Chapter 10.3 – 10.6

Structured Types

Structured Data Types

- **Simple DataType** - A single data item - it cannot be broken down into component parts. Example is int num, float avg
- **Structured DataType** - A DataType in which each value is a collection of component items.
- **Component item** - may be a simple DataType or another structured DataType
- A structured DataType gathers together a set of component items and usually imposes a specific arrangement on them

- A **record** is a heterogeneous structured DataType. Heterogeneous means that individual components of a record can be of different DataTypes
- Each component of a record is called a **field** of the record, and each field is given a name called the **field name**
Structures

- In C++, a record is called a structure. A structure DataType has a fixed number of heterogeneous components that are accessed by name.
- The fields of a structure are called members.

**SyntaxTemplate: StructDeclaration**

```cpp
struct StructTypeName
{
    MemberList
}; // note, a struct declaration ends in a semicolon
```

- **StructTypeName** is an identifier giving a name to the DataType being declared by the reserved word `struct`—
  - in other words, a new DataType is created for use in the program.
- To declare a variable of the struct DataType use:

  ```cpp
  StructTypeName identifier
  ```

- A **struct declaration** is a type declaration (discussed in chapter 10), and you still have to declare/define variables of this type for any memory allocations.
- Member names within a struct **must be unique**.
Structures (continued)

- **Member Selector** - The expression used to access components of a **struct variable**. It is formed by using the struct variable name and the member name separated by a period (.)
  
  `StructVariable.MemberName`  
  `StructVariable.MemberName.MemberName2` // MemberName is a struct

- A selected member is treated just like any other variable of the same **DataType** (as it is defined/declared in the **struct**)

---

```c
// Declare (no memory allocation) a structure for a Customer record  
struct CustomerRecord // CustomerRecord is the new TypeName created  
{
  string name; // Start of Member list
  int custId;
  int acctId;
  string lastPayDate;
  float balance;
};
```

- The above structure has 5 members. No memory is allocated until a variable declaration (which is usually a definition - memory is allocated) is made:
  
  `CustomerRecord customer1; // defines customer1 to be of DataType CustomerRecord`  
  `CustomerRecord customer2; // defines customer2 to be of DataType CustomerRecord`

See program **Structures_01.cpp** for an example of this structure
Struct Example (continued)

- The following operations are valid:
  - customer2 = customer1; // copies one struct to another
  - customer1.name = customer2.name; // member assignment
  - customer1.name = “Ron”; // member assignment
  - customer1.balance = 134.56; // member assignment
  - customer2.acctId = 5; // member assignment
  - while (customer2.acctId > 0)

- Anywhere you use a string, float, int, etc you can use a structure member (StructVariable.MemberName) of the same DataType

Aggregate Operations on Structs

- An **aggregate operation** is one that manipulates the **struct** as an entire unit, and not on an individual member of the **struct**

- **Allowable aggregate operations on a struct**
  - **Assignment** - can assign one struct variable to another
  - **Pass as a function argument** - by value or reference
  - **Returned as the function value** of a value-returning function

- **Aggregate operations NOT allowed on a struct**
  - **Arithmetic** - cannot multiply one **struct** variable by another **struct** variable
  - **Comparison** - cannot compare one **struct** variable with another **struct** variable. Comparisons are made on a member by member basis.
  - **Input/Output** - cannot use Input/Output on the entire structure at one time. **A struct variable must have the members accessed one member at a time for input/output**
Struct Example (continued)

The following shows passing of a struct as a reference parameter

```cpp
#include <iostream>
using namespace std;

// structure CustomerRecord, as declared on slide 6, goes here
void my_function (CustomerRecord& );  // function prototype

int main()
{
    // declare and define the identifier customer as a variable
    // of DataType CustomerRecord which is a structure
    CustomerRecord customer;
    my_function (customer);  // function call
    return 0;
}

void my_function (CustomerRecord& cust1)// function definition
{
    cust1.name = "Farmer Joe";  // and other statements in the function
}
```

See program Structures_01.cpp for an example of this program

More About Struct Declarations

- A more complete SyntaxTemplate for structures

  ```cpp
  struct [TypeName]
  {
      [MemberList]
  } [VariableList];
  ```

