Chapter 7 - Introduction to Classes and Objects

- In the previous chapter we finished with the concept of parallel arrays where there are two or more arrays and related information is found at a specific index value.
- Q1: Would it be easy to sort the information in the three previously mentioned parallel arrays (id, age, and weight)?
- Related information can be placed in a ***structure***, which has a general format as follows:

```
struct StructName
{
  // variable declarations
};
```


struct Definition

- structs store a collection of data elements of ***different*** data types?
- For example, what if we wanted to keep the following information on a particular employee:
  - employee id
  - SS#
  - number of children
  - salary
  - citizen
- The elements have different data types, so we can't conveniently use an array. Instead we will use a struct

Structure Declaration (7.1)

To store this information:

- employee id
- SS#
- number of children
- salary
- citizen

We would begin by defining a structure :

```
struct Employ
{
  int id;
  int ssnum;
  int numchild;
  float salary;
  bool bCitizen;
};
```

Struct Terminology (7.1)

For this struct:

```
struct Employ
{
  int id;
  int ssnum;
  int numchild;
  float salary;
  bool bCitizen;
};
```

- `Employ` is the **identifier name** and a new data type.
- The individual components id, ssnum, etc. are called **members.**

Notes on Structures (7.1)

- A semicolon is required after the closing brace of the structure declaration
- The structure declaration does not create a variable
- It just tells the compiler what that structure is made of
- The struct declaration is usually placed above the main

Variable Declaration (7.1)

- As with all data types, in order to use our new data type **`Employ`** we must *allocate* storage space by *declaring* variables of this data type:

```
Employ sEngineer, sTech;
```

- This will allocate space for two variables called **`sEngineer`** and **`sTech`**, each containing the previously described members **`id`**, **`ssnum`**, etc.
- Each of these variables is a separate instance of **`Employ`**

Dot Operator (7.2)

- To access a struct member, we use the ***dot operator*** (period between struct variable name and member name).
- In the variable **`sEngineer`** of data type **`Employ`** we can make the assignments:

```
sEngineer.id = 12345;
sEngineer.ssnum = 534334343;
sEngineer.numchild = 2;
sEngineer.salary = 45443.34;
sEngineer.bCitizen = true;
```


Notes on Structures (7.2)

- You cannot output the entire contents of a struct variable by simply using its name
  - `cout << sEngineer;  // ERROR!`
- Similarly, you cannot compare two struct variables by using their name
  - `if(sEngineer == sTech)  // ERROR!`

struct Definition

- structs are ***user defined data types*** that can be used to declare variables. The variables that appear inside of the struct definition are ***members*** of the structure.

Payroll Problem

- Consider the following structure:

```
struct PayRoll
{
  int    employeeNumber;
  string name;
  double hoursWorked,
         payRate,
         grossPay;
};
```


Payroll Problem

- Declare a `PayRoll` variable `deptHead` and assign the `employeeNumber`, `name`, and `payRate` with the values `123`, `Joe Smith`, and `10.00`.

Time Problem

- Consider the following struct:

```
struct Time
{
  int hours,
      minutes,
      seconds;
};
```

- Write the C++ code that will read in a military time in the form hh:mm:ss and place hh into hours, mm into minutes, and ss into seconds. Error check to make sure that hh is in the range of 0-23, mm is in the range of 0-59, and ss is in the range of 0-59.

Displaying/Comparing structs

- Which of the following C++ statements are legal given variables time1 and time2 of type Time exist?

a)
```
cout << time1 << time2;
```

b)
```
if(time1 == time2)
   {
     cout << "times are equal";
   }
```

c)
```
cout << time1.hours;
```

d)
```
cin >> time1;
```

e)
```
cin >> time1.Hours;
```


Initializing Structs (7.3)

- Use an initializer list
  - `Employ manager(12345, 534334356, 1, 76899, true);`
- You can initialize only some of the members in a struct, but members that follow a non initialized member must also be not initialized
  - `Employ manager(12345, 534334356, 1);`
  - `Employ manager(12345,,,, true);`

Initializing Structs (7.3)

- You cannot initialize structures in the declaration

```
struct Employ
{
  int id = 12345;
  int ssnum = 534334356;
  int numchild = 1;
  float salary = 75000;
  bool bCitizen = true;
};
```

ERROR!

- Why?

Using a Constructor (7.3)

- It is possible to initialize a structure during declaration
- Use a constructor
- Constructor: A special function that can be used to construct, or set up and initialize a structure
- Looks like a regular function, but it's name is is the same name as the name of the structure

Constructor Example (7.3)

```
struct Employ
{
   int id;
   int ssnum;
   int numchild;
   float salary;
   bool bCitizen;
  Employ()
  {
    id = 0;
    ssnum = 0;
    numchild = 0;
    salary = 0;
    bCitixen = true;
  }
};
```


Constructors (7.3)

- Constructors can accept arguments

```
struct PopInfo
{
  string name;
  long population;

  PopInfo(string n, long p)
  {
    name = n;
    population = p;
  }
};
```

Constructors (7.3)

- This allows as to initialize structure variables as they are defined

```
PopInfo forestGrove("Forest Grove", 19000);
PopInfo portland("Portland", 556000);
```


Constructors (7.3)

- But, what if we didn't want to initialize the struct variable
  - `PopInfo city;`
- Adding empty parenthesis is incorrect:
  - `PopInfo city();  // ERROR!`

Employee Problem

- Create a struct called **`Employee`** that has members **`name (string), age (int), gender (char)`**. The struct is to have a constructor that initializes the name to a null string, the age to 0, and gender to F for female.
- Create a variable of type **`Employee`**.