Structured Data in C++

Highlights of this lab:

In this lab, you will:

- Learn the Syntax of C++ **struct**ures.
- Structures and Functions
- Arrays of Structures

Lab Exercise:

- You will work with some program example to ensure a working knowledge of C++ structures.

Click the little computer above for a detailed description.

If you **need help** during the week, you are welcome to go to CL119 during the Office Hours posted for lab instructors.

"New" Unix/Linux Commands

Here are some more Unix/Linux commands that you need to learn. Remember that you can enter **man *command*** to get a complete description of any Unix/Linux command and its options.

Command	Description
`cal` *(month #) year*	Prints a calendar of the specified year. e.g. **cal 2010** If a month number is specified, prints only that month. e.g. **cal 3 2010** (for March 2010)
`cat` *file1 [file2 ...]*	Concatenate (join together) specified files and direct the output to the standard output device - the screen. This command is commonly used to display the contents of one file on the screen. (It's simpler than getting in and out of an editor.)
`date`	Print the current time and date.
`who`	Lists who is logged into a machine. It provides information such as the user's login name and the time when the user logged on.
`w`	Lists who is logged into a machine. Provides information such as the user's login name and the time when the user logged on. It also provides information about what the user is currently doing.
`sort`	Sorts the input stream or the contents of files. To sort the contents of a file, use **sort filename**.
`wc`	Displays the number of lines, words and characters in a file. To display only the number of lines, you can use **wc -l**.
`file` *file*	Perform tests on a file to determine its type. Useful if you want to make sure a file is not an executable before you try to edit it.
`cmp` *file1 file2*	Compare two files to see if they are the same. Reports just the first difference unless you specify *-l*
`diff` *file1 file2*	Displays the differences between *file1* and *file2*. This lists the changes necessary to convert *file1* to *file2*.
`find` *path option*	Search down directories for a file. e.g. **find . -name gold.cpp** would search in the current directory and in all subdirectories for the file called *gold.cpp*
`grep` [option] *string* [file(s)]	Search for a string pattern in a file. There are several options, e.g. **grep -namespace *.cpp** would search the current directory for the string "namespace" in all *.cpp* files and show the lines in each file where the string occurs. e.g. **grep -n -namespace *.cpp** would perform the same search but also give the line numbers in which the string was found.
`ps`	Lists the processes that are running for a terminal. To see all the processes that are running for you, use **ps -fu yourusername**. This command is often used with kill.
`kill` [option] *processid*	Kill the process specified, e.g. **kill -9 1455** would perform a "sure kill" (option 9) on process id "1455". This is a handy command if you change your mind after sending a job to the printer and want to delete it from the queue. See the **lpq** command to see how you can query the print queue for process ids.
`lpq -P`*[printername]*	Query the specified printer to see active jobs. Reports process ids of jobs. e.g. **lpq -Pcl122**
`quota -v`	Show how much disk space you are using ("usage") on a multi-user Unix system and what your limit is ("quota"). The numbers given refer to kilobytes of space.

Notes:

- Some commands such as sort, cat, and wc will accept input from the keyboard.
  If you type these commands, without specifying an argument, your command prompt will not return until you press CTRL-d (which indicates an end of file on Unix)

Learn the Syntax of C++ - *Structures*.

What is a C++ Structure?

When you start reading about C++ Structures, don't get distracted by all the terminology. You'll read that the traditional terms `records` and `fields` are akin to `structures` and `members` in C++ structures.

If this is meaningful to you, congratulations. If not, ignore it, and just think of ***structures*** as ***containers*** for various pieces of information about an object. You're going to see an object called `Student` in these notes. This is a user-defined object that contains three types of related information about a student:

Name	string
ID	int
Marks	int [3]

Defining a Structure

As a programmer, you need to define such a structure. The general syntax is:

```
    struct type_name
    {
        member_list   // these are standard C++
                      // variable declarations
    };    // note the closing semi-colon
```

The *`member_list`* is where you describe the types of data that are to be associated with the object that you are defining. For example, the member list for `Student` is:

```
    string name;
    int id;
    int mark[3];
```

Putting this inside the `struct` definition, we see:

```
struct Student
    {
        string name;
        int id;
        int mark[3];
};
```

This code just defines the format of the structure. In order to start using this particular structure you need to declare an *instance* of it. This is similar to defining a variable of a predefined type such as `int`.

Declaring a Structure *Instance*

To create an instance of a container - a structure - you need to declare it, just as you would declare an instance of a primitive data type. Following the example referred to in the previous section, the statement

```
    Student    stu;
```

declares an instance called `stu` of the structure called `Student`. Note that the general syntax would be:

```
    datatype    variable_name;
```

Think back to how you declare an instance of an `integer` . i.e. `int i; // datatype variable_name;`

Once you have defined a structure and declared an instance of that structure, you are ready to start accessing the data elements associated with the instance. These are also referred to as the `members` of that instance.

Using an Instance of a Structure

Again referring back to our little example, suppose you wanted to reference the `id` value in the `stu` instance. How does the C++ compiler know that the variable `id` is associated with the instance of a structure? The answer is that you, the programmer, must provide that information using *`dot notation`*. Notice the **`dot`** in the general syntax: *`structure_instance.member_name`*

In our example, to refer to `id` you would need to specify: `stu.id`

Now, let's have a look at the whole program example and see how the parts and pieces all fit together.

