Reference: Big C++.

Exercises 6:

Exercise R6.1.

* std::vector<int> numbers(10);

for (int i = 0; i < 10; i++)

numbers[i] = i + 1;

* std::vector<int> numbers(11);

for (int i = 0, j = 0; i <= 10; i++, j++)

{

numbers[i] = j \* 2;

std::cout << numbers[i] << " ";

}

std::cout << std::endl;

* std::vector<int> numbers(10);

for (int i = 0, j = 1; i < 10; i++, j++)

{

numbers[i] = j \* j;

std::cout << numbers[i] << " ";

}

std::cout << std::endl;

* std::vector<int> numbers(10);

for (int i = 0; i < 10; i++)

{

numbers[i] = 0;

std::cout << numbers[i] << " ";

}

std::cout << std::endl;

* std::vector<int> numbers(9);

for (int i = 0, j = 1; i < 4; i++, j++)

{

numbers[i] = j \* j;

}

numbers[4] = 9;

numbers[5] = 7;

numbers[6] = 4;

numbers[7] = 9;

numbers[8] = 11;

Exercise R6.2.

* srand(time(0));

int random = 1 + rand() % (100 - 1 + 1);

std::vector<int> numbers(10);

for (int i = 0; i < 10; i++)

{

numbers[i] = random;

std::cout << numbers[i] << " ";

}

std::cout << "\n";

* for different numbers

srand(time(0));

std::vector<int> numbers(10);

for (int i = 0; i < 10; i++)

{

int random = 1 + rand() % (100 - 1 + 1);

numbers[i] = random;

std::cout << numbers[i] << " ";

}

std::cout << "\n";

Exercise R6.3.

std::vector<int> numbers(4);

for (int i = 0; i < numbers.size(); i++)

{

std::cout << "Enter a number: ";

std::cin >> numbers[i];

}

int maximum = numbers[0];

int minimum = numbers[0];

for (int i = 1; i < numbers.size(); i++)

{

if (numbers[i] > maximum)

maximum = numbers[i];

else if (numbers[i] < minimum)

minimum = numbers[i];

}

std::cout << "Maximum: " << maximum <<

"Minimum: " << minimum << "\n";

Exercise R6.4.

First, this loop does not initialize the first element v[0], the second wrong is that: this program tries to access an address that it doesn’t allowed to it (i.e. it tries to access v[10] whereas the last element of the vector is v[9].

The remedy is simple, just start the loop with index equals zero and end with index less than (but not equal) ten.

The second solution is to start the loop with index equals zero and end with index less than or equal to nine.

Exercise R6.5.

* The index of a vector or an array is an indicator to the order of the elements in that vector or array.
* The legal index values goes from zero to one less than the size of the vector.
* Bounds errors appears when there is an access of an illegal (or invalid) position of a vector.

Exercise R6.6.

#include <iostream>

#include <vector>

int main()

