Structures in C++

CS 16: Solving Problems with Computers I
Lecture #14

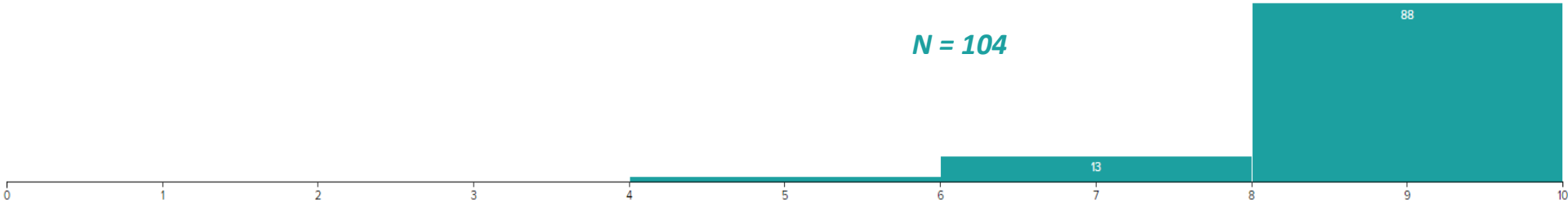
Ziad Matni
Dept. of Computer Science, UCSB

```
int main(int argc, char *argv[])
{
    if (argc > 1)
        filename = argv[1];
    ifstream setIn(filename);
    ifstream vecIn(filename);
    set<string> wordSet = getWordSet(setIn);
    vector<string> wordVec = getWordVec(vecIn);
    map<string, string> wordMap = generateMap(wordVec);

    string name = filename.substr(0, filename.size() - 4);
    string setFilename = name + "_set.txt";
    string vecFilename = name + "_vec.txt";
    string mapFilename = name + "_1_1.txt";

    // Writes set file
    ofstream setOut(setFilename);
    for (set<string>::iterator it = wordSet.begin(); it != wordSet.end(); it++)
    {
        setOut << *it << endl;
    }
    setOut.close();

    // Writes vector file
    ofstream vecOut(vecFilename);
    for (int i = 0; i < wordVec.size(); ++i)
    {
        vecOut << wordVec[i] << endl;
    }
    vecOut.close();

    // Writes to map
    ofstream mapOut(mapFilename);
    printMap(wordMap, mapOut);
    mapOut.close();

    // Generate and print random string
    string str = "";
    for (int i = 0; i < 100; i++)
    {
        cout << wordMap[str] << " ";
        str = wordMap[str];
    }
    cout << endl << endl << endl;

    // Generate more intelligent map
    map<string, vector<string>> wordVecMap;
    str = "";
    for (int i = 0; i < wordVec.size(); i++)
    {
        wordVecMap[str].push_back(wordVec[i]);
        str = wordVec[i];
    }
```

Administrative

- New lab (#7) and new homework (#7) are out!
- Homework 6 was due yesterday
  - Lab 6 is due Thursday
- Quiz 7 is on Friday
- Announcement about next week…

Quiz 6



- Mean: **8.9/10**
- Median: **9/10**

- Q1 was a “freebie” due to bad answer display on Gradescope… (worth 2 pts)

Question 3

Note that the deciding factor as to what determines descending or ascending order is the “if” statement:

```
array[j - 1] < array[j]
```

means “look for descending trend”

```
array[j - 1] > array[j]
```

means “look for ascending trend”

Best way (but not *only!*) to combine these with a Boolean var is:

```
if ( ( array[j - 1] < array[j] && descending )
    || ( array[j-1] > array[j] && !descending ) )
```

- Took off <u>1 point</u> for *correct* answer that was not optimized into 1 line.

Consider the Bubble Sort function definition introduced to you in Lecture 11:

```
void bubbleSort(int array[], int size)
{
  int temp;
  for (int i = size-1; i >= 0; i--) {
    for (int j = 1; j <= i; j++) {
      if (array[j-1] > array[j]) {
        temp = array[j-1];
        array[j-1] = array[j];
        array[j] = temp;
      } // if
    } // for j
  } // for i
}
```

This function only sorts the integer array argument in one direction. Let us consider a

ction, a boolean

e, the array

You will earn ALL 6 points ONLY if you are able to do this *without changing the number of statements* from the original function (partial credit applies otherwise). I won't take off points for styling, but keep your answer legible, please.

