);
showAddress(gs.campusAddr);
```
Nested Arrays and Structures

```c
struct Course {
    string course;
    int section;
    string title;
    string days;
    string time;
    string bldg;
    int roomNum;
    string instructor;
};

Student enrolledStudents[35000];

enrolledStudents[8].schedule[0].course = "CS1428";
```

Initializing arrays of structures

- Provide an initialization list for one or more of the elements in the array:

```c
Student roster[40] = {
    {123456,"John Smith",22, "Math"},
    {444555,"Lisa Simpson",18, "Biology"},
    {999999,"Tony Jackson",25, "Physics"},
    {887766,"Melissa Brown",20, "Engineering"}
};
```