**CSCI 1107 Lab Experience Five**

**Lab Exercises**

Given the following statements:

struct NameType {

string first;

string last;

};

struct DateType{

int month;

int day;

int year;

};

struct PersonalInfoType{

NameType name;

int identification;

DateType dob;

};

struct EmployeeType{

NameType name;

int rating;

int identification;

string dept;

double salary;

};

PersonalInfoType person;

PersonalInfoType classList[100];

NameType student;

EmployeeType employees[100];

EmployeeType newEmployee;

Answer the following questions based upon the definitions above.

1. Specify if the statement is valid or invalid. If the statement is invalid, why.

1. person.name.first = “Barney”; **Valid**
2. cout << person.name << endl; **Invalid, cannot directly cout a struct.**
3. classList[1] = person; **Valid**
4. classList[20].identification = 000011100; **Valid**
5. person = classList[20];  **Valid, as C was valid**
6. student = person.name; **Valid**
7. classList.dob.day = 1; **Invalid, using dot operator on an array (const \*).**

2. Given the above declarations write C++ statements to do the following:

1. Store the following information in newEmployee

name: Mickey Doe

identification: 1111111111

rating: 2

dept: Accounting

salary: 34567.78

**newEmployee = {{“Mickey”, “Doe”}, 2,** 1111111111, “Accounting”, 34567.78};

1. Initialize rating to zero in the employees array.

**// Could also use employees = {… , 0, …}; , but we don’t really know what to put for the rest here.**

**for(int i = 0; i < 100; i++)**

**employees[i].rating = 0;**

1. Copy the information of the 20th employee in the array employees into newEmployee.

**newEmployee = employees[19];**

3. On page 138 in your textbook do problem 2.

/\* Lab5-Prob3.cpp

\*

\* Johnathan Lee CSCI 1107

\* Lab 5 #3, problem text from P. 138 # 2.

\* Due 02/13/18

\*/

/\*

\* From book for reference: Peter the postman became bored one night and, to

\* break the monotony of the night shift, he carried out the following

\* experiment with a row of mailboxes in the post office. These mailboxes were

\* numbered 1-150, and beginning with mailbox 2, he opened the doors of all the

\* even numbered mailboxes, leaving the others closed. Next, beginning with

\* mailbox 3, he went to every 3rd mailbox, opening its door if it was closed,

\* and closing it if it was open. Then he repeated this procedure with every

\* 4th, every 5th, etc. When finished, he was surprised at the distribution of

\* closed mailboxes. Write a program to determine which mailboxes these were.

\*/

#include <iostream>

using namespace std;

void **flipBoxes**(bool\* boxAr, int numToFlip, int everyNth, bool startWithFirst);

void **displayBoxes**(bool boxes[], int numBoxes);

const int NUM\_BOXES = 150;

int **main**() {

// These mailboxes were numbered 1-150

bool boxes[NUM\_BOXES] = {true}; // Using true = closed.

// and beginning with mailbox 2, he opened the doors of all the even numbered

// mailboxes, leaving the others closed

//

// Since they started closed, flipping is the same as opening.

// Start with 0 instead so that even numbering works correctly.

// cout << "Flipping even!" << endl; // For debugging

flipBoxes(&boxes[0], NUM\_BOXES, 2, false);

// beginning with mailbox 3, he went to every 3rd mailbox, opening its door

// if it was closed, and closing it if it was open (flipping). Then he

// repeated this procedure with every 4th, every 5th, etc.

//

// Assuming continued all the way up to every NUM\_BOXES-th,

// and using i = which nth.

for (int i = 3; i < NUM\_BOXES; i++) {

// cout << "\n\nFlipping every n=" << i << endl; // For debugging

flipBoxes(&boxes[2], NUM\_BOXES - 2, i, true);

}

// Show distribution

displayBoxes(boxes, NUM\_BOXES);

return 0;

}

/// \brief Flips values in a boolean array

///

/// \pre boxAr MUST be a pointer to a bool and there MUST be at least numToFlip

/// values directly after it.

///

/// \param boxAr A pointer to the first value to flip.

/// \note Doesn't have the be the beginning of the array, just the first to flip

///

/// \param numToFlip How many values we should iterate over

///

/// \param everyNth How many values do we skip every time? (Flip every 2nd, 3rd,

/// etc)

///

/// \param startWithFirst Should we flip the first box, or just start counting

/// from there?

///

/// \post Values in boxAr have been toggled according to the rules in param.

void **flipBoxes**(bool\* boxAr, int numToFlip, int everyNth, bool startWithFirst) {

int i;

if (startWithFirst)

i = 0;

else

i = 1;

for (; i < numToFlip; i++)

if ((i + 1) % everyNth == 0) {

boxAr[i] = !boxAr[i];

// For debugging

// cout << "Flipped at i = " << i << "(Box num " << i + 1 << ")\n";

}

}

/// \brief Displays boxes in a 25x(numBoxes / 25) grid.

///

/// \param boxes The array of boxes to output.

/// \param numBoxes How many boxes in the array?

///

/// \post boxes has been printed to cout, using a '|' for closed and '-' for

/// open.

void **displayBoxes**(bool boxes[], int numBoxes) {

cout << "\nMailboxes: (| for closed and - for open): \n";

// Using | for closed, and - for open.