<u>Other cool & unconventional answers:</u>

```
if (array[j-1] < array[j] == descending)
```

```
if (array[j - !descending] > array[j - descending])
```

Lecture Outline

- Structures in C++
  - Definition
  - Use
  - Naming
  - Use with Functions
  - Initializations

- In the next lecture: Classes

Classes and Structures

- A **class** is a complex data type that *can* have
  - Multiple values within it and multiple "member functions"
  - Example: string class has member functions **.size()**, **.erase()**, **.find()**, etc…

- **Structures** are a good introduction to **classes**

- We'll learn to use them as "containers" (**objects**) for variables of possibly different types
  - Without member functions (for now… we'll deal w/ those in **classes**)
  - We'll call these **member variables**

Structures for Data

- These multiple values are logically related to one another and come together as a single item
  - Examples:
    A bank investment account (aka "CD") which has the following values:
    a balance
    an interest rate
    a term (how many months to maturity)

    What kind of values should these individually be?!

  - A student record which has the following values:
    the student's ID number
    the student's last name
    the student's first name
    the student's GPA

    What kind of values should these individually be?!

The CD Structure Example: Definition

- The Certificate of Deposit ("CD") structure (CDAccount) can be defined as

```
struct CDAccount
{
    double balance;         // a dollar amount
    double interest_rate;   // a percentage
    int term;               // a term amount in months
} ;
```

Remember this semicolon!

- Keyword **`struct`** begins a structure definition
- **`CDAccount`** is the structure ***tag*** – this is the structure's **type**
- Member names are ***identifiers*** declared in the braces

Using the Structure

- Structure definition generally placed ***outside*** any function definition
  - Including outside of **main( )**
    - So they can be globally used
  - This makes the structure type available to all code that follows the structure definition (i.e. global)
  - But CAN it be placed inside of a function, like main()?
    - Yes, but it will be local in that block only

- To declare two variables of type **`CDAccount`**:

```
CDAccount  my_account, your_account;
```

`my_account` and `your_account`
contain *distinct* member variables **`balance`**, **`interest_rate`**, and **`term`**

Specifying Member Variables

- Member variables are specific to the structure variable in which they are declared

- Syntax to specify a member variable (note the '**.**')
  *Structure_Variable_*Name **.** *Member_Variable_Name*

- Given the declaration:

```
CDAccount   my_account;
```

- Use the **dot operator** to specify a member variable, e.g.

`my_account.balance`	*is a double*
`my_account.interest_rate`	*is a double*
`my_account.term`	*is an int*

DEMO

structDemo.cpp

```
//Program to demonstrate the CDAccount structure type.
#include <iostream>
using namespace std;

//Structure for a bank certificate of deposit:
struct CDAccount
{
    double balance;
    double interest_rate;
    int term;//months until maturity
};

void get_data(CDAccount& the_account);
//Postcondition: the_account.balance and the_account.interest_rate
//have been given values that the user entered at the keyboard.
```

Note the struct definition is placed before main()

The usual way this is done

```
int main()
{
    CDAccount account;
    get_data(account);

    double rate_fraction, interest;
    rate_fraction = account.interest_rate/100.0;
    interest = account.balance*rate_fraction*(account.term/12.0);
    account.balance = account.balance + interest;

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "When your CD matures in "
         << account.term << " months,\n"
         << "it will have a balance of $"
         << account.balance << endl;
    return 0;
}
```

Note the declaration of CDAccount

We are going to "fill in" the data structure "account" using the get_data function…

Note the calculations done with the structure's member variables

Note that the structure is passed into the function as call-by-reference.