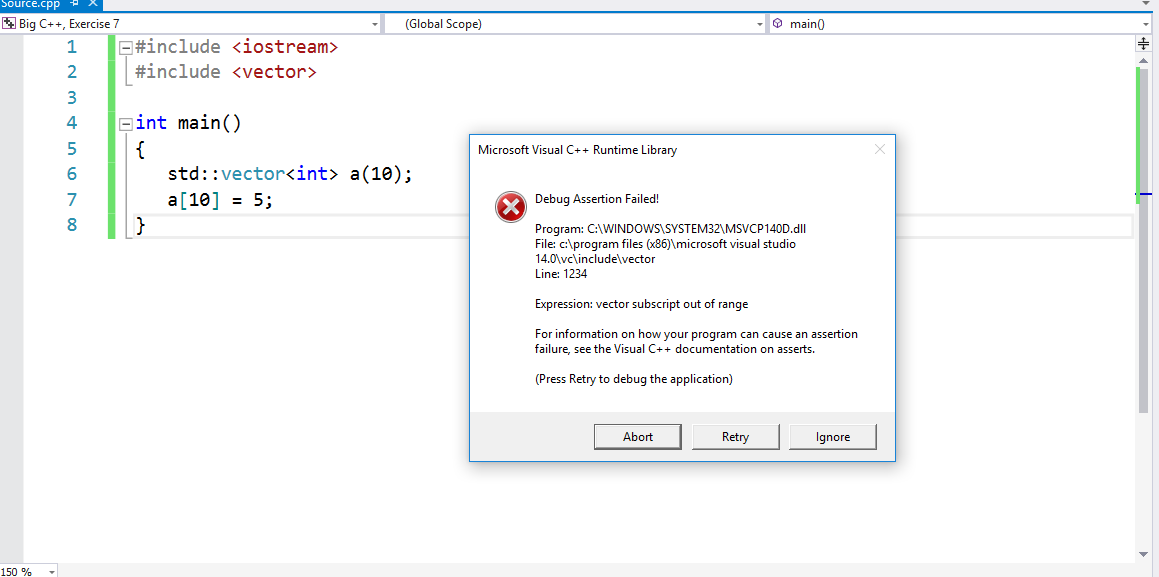
{

std::vector<int> a(10);

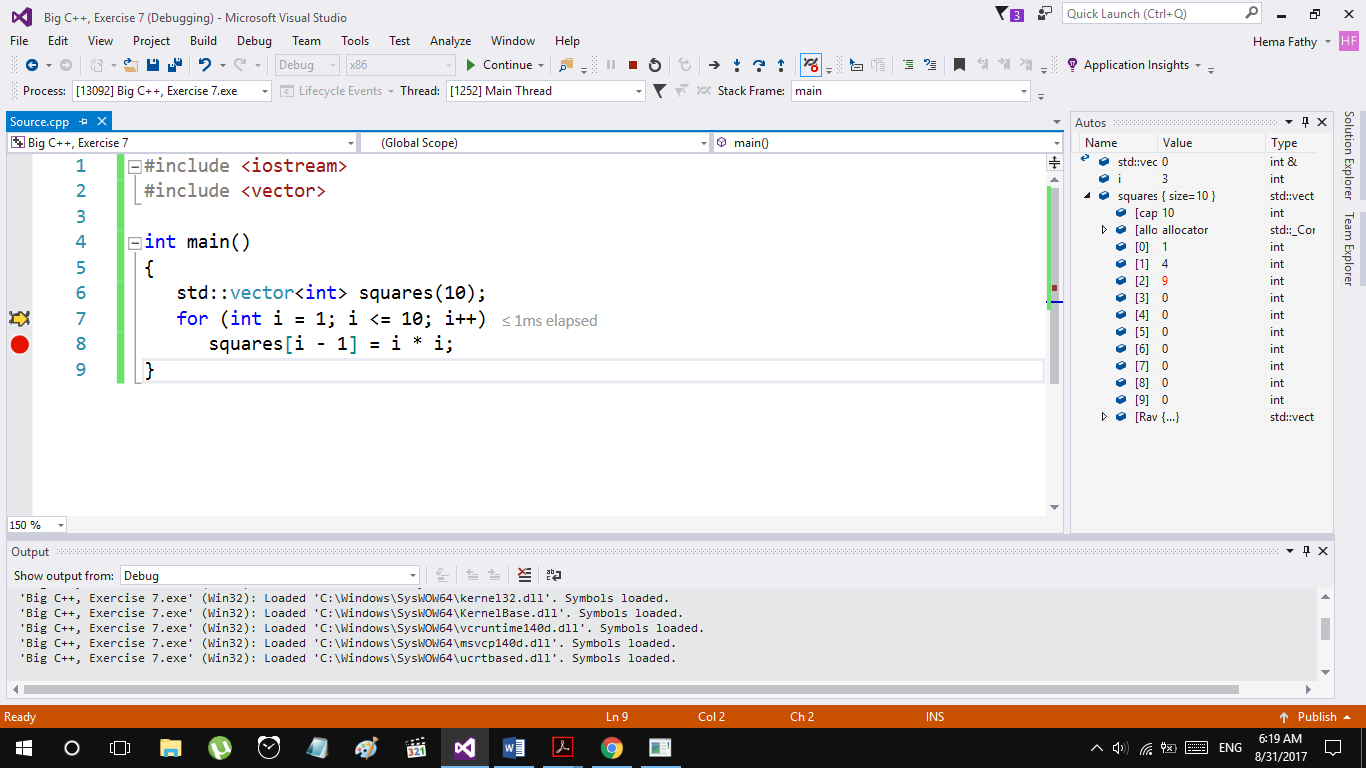
a[10] = 5;