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

struct Student
{
    string name;
    int id;
    int mark[3];
};

int main ()
{
    Student stu;
    int i;

    cout << "Your name, please: ";
    getline(cin, stu.name);   // command to input a string
    cout << "Your id is: ";
    cin >> stu.id;
    cout << "Enter your marks for three tests," << endl;
    for (i = 0); i < 3; i++)
    {
        cout << "Test " << i+1 << ": ";
        cin >> stu.mark[i];
    }

    cout << endl;
    cout << "Hello, " << stu.name << endl;
    cout << "Your Student ID is " << stu.id << endl;
    cout << "Your marks are: " << endl;
    for (i = 0; i < 3; i++)
    {
        cout << "Test " << i+1 << ": " << stu.mark[i] << " " << endl;
    }
}
```

This is well and fine, but consider if you want to have several students and print out their values. Half of the code from above would be duplicated to print another student. Rather than doing that, let us create a function that uses a `Student` argument.

Structures and Functions

Functions are useful if you are going to have several instance of a `Student` and you want to apply the same operations on them. Take note of how we have declared the following functions.

1. What are the return types?
2. What are the arguments?
3. How have we initialized the second instance of `Student`?
4. Is `main` shorter or longer than the code from above?

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

struct Student
{
    string name;
    int id;
    int mark[3];
};

void printStudent(const Student& c);
Student readStudent();

int main ()
{
    Student stu;
    Student stu2={ "Tom Hinks", 789000111, 88, 83, 81 };

    stu=readStudent();
    printStudent(stu);
    cout << endl << "Another Student:" <<endl;
    printStudent(stu2);
}

void printStudent(const Student& c)
{
    int i;

    cout << endl;
    cout << "Name: " << c.name << endl;
    cout << "ID: " << c.id << endl;
    for (i = 0); i < 3; i++)
    {
        cout << "Test " << i+1 << ": " << c.mark[i] << " " << endl;
    }
}
Student readStudent()
{
    int i;

    Student tempStu;
    cout << "Name?: ";
    getline(cin, tempStu.name);   // command to input a string
    cout << "ID: ";
    cin >> tempStu.id;
    for (i = 0); i < 3; i++)
    {
        cout << "Test " << i+1 << ": ";
        cin >> tempStu.mark[i];
    }
    return tempStu;
}
```

Arrays of Structures

If you want a to represent a classroom of students, you can do that by declaring an array of students. It might look something like this:

```
int main ()
{
    Student stuArray[NUM_STU];
    int i;

    for (int i = 0; i<NUM_STU; i++)
    {
        stuArray[i]=readStudent();
        cin.ignore(256,'\n'); //why do this?
    }
    for (int i = 0; i<NUM_STU; i++)
    {
        printStudent(stuArray[i]);
    }
    return 0;
}
```

Note that `NUM_STU` is declared as a constant which can be easily changed to the number of students that you have.

Lab Exercise -- C++ Structures

Objective:

The purpose of the program is to:

1. Write functions to read in and print an Employee struct
2. Write an additional function that passes in an array of Employees and returns the index to a found id.

Step 1:

- Create an `Employee.cpp` file to go with the existing `Employee.h`. This involves defining the two functions:
  - `void printEmployee(const Employee& e)` --print the values of the Employee structure (each record on one line)
  - `Employee readEmployee()` --ask the user for input and return the Employee structure
- Remember, you can compile and link the three files with the following commands:
  - `g++ -c main.cpp`
  - `g++ -c Employee.cpp`
  - `g++ main.o Employee.o -o output`
- At this point, your run should look like the following:

```
Employee Name?: Mary Contrary
Employee Id?: 1222222
Employee Yearly Salary?: 32003.33
Employee Name?: Sue Morgan
Employee Id?: 2444444
Employee Yearly Salary?: 78087.88
Employee Name?: Tom Hinks
Employee Id?: 6777777
Employee Yearly Salary?: 89018.38
Employee Name?: Jack Sprat
Employee Id?: 8999999
Employee Yearly Salary?: 42018.09
Employee Name?: Scott Burns
Employee Id?: 3444444
Employee Yearly Salary?: 22002.89
The Employee info is: Mary Contrary, 1222222, $32003.33
The Employee info is: Sue Morgan, 2444444, $78087.88
The Employee info is: Tom Hinks, 6777777, $89018.38
The Employee info is: Jack Sprat, 8999999, $42018.09
The Employee info is: Scott Burns, 3444444, $22002.89
```

Step 2:

- Add this new function prototype to Employee.h:
  `int findEmployee(const Employee array[],int tId,int num);`
- Define the findEmployee function in Employee.cpp the following are the specifications:
  Arguments:
  - `array`: array of `Employee`(s)
  - `tId`: the id you are searching for
  - `num`: the size of the array

  Returns:
  - -1 if the id is not found
  - an index to the array of Employees where the id was found
- Add code in main to call the `findEmployee` function and print out the corresponding name to go with the id.
- Sample output is:

```
Employee Name?: Mary Contrary
Employee Id?: 1222222
Employee Yearly Salary?: 32003.33
Employee Name?: Sue Morgan
Employee Id?: 2444444
Employee Yearly Salary?: 78087.88
Employee Name?: Tom Hinks
Employee Id?: 6777777
Employee Yearly Salary?: 89018.38
Employee Name?: Jack Sprat
Employee Id?: 8999999
Employee Yearly Salary?: 42018.09
Employee Name?: Scott Burns
Employee Id?: 3444444
Employee Yearly Salary?: 22002.89
The Employee info is: Mary Contrary, 1222222, $32003.33
The Employee info is: Sue Morgan, 2444444, $78087.88
The Employee info is: Tom Hinks, 6777777, $89018.38
The Employee info is: Jack Sprat, 8999999, $42018.09
The Employee info is: Scott Burns, 3444444, $22002.89
Enter an id to look for: 8999999
Found Employee: Jack Sprat
```

  By contrast, if the employee id was not found, you should display a message of:

```
Did not find an Employee with that Id!
```