C++ Programming Essentials

Unit 3: Basic Abstract Data Types

CHAPTER 12: VECTORS

DR. ERIC CHOU

IEEE SENIOR MEMBER

LECTURE 1

Motivation for Vectors

Topics

- Motivation for Vectors (ArrayList in Java)
- Vector class
- Member Functions for Vector Class
- Case Study: student score average report generation

Motivation for Vectors

Example Case Study – Calculate the Student Score Average

- A file contains a sequence of names and scores:
  - Ann 92
  - Bob 84
  - Chris 89
  - . . .

Note the different types of data

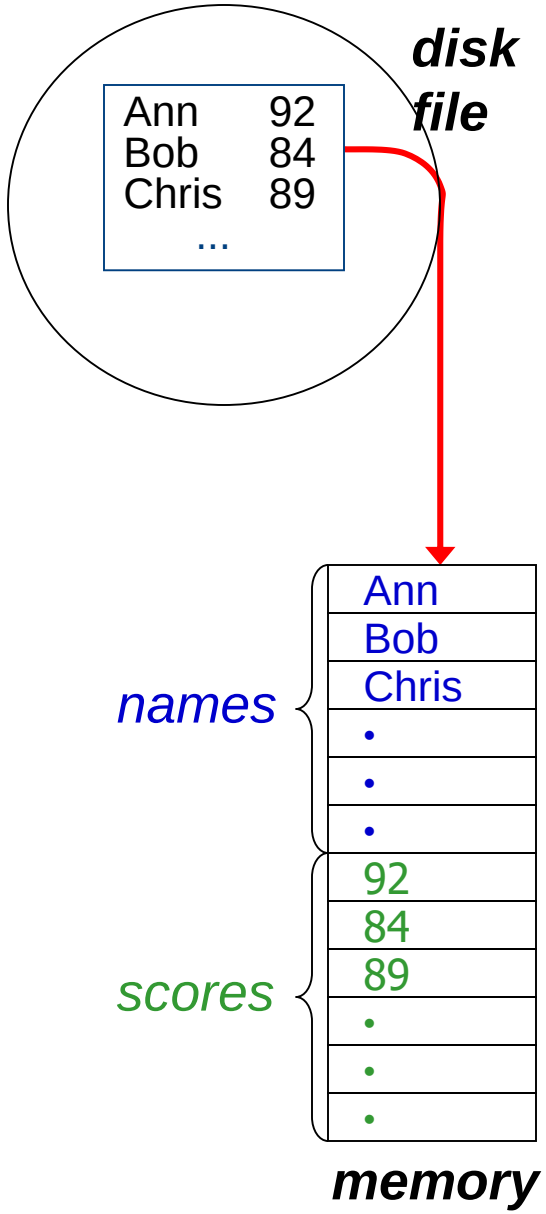
- Write a program that computes the average score, and displays each name, its score, and its difference from the average.

Preliminary Analysis

- This problem requires us to process the scores twice:
- Once to compute their average; and
- Once to compute the difference of each score from the average.
- One way is to close the file, then reopen it with **open()** [or use **seekg()** to begin reading from the beginning of the file]. But that would force us to read each name twice, which is unnecessary; it is also
- inefficient because disk I/O is considerably slower than transferring data from internal memory.

- A better way is to use
- **containers in main memory**
- and store the:
  - sequence of names in one container
  - sequence of scores in another container.

- By using two such containers, we can process the scores twice without processing the names twice and without having to read the values from disk more than once.



Sequential Containers

C++ provides several in-memory containers for storing and processing sequences of values of the same type. The candidates here are:

Arrays (from C)

Valarrays

Vectors

One of the containers in STL (Standard Template Library) www.sgi.com/tech/stl

Other containers studied in data structures courses:

- stacks
- queues
- priority queues

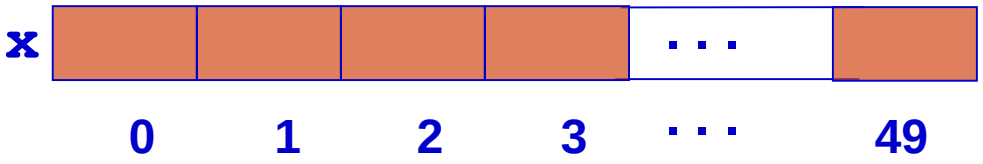
. . . and more . . .