}

A Runtime error occurs.



Exercise R6.7.



Exercise R6.8.

#include <iostream>

#include <vector>

int main()

{

std::vector<int> squares(10);

int i, n = squares.size();

std::cout << "Enter a set of numbers: \n";

for (i = 0; i < n; i++)

std::cin >> squares[i];

for (i = n - 1; i >=0; i--)

std::cout << squares[i] << " ";

std::cout << std::endl;

}

Exercise R6.11.

* Parallel vectors are vectors of some data that describe one thing, in other words, vectors that are related to each other.
* They are an indication of poor programming because the programmer must ensure that the vectors always have the same length and that each slice is filled with values that actually belong together. Moreover, any function that operates on a slice must get all vectors as parameters, which is tedious to program.
* They can be avoided by making vectors of objects.

Exercise R6.12.

***Header File:***

#pragma once

#ifndef STAFF\_H

#define STAFF\_H

#include <vector>

#include "ccc\_empl.h"

class Staff

{

public:

/\*\*

Default constructor of the Staff Class

\*/

Staff();

/\*\*

Constructs the vector staff\_members with n elemnts

@param n the number of members in the staff (size of the vector)

\*/

Staff(int n);

/\*\*

Prints the Employees' info (name and salary) on the screen

\*/

void print\_staff();

/\*\*

Appends additional Employee to the staff at the end of the vector staff\_members

@param empl the employee which will be appended

\*/

void append(const Employee& empl);

/\*\*

Insert an additonal Employee to the staff in the kth element

@param empl the employee which will be appended

@param k the index in the vector staff\_members in which the Employee empl will

be added in

\*/

void insert(const Employee& empl, int k);

/\*\*

Get the Employee with index k from the vector staff\_members

@param k the index in the vector staff\_members in which the Employee will

be returned from

@return the Employee with the index k

\*/

Employee get\_employee(int k);

/\*\*

Get the number of members in the staff

@return the size of the vector staff\_members

\*/

int get\_staff\_number();

private:

std::vector<Employee> staff\_members;

};

#endif

***CPP file:***

#include "Staff.h"

#include <iostream>

Staff::Staff()

{

}

Staff::Staff(int n)

:staff\_members(n)

{

}

void Staff::print\_staff()

{

for (int i = 0, n = staff\_members.size(); i < n; i++)

{

std::cout << staff\_members[i].get\_name() << " ... " <<

staff\_members[i].get\_salary() << "\n";

}

std::cout << "\n";

}

void Staff::append(const Employee& empl)

{

staff\_members.push\_back(empl);

}

void Staff::insert(const Employee & empl, int k)

{

int size = staff\_members.size();

if (k > size)

return;

else

{

int last = size - 1;

staff\_members.push\_back(staff\_members[last]);

for (int i = last; i > k; i--)

staff\_members[i] = staff\_members[i - 1];

staff\_members[k] = empl;

}

}

Employee Staff::get\_employee(int k)

{

return staff\_members[k];

}

int Staff::get\_staff\_number()

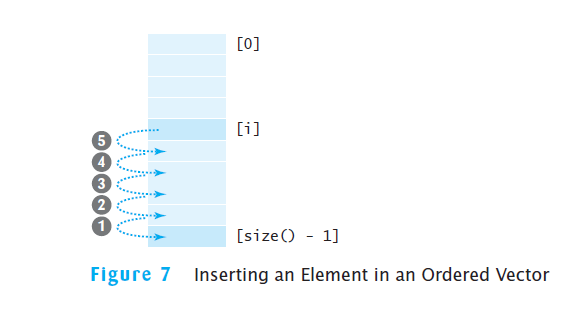
{

return staff\_members.size();

}

Exercise R6.13.

