Structs

grouping values

- we've done this before with arrays
- but they all have to be of the same type
- can we do the same thing with values of different types?
- e.g. grouping a string, int, float
    - string name, int daysAbsent, float grade
- we've done this with classes

definition

- structs are a structured data type
- they allow grouping of related values of different types
- this is basically synonymous with “record” in other languages or in semantics
- structs are a heterogeneous data type (different types)
    - arrays are homogeneous (same types)
- they are very similar to classes but have less “power”
    - they are typically used for simple “objects” with no methods (only attributes)

syntax

```
struct structName
{
        dataType1 identifier1;
        dataType2 identifier2;
        ...
        dataTypeN identifierN;
};
```

- note the semicolon after the close brace!
- identifiers are called members
- a struct is only a definition (not a declaration) so it does not assign memory
- we put these before function prototypes

declaration

given:

```
struct studentRecord
{
        string name;
        int id;
        float grade;
        char letterGrade;
};

studentRecord student;
studentRecord anotherStudent;
```

accessing members

```
student.name = “Joe Somebody”;
student.id = 123456;
student.grade = 94.66;
student.letterGrade = 'A';
anotherStudent.name = “Susan Sarandon”;
...
```

- the dot (.) operator is called the member access operator
- more e.g.

```
cout << student.id;
cout << “Enter grade: ”;
cin >> student.grade;
cout << “Integer part of grade: ” << (int)student.grade;
```

assignment

```
anotherStudent = student;
```

- this is valid!
- each member is copied one-at-a-time (internally)
- this is an aggregate operation that works (unlike with arrays)

comparison

- valid

```
if (student.grade >= 90.0)
if (anotherStudent.letterGrade == student.letterGrade)
```

- invalid

```
if (anotherStudent == student)
```

- so structs can only be compared “member-wise”
- no aggregate operation allowed in this case

passed as parameter

- can be passed by value or reference
- e.g.

```
void load(studentRecord&);
void print(studentRecord);
...
int main()
{
        studentRecord student;
        load(student);
}
...
void load(studentRecord& s)
{
        cout << “Enter name: ”;
        cin >> s.name;
        ...
}
...
void print(studentRecord s)
{
        cout << “Name: ” << s.name << endl;
        ...
        }
```

returned as function values

```
    studentRecord createStudent()
    {
        studentRecord s;

        s.name = “John”;
        s.id = 123456;

        return s;
    }

    studentRecord copyStudent(studentRecord s1)
    {
        studentRecord s2;

        s2.name = s1.name;
        s2.id = s1.id;

        // or just:
        s2 = s1;

        return s2;
    }
```

arrays in structs

```
    struct studentRecord
    {
        string name;
        int id;
        float grades[10];
    };

    studentRecord student;

    student.name = “John”;
    student.id = 123456;
    student.grades[0] = 100.0;
    student.grades[1] = 87.55;

    for (int i=0; i<10; i++)
        cout << student.grades[i] << endl;
```

so we can search, sort, etc arrays in structs just like we did on regular arrays
the only difference is that these arrays are just in a struct
which really makes no difference to us at all!

structs in arrays

```
studentRecord students[10];

students[0].name = “John”;
students[0].id = 123456;
students[0].grade = 98.5;
students[0].letterGrade = 'A';

// list student names
for (int i=0; i<10; i++)
        cout << students[i].name << endl;
```

structs in a struct

```
struct employeeType
{
        string firstName;
        string lastName;
        int id;
        string address1;
        string address2;
        string city;
        string state;
        int zip;
        string phone;
        string cell;
        string email;
        double salary;
};
```

too much information here
so split it up into several structs

```
struct nameType
{
        string first;
        string last;
};

struct addressType
{
        string address1;
        string address2;
        string city;
        string state;
        int zip;
};

struct contactType
{
        string phone;
        string cell;
        string email;
    };

    struct employeeType
    {
        nameType name;
        int id;
        addressType address;
        contactType contactInfo;
        double salary;
    };

    employeeType employee;

    employee.salary = 126500.00;
    employee.name.first = "Keira";
    employee.name.last = "Knightley";
    ...
    employee.address.city = "Hattiesburg";
    employee.contactInfo.cell = "123-456-7890";
    ...
    employee.salary = 123456789;
```

we can even loop through an array of employees:

```
        employeeType employees[10];
        ...
        cout << "Employee names:\n";
        for (int i=0; i<10; i++)
            cout << employees[i].name.last << ", " << employees[i].name.first << endl;
```

HANDOUT struct_worksheet