- linked lists
- binary search trees
- directed graphs

C-Style Arrays

- C (and C++) provide two types of arrays:
  - Static: Memory allocation is specified at compile-time (This is the only kind we'll study — see §12.2)
  - Dynamic: Memory is allocated during run-time

- An array declaration (static) has the form:
  element_type name[constant_capacity];
- **Example:**
  double x[50];
- This allocates a block of 50 consecutive memory locations for storing doubles and associates the name **x** with it. They are numbered 0, 1, . . ., 49.



Two-dimensional array (matrix)?

```
double y[10][50];
```

 # rows *# columns*

- Because it may be necessary to change the capacity of the array to store more/fewer data items, it is better to not "hard-wire" the capacity into the declaration; instead, use a named constant:

```
const int CAPACITY = 50;
  . . .
double x[CAPACITY];
```

- To change the capacity, we need only change the **const** declaration.

- **Extra:** An example of how to create a dynamic array

```
double * x;
// x is a pointer to (address of) a memory
// location where a double can be stored

int cap;
cout << "Enter array's capacity: ";
cin >> cap;
x = new double[cap];  // allocate memory block for x
```

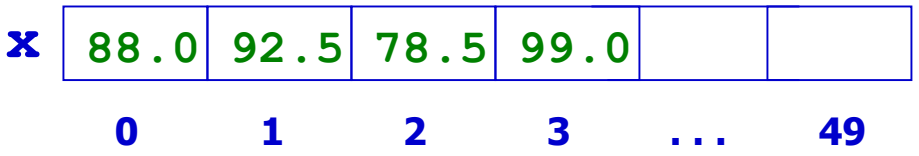
- For either kind of array, data values can now be stored in the array elements. They are accessed by using the subscript operator. The names of the individual elements are:

  x[0], x[1], x[2], . . .

- In general, **x[i]** is the **i**-th location in the array **x**;
  i is called an index or subscript.

Example: Read and store up to 50 doubles in **x**.

```
cout << "# of values (<= " << CAPACITY << ")?";
cin >> numVals;
for (int i = 0; i < numVals; i++)
   cin >> x[i];
```



C++ provides some other sequential containers that remove some of the weaknesses of C-style arrays:

Valarrays (for mathematical vectors)

- Have predefined operations, but still fixed capacity
- Only numeric type

Vectors (Similar to non-shrinking ArrayList in Java)

- Have predefined operations
- Can grow when necessary (but not shrink)
- Elements can be of any type

So, we'll use vectors . . .

LECTURE 2

C++ STL Vectors

Introduction to the STL `vector`

- A `vector` is like an array in the following ways:
  - A `vector` holds a sequence of values, or elements.
  - A `vector` stores its elements in contiguous memory locations.
  - You can use the array subscript operator `[]` to read the individual elements in the `vector`

Note: STL (Standard Templete Library). Vector is similar to ArrayList in Java

Introduction to the STL `vector`

- However, a `vector` offers several advantages over arrays. Here are just a few:
  - You do not have to declare the number of elements that the vector will have.
  - If you add a value to a vector that is already full, the vector will automatically increase its size to accommodate the new value.
  - `vector`s can report the number of elements they contain.

Declaring a `vector`

- To use vectors in your program, you must first `#include` the vector header file with the following statement:

```
#include <vector>
```

Note: There is no .h at the end of the file name.

Declaring a `vector`

The next step is to include the following statement after your `#include` statements:

```
using namespace std;
```

The STL uses *namespaces* to organize the names of its data types and algorithms.

Declaring a `vector`

Now you are ready to declare an actual `vector` object. Here is an example:

```
vector<int> numbers;
```

The statement above declares `numbers` as a `vector` of `int`s.

Declaring a `vector`

- You can declare a starting size, if you prefer. Here is an example:

```
vector<int> numbers(10);
```

The statement above declares `numbers` as a `vector` of 10 `int`s.

Other examples of `vector` Declarations

Declaration Format	Description
`vector<float> amounts;`	Declares `amounts` as an empty `vector` of `float`s.
`vector<int> scores(15);`	Declares `scores` as a `vector` of 15 `int`s.
`vector<char> letters(25, 'A');`	Declares `letters` as a `vector` of 25 characters. Each element is initialized with 'A'.
`vector<double> values2(values1);`	Declares `values2` as a `vector` of `double`s. All the elements of `values1`, which also a `vector` of `doubles`, are copied to `value2`.

Storing and Retrieving Values in a `vector`

- To store a value in an element that already exists in a `vector`, you may use the array subscript operator `[]`.
- Can have pre-defined vector size or not.
- This is easier than the ArrayList in Java.
- Performance-wise vector is not too efficient. In C++ 11, new array class has been invented to replace this data structure.

