Chapter 11 - Notes
Records (structs)

I. Records (structs)

A. Definition: A collection of a fixed number of components in which the components are accessed by name. The components may be of different data types.

1. Where an array is a homogeneous collection of the same data type, a struct is a heterogeneous collection of different data types.

2. A struct is often used to group related components of data that are of different data types.
   
   a. Example: name, address, phone number, ID, all of one person.
   
   b. The above example illustrates how grouping information about the same subject may require the use of different data types. Obviously, the name and address might have to be saved as a string while the phone number and ID could be saved as an integer.

3. The components that make up a struct are called members.

B. Syntax and Use

1. struct is a reserved word.

2. Syntax

   struct structName
   {
       dataType_1  identifier_1 ;
       dataType_2  identifier_2 ;
       :  
       :  
       dataType_n  identifier_n ;
   } ;

   a. NOTE - the semicolon at the end of the struct statement is required.
3. **Example**

```cpp
struct studentType
{
    string firstName ;
    string lastName ;
    char courseGrade ;
    int testScore ;
    int programmingScore ;
    double GPA ;
};
```

4. The above is a definition of a data type, **not** a declaration of a variable. No memory is allocated at this point.

5. Like the Enumerated type, you can declare a `struct` variable when you define the `struct`.
   
a. Example: ```cpp
struct studentType
{
    string firstName ;
    string lastName ;
    char courseGrade ;
    int testScore ;
    int programmingScore ;
    double GPA ;
};
```

6. Where is the `struct` defined in the program?
   
a. Typically, the `struct` definition is placed before the function prototypes (or function definitions, if that's how you do it) so that it becomes global and `struct` variables can be declared in any function.

b. Example: ```cpp
#include <iostream>
using namespace std;

const double PI = 3.14159 ;

struct studentType
{
    string firstName ;
    string lastName ;
    char courseGrade ;
```
structVariableName.memberName

a. Notice the dot (period) between the struct variable name and the name of the struct member. The dot is called the member access operator.

b. Example: studentType springStudent;  

   springStudent.firstName = "Gumby";
   springStudent.lastName = "Smith";

   cout << springStudent.firstName << " "
       << springStudent.lastName << endl ;

D. Assignment

1. It is permissible to assign the value(s) of one struct variable to another struct variable of the same struct type.

2. For instance, if I had the following declaration in my main program:

   studentType springStudent newStudent ;
springStudent.firstName = "Gumby";
springStudent.lastName = "Smith";
springStudent.courseGrade = 'B';
springStudent.testScore = 89;
springStudent.programmingScore = 82;
springStudent.GPA = 3.475;

I could make the following assignment statement:

newStudent = springStudent;

3. The above statement would be equivalent to the following statements:

    newStudent.firstName = springStudent.firstName;
    newStudent.lastName = springStudent.lastName;
    newStudent.courseGrade = springStudent.courseGrade;
    newStudent.testScore = springStudent.testScore;
    newStudent.programmingScore = springStudent.programmingScore;
    newStudent.GPA = springStudent.GPA;

E. Comparison (Relational Operators)

1. To compare struct variables, you must compare struct members of the same data type. It is not permissible to compare a struct variable with another struct variable.

    if ( newStudent == springStudent ) ← illegal

2. The following IS legal using members of the struct variables:

    if ( newStudent.firstName == springStudent.firstName )

    if ( newStudent.testScore <= springStudent.testScore )

    etc.

F. Input / Output

1. No aggregate input/output operations are allowed on structs (just like arrays). That means you can not input or output a whole struct at a time.

2. As a result, you must output (or input) a struct variable, one member at a time.
3. Example of Input for a \textit{struct} variable member:

\begin{verbatim}
cout << "Please input the first name: ";
cin >> newStudent.firstName ;
\end{verbatim}

4. Example of Output for a \textit{struct} variable member:

\begin{verbatim}
cout << newStudent.firstName << " " << newStudent.lastName ;
\end{verbatim}

G. \textit{struct} Variables and Functions

1. A \textit{struct} variable can be passed as a value parameter \textbf{or} a reference parameter.

2. A function \textbf{can} return a value of type \textit{struct}.

3. Example program:

\begin{verbatim}
#include <iostream>
#include <cstdlib>
#include <iomanip>
#include <time.h>
using namespace std;

struct  car  // Defining struct data type
{
  string make ;
  string model ;
  int  year ;
  double cost ;
  double currentValue ;
  string color ;
  double  mpg ;
};

car  costUpdate( car myRide )
{
  myRide.currentValue = myRide.currentValue * 1.28 ;

  return myRide ;
};
\end{verbatim}
int main ()
{
    car mySUV, mySUV2;  // declaring variable of type car