* Increase the vector size by one and copy the last element in the original vector to it.
* Now we have two elements equal to each other (the last and the one before the last).
* Suppose the new employee will be added in the ith slot.



* Then we start transferring data:
* Finally, we add the desired element in its position i.

Exercise R6.14.

#include <iostream>

#include <vector>

int main()

{

std::vector<int> problem14(4);

int length = 0, n = problem14.size();

for (int i = 0; i < n; i++)

{

std::cin >> problem14[i];

if (std::cin)

length++;

else

{

std::cout << "Error, Input failed\n";

std::cin.clear();

std::cin.ignore();

}

}

std::vector<int> problem14\_doubled;

if (length == n)

{

problem14\_doubled = std::vector<int> (n \* 2);

for (int i = 0; i < n; i++)

problem14\_doubled[i] = problem14[i];

std::cout << "Doubled!!\n";

}

return 0;

}

Exercise R6.15.

1. Test that two vectors contain the same elements in the same order.

#include <iostream>

#include <vector>

void read\_vector\_int(std::vector<int>& vector);

bool test\_vector(const std::vector<int>& vector1, const std::vector<int>& vector2);

int main()

{

std::vector<int> vector1(4), vector2(5);

int n1 = vector1.size(), n2 = vector2.size();

read\_vector\_int(vector1);

read\_vector\_int(vector2);

if (test\_vector(vector1, vector2))

std::cout << "They are the same\n";

else

std::cout << "They are not the same\n";

return 0;

}

void read\_vector\_int(std::vector<int>& vector)

{

for (int i = 0, n = vector.size(); i < n; i++)

{

std::cout << "Enter the content of slot " << i + 1 << ": ";

std::cin >> vector[i];

}

}

bool test\_vector(const std::vector<int>& vector1, const std::vector<int>& vector2)

{

int n1 = vector1.size(), n2 = vector2.size();

if (n1 != n2)

return false;

for (int i = 0; i < n1; i++)

if (vector1[i] != vector2[i])

return false;

return true;

}

1. Copy one vector to another. (Hint: You may copy more than one element at a time.)

std::vector<int> vector1(4), vector2(5);

vector2.swap(vector1);

1. Fill a vector with zeroes, overwriting all elements in it.

std::vector<int> vector1(4);

int n1 = vector1.size();

for (int i = 0; i < n1; i++)

{

vector1[i] = 0;

}

1. Remove all elements from a vector. (Hint: You need not remove them one by one.)

std::vector<int> vector1(4);

vector1.clear(); // This removes all the elements and make vector1 with size 0

Exercise R6.16.

1. All elements of a vector are of the same type. **True**
2. Vector subscripts must be integers. **True**
3. Vectors cannot contain strings as elements. **False**
4. Vectors cannot use strings as subscripts. **True**
5. Parallel vectors must have equal length. **True**
6. Two-dimensional arrays always have the same numbers of rows and columns. **False**
7. Two parallel arrays can be replaced by a two-dimensional array. **False (true if the two arrays are of the same line).**
8. Elements of different columns in a two-dimensional array can have different types. **False**

Exercise R6.17.

1. All vector parameters are reference parameters. **False, Arrays are.**
2. A function cannot return a two-dimensional array. **True**
3. A function cannot change the dimensions of a two-dimensional array that is passed by value. **True**
4. A function cannot change the length of a vector that is passed by reference. **False, you can using push\_back function.**
5. A function can only reorder the elements of a vector, not change the elements. **False**