Accessing elements

- The [] operator reads and writes the elements of a vector in much the same way it accesses the characters in an pstring. The indices start at zero, so count[0] refers to the "zeroeth" element of the vector, and count[1] refers to the "oneth" element. You can use the [] operator anywhere in an expression:

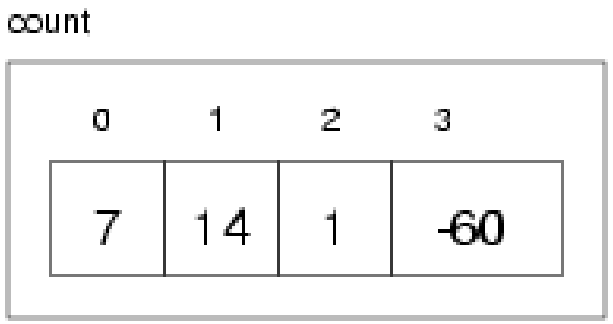
```
count[0] = 7;

count[1] = count[0] * 2;

count[2]++;

count[3] -= 60;
```

- All of these are legal assignment statements. Here is the effect of this code fragment:



Use Vector like Array

You can use any expression as an index, as long as it has type int. One of the most common ways to index a vector is with a loop variable. For example:

```
int i = 0;
while (i < 4) {
  cout << count[i] << endl;
  i++;
}
```

This while loop counts from 0 to 4; when the loop variable i is 4, the condition fails and the loop terminates. Thus, the body of the loop is only executed when i is 0, 1, 2 and 3.

Each time through the loop we use i as an index into the vector, outputting the ith element. This type of vector traversal is very common. Vectors and loops go together like fava beans and a nice Chianti.

Copying vectors

- There is one more constructor for pvectors, which is called a copy constructor because it takes one vector as an argument and creates a new vector that is the same size, with the same elements.

```
vector<int> copy (count);   // copy constructor
```

- Although this syntax is legal, it is almost never used for pvectors because there is a better alternative:

```
vector<int> copy = count;  // copy constructor
```

- The = operator works on vectors in pretty much the way you would expect.

LECTURE 3

Basic Operations for Vectors
Demo Program I

Demo Program

vector1.cpp

Go Dev C++!!!

```
// This program stores, in two vectors, the hours worked by 5
// employees, and their hourly pay rates.
#include <iostream>
#include <vector>  // Needed to declare vectors
using namespace std;

int main()
{
    vector<int> hours(5);       // Declare a vector of 5 integers
    vector<float> payRate(5);   // Declare a vector of 5 floats

    cout << "Enter the hours worked by 5 employees and their\n";
    cout << "hourly rates.\n";
    for (int index = 0; index < 5; index++)
    {
        cout << "Hours worked by employee #" << (index + 1);
        cout << ": ";
        cin >> hours[index];
        cout << "Hourly pay rate for employee #";
        cout << (index + 1) << ": ";
        cin >> payRate[index];
    }
```

```
    cout << "Here is the gross pay for each employee:\n";
    cout.precision(2);
    cout.setf(ios::fixed | ios::showpoint);
    for (index = 0; index < 5; index++)
    {
        float grossPay = hours[index] * payRate[index];
        cout << "Employee #" << (index + 1);
        cout << ": $" << grossPay << endl;
    }
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter the hours worked by 5 employees and their
hourly rates.
Hours worked by employee #1: 10 [Enter]
Hourly pay rate for employee #1: 9.75 [Enter]
Hours worked by employee #2: 15 [Enter]
Hourly pay rate for employee #2: 8.62 [Enter]
Hours worked by employee #3: 20 [Enter]
Hourly pay rate for employee #3: 10.50 [Enter]
Hours worked by employee #4: 40 [Enter]
Hourly pay rate for employee #4: 18.75 [Enter]
Hours worked by employee #5: 40 [Enter]
Hourly pay rate for employee #5: 15.65 [Enter]
Here is the gross pay for each employee:
Employee #1: $97.50
Employee #2: $129.30
Employee #3: $210.00
Employee #4: $750.00
Employee #5: $626.00
```

LECTURE 4

Member Functions for Vector Class I

vector class methods	Description
unsigned int size()	Returns the number of elements in a vector
push_back(*type* element)	Adds an element to the end of a vector
bool empty()	Returns true if the vector is empty
void clear()	Erases all elements of the vector
***type* at(int n)**	Returns the element at index n, with bounds checking

vector class overloaded operators	Description
=	Assignment replaces a vector's contents with the contents of another
==	An element by element comparison of two vectors
[ ]	Random access to an element of a vector (usage is similar to that of the operator with arrays.) Keep in mind that it does not provide bounds checking.