    // Initializing mySUV struct variable
    mySUV.make = "Chevy";
    mySUV.model = "S-10 Blazer";
    mySUV.year = 1984;
    mySUV.cost = 3000.00;
    mySUV.currentValue = 1000.00;
    mySUV.color = "blue";
    mySUV.mpg = 13.6;

    cout << fixed << setprecision(2);
    cout << "My " << mySUV.year << " " << mySUV.model << " cost me $";
    cout << mySUV.cost << " when I bought it."
    cout << endl;
    cout << "It was probably worth $" << mySUV.currentValue
    cout << endl << endl;

    mySUV2 = costUpdate( mySUV );

    cout << "I got my ride pimped, now it's worth $" << mySUV2.currentValue;
    cout << endl << endl;

    system("pause");
    return 0;
}

OUTPUT

My 1984 S-10 Blazer cost me $3000.00 when I bought it.  
It was probably worth $1000.00

I got my ride pimped, now it's worth $1080.00

Press any key to continue . . .
H. Arrays versus structs

1. Arrays and structs have similarities, but also some differences. The following table shows a comparison of the two:

<table>
<thead>
<tr>
<th>Aggregate Operation</th>
<th>Array</th>
<th>struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Assignment</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Input/Output</td>
<td>No (except strings)</td>
<td>No</td>
</tr>
<tr>
<td>Comparison</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Parameter Passing</td>
<td>By Reference Only</td>
<td>By Value or Reference</td>
</tr>
<tr>
<td>Function Returning a Value</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

I. Arrays in structs

1. It is permissible to have one of the members of a struct be an array.

2. Example:

```c
struct stuInfo
{
    string fName ;
    string Lname ;
    int ID ;
    int testsArray[5] ;
    double avg ;
} ;
```

   a. Now we can declare a variable of type stuInfo:

   ```c
   stuInfo newStudent ;
   ```

   b. And we can then initialize the struct variable with values in the following manner:

   ```c
   newStudent.fName = "Jethro" ;
   newStudent.LName = "Bodine" ;
   newStudent.ID = 12345 ;
   ```
newStudent.testsArray[0]  =  86 ;  **
newStudent.testsArray[1]  =  79 ;  **
newStudent.testsArray[2]  =  91 ;  **
newStudent.testsArray[3]  =  84 ;  **
newStudent.testsArray[4]  =  89 ;  **
newStudent.avg =  0.0 ;

c.  ** Notice how the array in the struct is accessed. The struct variable is written, followed by a dot ( period ), followed by the name of the array member in the struct, followed by the square braces with the position (index) inside.

J.  structs in Arrays

1. Multi-dimensional arrays can be useful storing related data, but the data would all have to be of the same type. With an array of structs, it's possible to simulate a multidimensional array of homogeneous types. This comes in handy if your program requires many struct variables.

2. C++ allows the programmer to create an array of data type struct of which the programmer can define.

3. Here's how it works:

a. First you must define the struct data type:

Example:  
```c
struct employeeType 
{
    string  firstName ;
    string  lastName ;
    int      ID ;
    double yearlySalary ;
    double monthlySalary ;
    double yearToDatePaid ;
} ;
```

b. Because we may have many, many employees that all require the same information, we can declare an array to hold the struct data type we have just defined.

Example:  
```c
employeeType   empArray[ 1000 ] ;
```
c. Now that we have created an array of employeeType data structures, we need to be able to access the individual array positions and the members of the struct inside that array position.

Here's how:

```cpp
empArray[0].firstName = "Jerry" ;
empArray[0].lastName = "Lewis" ;
empArray[0].ID = 12345 ;
empArray[0].yearlySalary = 34550.40 ;
empArray[0].monthlySalary = 2879.20 ;
empArray[0].yearToDatePaid = 8637.60 ;
```

d. Notice that in the above code, each member of the struct data type has been accessed in the first position of the array (position zero). It is not hard to see how a whole array of structs could be initialized from an input file using a FOR Loop.

Example:

```cpp
for ( i = 0 ; i < 1000 ; i++ )
{
    cin >> empArray[ i ].firstName ;
    cin >> empArray[ i ].lastName ;
    cin >> empArray[ i ].ID ;
    cin >> empArray[ i ].yearlySalary ;
    cin >> empArray[ i ].monthlySalary ;
    cin >> empArray[ i ].yearToDatePaid ;
}
```

K. **structs within a struct**

1. C++ even allows the programmer to put a struct inside another struct.

2. Don't forget that like variables, functions and structs must be declared (or defined) before they can be used. Therefore, a struct being used inside another struct definition must be previously defined.

3. Let's look at a simple example:
struct nameType
{
    string first;
    string middle;
    string last;
};

struct employeeType
{
    nameType name;            // This is now a struct w/in a struct
    int empID;
    string dept;
};

4. Now let's look at how we access the members of a struct within a struct.

Example:

employeeType newEmp;        // Declaring struct variable

newEmp.name.first = "John";
newEmp.name.middle = "Paul";
newEmp.name.last = "Jones";

5. Notice that between the struct variable newEmp and the struct inside called name there is a dot (period), and between the word name and the member called first, there is another dot (period). This is how embedded struct members are accessed.