You can also pass a structure call-by-value.

```
//Uses iostream:
void get_data(CDAccount& the_account)
{
    cout << "Enter account balance: $";
    cin >> the_account.balance;
    cout << "Enter account interest rate: ";
    cin >> the_account.interest_rate;
    cout << "Enter the number of months until maturity\n"
         << "(must be 12 or fewer months): ";
    cin >> the_account.term;
}
```

Sample Dialogue

```
Enter account balance: $100.00
Enter account interest rate: 10.0
Enter the number of months until maturity
(must be 12 or fewer months): 6
When your CD matures in 6 months,
it will have a balance of $105.00
```

Note the use of the structure's member variables with an input stream.

Duplicate Names

- Member variable names duplicated between structure types are **not** a problem

```
struct FertilizerStock
{
    double quantity;
    double nitrogen_content;
};
…
FertilizerStock  super_grow;
```

```
struct CropYield
{
   int quantity;
   double size;
};
…
CropYield  apples;
```

- This is because we have to use the dot operator
- **super_grow.quantity** and **apples.quantity** are *different* variables stored in different locations in computer memory

Structures as Return Function Types

- Structures **can also** be the type of a value ***returned*** by a function

Example:

```
CDAccount shrink_wrap
            (double the_balance, double the_rate, int the_term)
{
    CDAccount temp;
    temp.balance = the_balance;
    temp.interest_rate = the_rate;
    temp.term = the_term;
    return temp;
}
```

What is this function doing?

Example: Using Function **shrink_wrap**

- **shrink_wrap** builds a complete structure value in the structure **temp**, which is returned by the function

- We can use **shrink_wrap** to give a variable of type **CDAccount** a value in this way:

```
CDAccount  new_account;
new_account = shrink_wrap(1000.00, 5.1, 11);
```

Initializes *.balance* *.interest_rate* *.term*

Assignment and Structures

- The assignment operator (**=**) can also be used to give values to structure types
- Using the **CDAccount** structure again for example:

```
CDAccount my_account, your_account;

my_account.balance = 1000.00;
my_account.interest_rate = 5.1;
my_account.term = 12;
// or alternatively, use the shrink_wrap() function defined before…

your_account = my_account;
```

- Note:
  This last line assigns *all member variables* in **your_account** the *corresponding* values in **my_account**

Initializing Structures

- A structure can also be initialized when declared

Example:

```
struct Date
{
    int month;
    int day;
    int year;
};
```

month day year

- Can be initialized in this way – watch out for the order AND the amount!:

```
Date due_date = {4, 20, 2018};
Date birthday = {12, 25, 2000};
Date birthday = {12, 25, 2020, 11}; // will cause error – WHY?
Date birthday = {0};   // will not cause error – WHY??
```

Hierarchical Structures

- Structures **can** contain member variables that are **also structures**

```
struct Date
{
    int month;
    int day;
    int year;
};
```

```
struct PersonInfo
{
    double height;
    int weight;
    Date birthday;
    string fullName;
};
```

- struct **PersonInfo** contains a **Date** structure

Using **PersonInfo**

*An example on "." **operator** use*

```
struct PersonInfo
{
    double height;
    int weight;
    Date birthday;
};
```

```
struct Date
{
    int month;
    int day;
    int year;
};
```

- A variable of type **PersonInfo** is declared:

```
PersonInfo person1;
```

- To display the birth year of **person1**,
  first access the birthday member of person1

```
cout <<  person1.birthday
```
…*(wait! not complete yet!)*

- But we want the ***year***, so we now specify the year member of the birthday member

```
cout << person1.birthday.year;
```
…(whoooaaa! Double dot operator!!)

Is this Legal?

```
struct House {
   string city;
   string streetName;
   int addressNumber;
   int zipCode;
   bool HasFireplace;
} SmithHouse = {“SB”, “Main St”, 123, 93101, true};
```

Yes

It is the same as when we do this: int number = 42;

Note that we didn’t need a semicolon after the struct def!

YOUR TO-DOs

- Finish Lab #7 and Homework #7
  - Due on Monday, Nov. 23rd
- Take advantage of office hours this week!!
- Take the Quiz on Friday!

No classes, labs, office hours, quiz next week!!!
(Thanksgiving Break)

</LECTURE>