Using the `push_back` Member Function

- You cannot use the `[]` operator to access a `vector` element that does not exist.
- To store a value in a `vector` that does not have a starting size, or is already full, use the `push_back` member function. Here is an example:

```
numbers.push_back(25);
```

Demo Program

vector2.cpp

Go Dev C++!!!

```
// This program stores, in two vectors, the hours worked by a specified
// number of employees, and their hourly pay rates.

#include <iostream>
#include <vector>  // Needed to declare vectors
using namespace std;

int main()
{
    vector<int> hours;      // hours is an empty vector
    vector<float> payRate;  // payRate is an empty vector
    int numEmployees;       // The number of employees

    cout << "How many employees do you have? ";
    cin >> numEmployees;
    cout << "Enter the hours worked by " << numEmployees;
    cout << " employees and their hourly rates.\n";
```

```
    for (int index = 0; index < numEmployees; index++)
    {
        int tempHours;   // To hold the number of hours entered
        float tempRate; // To hold the payrate entered

        cout << "Hours worked by employee #" << (index + 1);
        cout << ": ";
        cin >> tempHours;
        hours.push_back(tempHours); // Add an element to hours
        cout << "Hourly pay rate for employee #";
        cout << (index + 1) << ": ";
        cin >> tempRate;
        payRate.push_back(tempRate); // Add an element to payRate
    }
    cout << "Here is the gross pay for each employee:\n";
    cout.precision(2);
    cout.setf(ios::fixed | ios::showpoint);
    for (index = 0; index < numEmployees; index++)
    {
            float grossPay = hours[index] * payRate[index];
        cout << "Employee #" << (index + 1);
        cout << ": $" << grossPay << endl;
    }
    return 0;

}
```

```cpp
// This program stores, in two vectors, the hours worked by a specified
// number of employees, and their hourly pay rates.
#include <iostream>
#include <vector>   // Needed to declare vectors
using namespace std;
int main(){
    vector<int> hours;      // hours is an empty vector
    vector<float> payRate;  // payRate is an empty vector
    int numEmployees;       // The number of employees

    cout << "How many employees do you have? ";
    cin >> numEmployees;
    cout << "Enter the hours worked by " << numEmployees;
    cout << " employees and their hourly rates.\n";

    for (int index = 0; index < numEmployees; index++){
        int tempHours;   // To hold the number of hours entered
        float tempRate; // To hold the payrate entered

        cout << "Hours worked by employee #" << (index + 1);
        cout << ": ";
        cin >> tempHours;
        hours.push_back(tempHours); // Add an element to hours
        cout << "Hourly pay rate for employee #";
        cout << (index + 1) << ": ";
        cin >> tempRate;
        payRate.push_back(tempRate); // Add an element to payRate
    }
    cout << "Here is the gross pay for each employee:\n";
    cout.precision(2);
    cout.setf(ios::fixed | ios::showpoint);
    for (int index = 0; index < numEmployees; index++){
            float grossPay = hours[index] * payRate[index];
        cout << "Employee #" << (index + 1);
        cout << ": $" << grossPay << endl;
    }
    return 0;
}
```

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch12\vector2\vector2.exe
How many employees do you have? 3
Enter the hours worked by 3 employees and their hourly rates.
Hours worked by employee #1: 10
Hourly pay rate for employee #1: 100
Hours worked by employee #2: 13
Hourly pay rate for employee #2: 80
Hours worked by employee #3: 12
Hourly pay rate for employee #3: 105
Here is the gross pay for each employee:
Employee #1: $1000.00
Employee #2: $1040.00
Employee #3: $1260.00

--------------------------------
Process exited after 22.56 seconds with return value 0
Press any key to continue . . .
```

Determining the Size of a `vector`

- Unlike arrays, vectors can report the number of elements they contain. This is accomplished with the `size` member function. Here is an example of a statement that uses the size member function:

```
numValues = set.size();
```

- In the statement above, assume that `numValues` is an `int`, and `set` is a `vector`. After the statement executes, `numValues` will contain the number of elements in the vector `set`.

Determining the Size of a `vector`

When Size is undefined

Example:

```
void showValues(vector<int> vect)
{
      for (int count = 0; count < vect.size(); count++)
            cout << vect[count] << endl;
}
```

Demo Program

vector3.cpp

Go Dev C++!!!