- TypeName and VariableList are optional. TypeName is the
  name of the struct being declared, and VariableList is a list of
  variable names to be declared

- Omitting the TypeName makes the struct anonymous type,
  and no variables (other than those in the VariableList) can be
  declared of that type – usually not a good idea
Struct Example -
The following two segments have the same result

**Segment #1:**
```c
struct Student
{
    string name;
    int age;
    string classYear;
    float gpa;
}
Fred, temp;
```

**Segment #2 (preferred method):**
```c
struct Student
{
    string name;
    int age;
    string classYear;
    float gpa;
} ;
Student Fred, temp;
```

Fred and temp are variables of the **struct** DataType Student

See program **Structures_02.cpp** for an example of this idea

---

**Struct Example (Similar to slide 9)**

```c
#include <iostream>
#include <string>
using namespace std;

struct TestInfo // declaration for structure TestInfo
{
    int machine_id;
    string test_name;
    string operator;
};
TestInfo GetInfo(); // function prototype

int main()
{
    TestInfo machine1; // definition of variable Machine1 as a
                        // structure of TestInfo
                        // structure of TestInfo
    machine1 = GetInfo();
}
TestInfo GetInfo()
{
    TestInfo machine;
    // statements to enter in the member values.
    // For example, machine.machine_id = 256;
    return machine;
}
```
Hierarchical Records

- **Hierarchical record** - a record in which at least one of the components is itself a record.

- **Hierarchical structure** is then a structure in which at least one of the components is itself a structure.

- To access the components of a hierarchical structure, build the accessing expressions (member selectors) for the members of the embedded structs from left to right beginning with the top struct variable name:
  - `TopStructVariable.MemberType1.MemberName`
  - `MemberType1` is another structure variable that has `MemberName` as one of its members.

- **Excellent example** in the book on page 486 - 488 – figure 10-6 on page 488

Hierarchical Struct example

See program Structures_03.cpp for an example of this program.

```cpp
struct Date{
    int day; // 1 - 31  // simple data type
    int month; // 1-12  // simple data type
    int year;  // 1900 -  // simple data type
};

struct Shirt {
    int neck_size;     // simple data type
    int sleeve_length; // simple data type
    int sleeve_type;  // simple data type
    Date purch_date;  // purch_date is a structure of data type Date
};

struct Clothes {
    Shirt tops;  // shirt is a structure of data type Shirt
    int quantity; // simple data type
};

int main()
{
    Clothes ron;  // ron is a structure of data type Clothes
                 //To access the purchase month of a shirt:
    cout << ron.tops.purch_date.month;
    //To access the neck size of a shirt:
    cout << ron.tops.neck_size;
}
```
Unions

- A **union** is a **struct** that holds a value for one of its members only at any given time during program execution.
- Declaring a union is the same as declaring a **struct** except use the reserved word **union** instead of **struct**.
- Furthermore, unions cannot contain string variables for members.

```c
union UnionNameType // union declaration
{
    char Middle_initial;
    int Id;
};
UnionNameType person // person is a variable of DataType UnionNameType
```
- A union represents only a single value from among several potential values. The assumption is that the program does not need more than one member simultaneously.

See program Structures_04.cpp for an example of this concept

---

Using Structures vs. Not Using Structures

- What are the differences between using variables and using structures in functions?
  - **Function Prototypes** (and therefore definitions)
    - void GetInfo1(Time& myTime); // Time is a structure data type
    - void GetInfo2(int& hours, int& minutes, int& seconds);
  - **Function Calls**
    - GetInfo1(time); // Uses variable declaration Time time;
    - GetInfo2(hours, minutes, seconds); // uses variable declarations of
      // int hours, int minutes, int seconds
  - **Reading and/or Outputting values**
    - With a structure:
      ```c
      cout<<time.hours<<"."<<time.minutes<<"."<<time.seconds<<endl;
      ```
    - With variables:
      ```c
      cout << hours << "." << minutes << "." << seconds << endl;
      ```

See program Structures_05.cpp for an example of this concept