```
// This program demonstrates the vector size
// member function.
#include <iostream>

using namespace std;

#include <vector>
using namespace std;

// Function prototype
void showValues(vector<int>);

int main()
{
    vector<int> values;

    for (int count = 0; count < 7; count++)
        values.push_back(count * 2);
    showValues(values);
    return 0;

}
```

```
//*****************************************
// Definition of function showValues.     *
// This function accepts an int vector as its *
// argument. The value of each of the vector's *
// elements is displayed.                 *
//*****************************************

void showValues(vector<int> vect)
{
    for (int count = 0; count < vect.size(); count++)
       cout << vect[count] << endl;
}
```

```cpp
// This program demonstrates the vector size
// member function.
#include <iostream>
using namespace std;
#include <vector>
using namespace std;

// Function prototype
void showValues(vector<int>);

int main(){
    vector<int> values;
    for (int count = 0; count < 7; count++)
        values.push_back(count * 2);
    showValues(values);
    return 0;
}

void showValues(vector<int> vect)
{
    for (int count = 0; count < vect.size(); count++)
        cout << vect[count] << endl;
}
```

Program Output

```
0
2
4
6
8
10
12
```

LECTURE 5

Member Functions for Vector Class II

Removing Elements from a `vector`

Use the `pop_back` member function to remove the last element from a `vector`.

```
collection.pop_back();
```

The statement above removes the last element from the `collection` vector.

Demo Program

vector4.cpp

Go Dev C++!!!

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch12\vector4\vector4.exe
The size of values is 3
Popping a value from the vector...
The size of values is now 2
Popping a value from the vector...
The size of values is now 1
Popping a value from the vector...
The size of values is now 0

--------------------------------
Process exited after 0.01016 seconds with return value 0
Press any key to continue . . .
```

```cpp
// This program demosntrates the vector size member function.
#include <iostream>
#include <vector>
using namespace std;

int main(){
    vector<int> values;
    // Store values in the vector
    values.push_back(1);
    values.push_back(2);
    values.push_back(3);
    cout << "The size of values is " << values.size() << endl;
    // Remove a value from the vector
    cout << "Popping a value from the vector...\n";
    values.pop_back();
    cout << "The size of values is now " << values.size() << endl;
    // Now remove another value from the vector
    cout << "Popping a value from the vector...\n";
    values.pop_back();
    cout << "The size of values is now " << values.size() << endl;
    // Remove the last value from the vector
    cout << "Popping a value from the vector...\n";
    values.pop_back();
    cout << "The size of values is now " << values.size() << endl;
    return 0;
}
```

```
// This program demosntrates the vector size member function.

#include <iostream>

using namespace std;
#include <vector>
using namespace std;

int main()
{
    vector<int> values;

    // Store values in the vector
    values.push_back(1);
    values.push_back(2);
    values.push_back(3);
    cout << "The size of values is " << values.size() << endl;

    // Remove a value from the vector
    cout << "Popping a value from the vector...\n";
    values.pop_back();
    cout << "The size of values is now " << values.size() << endl;
```

```
    // Now remove another value from the vector
    cout << "Popping a value from the vector...\n";
    values.pop_back();
    cout << "The size of values is now " << values.size() << endl;

    // Remove the last value from the vector
    cout << "Popping a value from the vector...\n";
    values.pop_back();
    cout << "The size of values is now " << values.size() << endl;
    return 0;
}
```

Program Output

```
The size of values is 3
Popping a value from the vector...
The size of values is now 2
Popping a value from the vector...
The size of values is now 1
Popping a value from the vector...
The size of values is now 0
```

Clearing a `vector`

To completely clear the contents of a `vector`, use the `clear` member function. Here is an example:

```
numbers.clear();
```

After the statement above executes, the `numbers` vector will be cleared of all its elements.

Demo Program

vector5.cpp

Go Dev C++!!!

```
// This program demosntrates the vector size member function.
#include <iostream>using namespace std;
#include <vector>
using namespace std;

int main()
{
   vector<int> values(100);

   cout << "The values vector has “
      << values.size() << " elements.\n";
   cout << "I will call the clear member function...\n";
   values.clear();
   cout << "Now, the values vector has “
      << values.size() << " elements.\n";
   return 0;

}
```

Program Output

```
The values vector has 100 elements.
I will call the clear member function...
Now, the values vector has 0 elements.
```

LECTURE 6

Member Functions for Vector Class III

Detecting an Empty `vector`

To determine if a vector is empty, use the `empty` member function. The function returns `true` if the vector is empty, and `false` if the `vector` has elements stored in it. Here is an example of its use:

```
if (set.empty())
    cout << "No values in set.\n";
```

Demo Program

vector6.cpp

Go Dev C++!!!

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch12\vector6>vector6
How many values do you wish to average? 3
Enter a value: 23
Enter a value: 43
Enter a value: 43
Average: 36
```

```cpp
// This program demonstrates the vector's empty member function.
#include <iostream>
#include <vector>
using namespace std;

// Function prototype
float avgVector(vector<int>);

int main()
{
   vector<int> values;
   int numValues;
   float average;

   cout << "How many values do you wish to average? ";
   cin >> numValues;
   for (int count = 0; count < numValues; count++){
      int tempValue;

      cout << "Enter a value: ";
      cin >> tempValue;
      values.push_back(tempValue);
   }
   average = avgVector(values);
   cout << "Average: " << average << endl;
   return 0;
}

float avgVector(vector<int> vect){
   int total = 0;  // accumulator
   float avg;      // average

   if (vect.empty()){ // Determine if the vector is empty
      cout << "No values to average.\n";
      avg = 0.0;
   }
   else{
      for (int count = 0; count < vect.size(); count++)
         total += vect[count];
      avg = total / vect.size();
   }
   return avg;
}
```

```
// This program demonstrates the vector's empty member function.

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

// Function prototype
float avgVector(vector<int>);

int main()
{
    vector<int> values;
    int numValues;
    float average;

    cout << "How many values do you wish to average? ";
    cin >> numValues;
```

```
    for (int count = 0; count < numValues; count++)
    {
        int tempValue;

        cout << "Enter a value: ";
        cin >> tempValue;
        values.push_back(tempValue);
    }
    average = avgVector(values);
    cout << "Average: " << average << endl;

    return 0;
}

//*********************************************************
// Definition of function avgVector.                      *
// This function accepts an int vector as its argument. If *
// the vector contains values, the function returns the    *
// average of those values. Otherwise, an error message is *
// displayed and the function returns 0.0.                *
//*********************************************************
```

```
float avgVector(vector<int> vect)
{
    int total = 0; // accumulator
    float avg;     // average

    if (vect.empty())  // Determine if the vector is empty
    {
        cout << "No values to average.\n";
        avg = 0.0;
    }
    else
    {
        for (int count = 0; count < vect.size(); count++)
            total += vect[count];
        avg = total / vect.size();
    }
    return avg;
}
```

Program Output with Example Input Shown in Bold

```
How many values do you wish to average?
Enter a value: 12
Enter a value: 18
Enter a value: 3
Enter a value: 7
Enter a value: 9
Average: 9
```

Program Output with Example Input Shown in Bold

```
How many values do you wish to average? 0
No values to average.
Average: 0
```

LECTURE 7

Summary for Vector Member Functions

Summary of vector Member Functions

Member Function	Description
`at(`*`element`*`)`	Returns the value of the element located at *element* in the vector. ***Example:*** `x = vect.at(5);` The statement above assigns the value of the 5th element of `vect` to `x`.
`capacity()`	Returns the maximum number of elements that may be stored in the vector without additional memory being allocated. (This is not the same value as returned by the size member function). ***Example:*** `x = vect.capacity();` The statement above assigns the capacity of `vect` to `x`.

Summary of vector Member Functions

`clear()`	Clears a vector of all its elements. ***Example:*** `vect.clear();` The statement above removes all the elements from `vect`.

`empty()`	Returns true if the vector is empty. Otherwise, it returns false.
	Example:
	`if (vect.empty())`
	`  cout << "The vector is empty.";`
	The statement above displays the message if `vect` is empty.

`pop_back()`	Removes the last element from the vector. ***Example:*** `vect.pop_back();` The statement above removes the last element of `vect`, thus reducing its size by 1.

Summary of vector Member Functions

`push_back(`*`value`*`)`	Stores a value in the last element of the vector. If the vector is full or empty, a new element is created. ***Example:*** `vect.push_back(7);` The statement above stores 7 in the last element of `vect`.
`reverse()`	Reverses the order of the elements in the vector (the last element becomes the first element, and the first element becomes the last element.) ***Example:*** `vect.reverse();` The statement above reverses the order of the element in `vect`.
`resize(`*`elements`*`, `*`value`*`)`	Resizes a vector by *elements* elements. Each of the new elements is initialized with the value in *value*. ***Example:*** `vect.resize(5, 1);` The statement above increases the size of `vect` by 5 elements. The 5 new elements are initialized to the value 1.

Summary of vector Member Functions

`swap(`*`vector2`*`)`	Swaps the contents of the vector with the contents of *vector2*. ***Example:*** `vect1.swap(vect2);` The statement above swaps the contents of `vect1` and `vect2`.

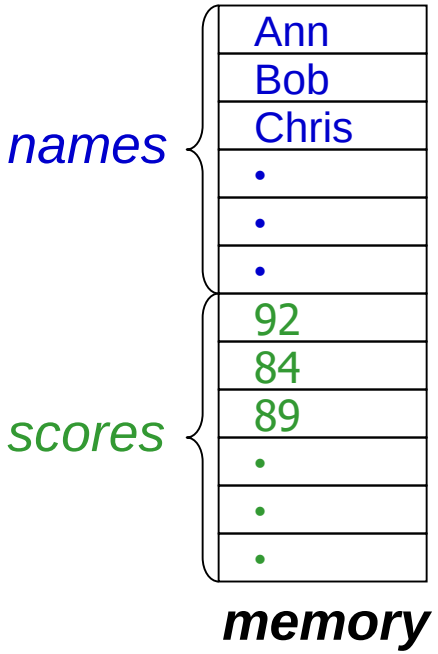
LECTURE 8

Case Study
Function I: fillVectors()

Back to our Program:

Average Score of Students

1. Get the name of the input file from the user.
2. From the input file, read names into one vector and scores into another vector.
3. Compute the average of the values in the scores vector.
4. Finally, output each name, its corresponding score, and the difference of that score and the average.



Implementation in C++

1. We know how to use a **`string`** variable to store the name of a file and use **`.data()`** to extract the name from it.

   Or we could use:
   ```
   char inFileName[30];
   cin >> inFileName;
   ifstream fin(inFileName);
   ```

2. A declaration of a **`vector`** in C++ has the form:

```
vector<T> vec;
```

   where ***T*** is the type of the elements of the vector **`vec`**. **`vector`** is really a class template (like **`complex<T>`**). It becomes a class when we specify what type ***T*** is.

So we need two vectors:

•vector of strings to store the names:

```
vector<string> nameVec;
```

•vector of doubles to store the scores:

```
vector<double> scoreVec;
```

Now we need to:

- fill these vectors from the file of names and scores
- compute the average of the scores in **`scoreVec`**
- output the names in **`nameVec`**, the scores in **`scoreVec`**, and the differences between the scores and the average.

Since none of these operations are built into C++, we will write functions to perform them.

Function #1: `fillVectors()`

- This function should receive from its caller the name of the input file, an (empty) vector of strings, and an (empty) vector of doubles. It should open an ifstream to the file.
- Then, using an input loop, the function should read a name and score, append the name to the vector of strings, and append the score to the vector of doubles.
- On exiting the loop, the function should pass the name-vector and score-vector back to the caller.

Observations about fillVectors()

- It must pass back two vectors, so these must be *reference parameters*.
- It must repeatedly read a name and a score and then append the name to the vector of names, and the score to the vector of scores. The vector class template provides a built-in operation for this:

```
vecName.push_back(value);
```

Algorithm for **`fillVectors()`***:*

1.Receive inFileName, *nameVec* and *scoreVec*.

2.Open *inStream*, a stream to *inFileName*, and check that it opened successfully.

3.Loop

    1.Read *name* and *score* from *inStream*.

    2.If end-of-file was reached, terminate repetition.

    3.Append *name* to *nameVec*.

    4.Append *score* to *scoreVec*.

4.End loop.

5.Close *inStream*.

```
//-- Needs <ifstream>, <cassert>, and <vector> libraries

void fillVectors(string inFileName,
                 vector<string> & nameVec,
                 vector<double> & scoreVec)
{
  // open input stream to inFileName
  ifstream fin(inFileName.data());
  assert(fin.is_open());

  string name;                    // input variables
  double score;

  for (;;)                        //Loop:
  {
    fin >> name >> score;         // read name, score
    if (fin.eof()) break;         // if none left, quit
    nameVec.push_back(name);      // append name to nameVec
    scoreVec.push_back(score); // append score to scoreVec
  }
  fin.close();
}
```

LECTURE 9

Case Study
Function II: average()

Function #2: **average()**

- This function should receive from its caller a vector of doubles. It must then compute the average of these doubles, and return this average.

Observations about average()

- To avoid having to copy this (potentially large) vector, we make it a *constant reference parameter* instead of a value parameter.
- To compute the average score, we need to know how many scores there are; **vector** has a built-in function member for this:

```
vecName.size();
```

- We need the sum of the scores in **scoreVec**. Like arrays, vectors have a subscript operator **[ ]**, so we could just use a for-loop for this:

```
double sum = 0;
for (int i = 0; i < scoreVec.size(); i++)
  sum += scoreVec[i];
```

- However, C++ has a function that does this.

Algorithm for `average()`:

1. *Receive numVec*.
2. Compute *sum*, the sum of the values in *numVec*.
3. Compute *numValues*, the number of values in *numVec*.
4. If *numValues* > 0:
    Return *sum* / *numValues*.
    Otherwise
        1. Display an error message via `cerr`.
        2. Return 0.0 as a default value.
5. End if.

accumulate() function

- The **`accumulate()`** function can be used to sum the values in a C++ container — **`vector`**, in particular. It is found in the **`<numeric>`** library (possibly in **`<algorithm>`** in older versions of C++).

Pattern:

```
sum = accumulate(vec.begin(),   // start
                 vec.end(),     // stop
                 0.0);          // initial
                                //  value
```

See "C++ algorithm Library Quick Reference"

Initial value of sum; it also determines the type of the sum (`int` or `double`)

iterators:
`begin()` points to first element,
`end()` points just past last element,

```
// Needs <vector> and <numeric> libraries

double average(const vector<double> & numberVec)
{
  int numValues = numberVec.size();
  if (numValues > 0)
  {
    double sum = accumulate(numberVec.begin(),
           numberVec.end(), 0.0);
    return sum / numValues;
  }
  else
  {
    cerr << "\n*** Average: vector is empty!\n" << endl;
    return 0.0;
  }
}
```

LECTURE 10

Case Study Function III: displayResults()

Function #3: **displayResults()**

- This function should receive from its caller a vector of names, a vector of scores, and an average.
- It should then use a loop to output each name, its corresponding score, and the difference of the score from the average.

Observations about displayResults()

- Because it returns no value, like Application #1, this will be a void function that receives the two vectors — the names and the scores — and the average score.
- Again, to avoid having to copy potentially large vectors, we make them constant reference parameters instead of value parameters.
- We can use a for-loop to run through the two vectors and output each name, the corresponding score, and the difference between that score and the average, using the subscript operator [ ] to access the individual names and scores.

```
// Needs <vector> library

void displayResults(const vector<string> & names,
                    const vector<double> & scores,
                    double theAverage)
{
   for (int i = 0; i < names.size(); i++)
      cout << names[i] << '\t'
           << scores[i] << '\t'
           << scores[i] - theAverage << endl;
}
```

LECTURE 11

Case Study: Main Program

Main Function Coding

```
/* processScores.cpp
     ...
 ----------------------*/

#include <iostream>                  // cin, cout, ...
#include <fstream>                   // ifstream, ...
#include <string>                    // string
#include <cassert>                   // assert()
#include <vector>                    // vector
#include <numeric>                   // accumulate()
using namespace std;

                                     // local prototypes
void fillVectors(string inFileName,
                 vector<string> & nameVec,
                   vector<double> & scoreVec);

double average(const vector<double> & numberVec);

void displayResults(const vector<string> & names,
                    const vector<double> & scores,
                    double theAverage);
```

```
int main()
{
  cout << "\nTo process the names and scores in an input file,"
       << "\n enter the name of the file: ";
  string inFileName;
  getline(cin, inFileName);

  vector<string> nameVec;
  vector<double> scoreVec;
  fillVectors(inFileName, nameVec, scoreVec);       // input

  double theAverage = average(scoreVec);            // process

  displayResults(nameVec, scoreVec, theAverage );  // output
}


// Definitions of fillVectors(), average(), displayResults() go here
```

Program Testing

To test our program, we use a text editor and create easy-to-check input files:

```
Ann 90
Bob 70
Chris 80
```

If we name this particular file *test1.txt*, then for it, our program should display:

```
Ann      90       10
Bob      70       -10
Chris    80       0
```

Some Other Useful & Powerful Functions

- You should explore the standard C++ libraries **<algorithm>** and **<numeric>** to see some of the functions you can "plug into" your programs and save yourself hours of coding and testing if you had to write them yourself.
- Also, these powerful functions use the most efficient methods known for carrying out operations.
- And they have been thoroughly tested so they can be used with confidence.

Some Other Useful & Powerful Functions

- One that you will learn about in the assignment is the **`sort()`** function in the **`<algorithm>`** library. It uses one of the fastest, if not the fastest, sorting methods.

- And like the other library functions, it is very easy to use because they use iterators in much the same way as the **`accumulate()`** function. For example, to sort the scores in our example, we need only write:

```
sort(scoreVec.begin() , scoreVec.end() );
```

Demo Program

Student Score Project (processScores.cpp)

Go Dev C++!!!

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch12\processScores>build

C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch12\processScores>g++ -std=c++11 processScores.cpp -o processScores

C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch12\processScores>processScores

To process the names and scores in an input file,
 enter the name of the file: test1.txt
Ann     90      10
Bob     70      -10
Chris   80      0
```