// Think of a normal mailbox, with the door up when closed and down when open

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

cout << (i % 25 == 0 ? "\n" : " ") << (boxes[i] ? "|" : "-");

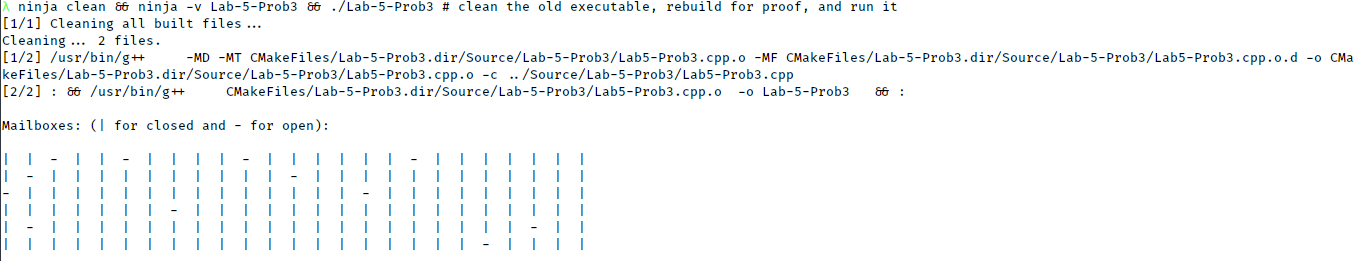
cout << "\n"; // Extra newline since my console messes up without a trailing

// newline.

}

Capture the console window for each test run and paste it in your word document. Hand in a print-out of your program.

**Only one possible test run?**



4. Do Lab 7.1 Vectors --- This is a self-guided lab for you to learn how to use the Vector datatype.

/\*--- vectorlab.cpp -------------------------------------------------------

A study of STL's vector container

Written by: Larry R. Nyhoff

Written for: Lab Manual for ADTs, Data Structures, and Problem

Solving with C++, 2E

Lab #7.1

Johnathan Lee CSCI 1107

Lab 5 Due 02/13/18

--------------------------------------------------------------------------\*/

#include <iostream>

#include <vector>

using namespace std;

template <typename T>

ostream& *operator*<<(ostream& out, const vector<T>& v)

/\*-------------------------------------------------------------------------

Overloaded output operator for vector<T>s.

Precondition: ostream out is open.

Postcondition: Elements of v have been output to out and out is returned.

--------------------------------------------------------------------------\*/

{

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

out << v.operator[](i) << " ";

return out;

}

int **main**() {

// Declare 6 vectors v1, v2, v3, v4, v5, v6 to illustrate the

// various types of declarations (and constructors)

vector<int> v1;

vector<int> v2(2);

int numInts;

cout << "Enter capacity of v3: ";

cin >> numInts;

vector<int> v3(numInts);

vector<int> v4(3, 99);

// The preceding declaration should work, but it may not in some

// versions of some compilers. The following is a work-around:

// vector<int> v4(3);

// for (int i = 0; i < 3; i++) v4[i] = 99;

//--- End of work-around

int a[] = {1, 4, 9, 16, 25};

vector<int> v5(a, a + 5);

vector<int> v6;

//--- 1 --- Add:

// Statements to display the capacity and size of each vector<int>

// and whether it is empty

cout << "(Size/cap/empty)\n";

cout << "V1: " << v1.size() << "/" << v1.capacity() << "/" << v1.empty()

<< "\n";

cout << "V2: " << v2.size() << "/" << v2.capacity() << "/" << v2.empty()

<< "\n";

cout << "V3: " << v3.size() << "/" << v3.capacity() << "/" << v3.empty()

<< "\n";

cout << "V4: " << v4.size() << "/" << v4.capacity() << "/" << v4.empty()

<< "\n";

cout << "V5: " << v5.size() << "/" << v5.capacity() << "/" << v5.empty()

<< "\n";

//--- 2 --- Add:

// Statements to display the maximum capacity of a vector<int>

cout << "Max cap of vector<int>: " << v6.max\_size() << endl;

//--- 3 --- Add:

// Statements to see the effect of the reserve() member function

v4.reserve(7);

cout << "New V4: " << v4.size() << "/" << v4.capacity() << "\n";

//--- 4 --- Add:

// Output statements of the form cout << vector-variable << endl;

// to display the contents of each vector

// NOTE: Ask about using a range based loop.

cout << "V1: " << v1 << endl;

cout << "V2: " << v2 << endl;

cout << "V3: " << v3 << endl;

cout << "V4: " << v4 << endl;

cout << "V5: " << v5 << endl;

//--- 5 --- Add:

// Statements to append 11 to v2 and then output v2's size and contents

// append 22 to v2 and then output v2's size and contents

// append 33 to v2 and then output v2's size and contents

// remove the last element of v2 and then output v2's size

// and contents

for (int i = 11; i < 33; i += 11) {

v2.push\_back(i);

cout << "New V2: " << v2 << " (" << v2.size() << " size)\n";

}

v2.pop\_back();

cout << "New V2: " << v2 << " (" << v2.size() << " size)\n";

//--- 6 --- Statements to investigate how capacities grow

// Add statements to append 111 to v1 and then output v1's capacity, size,

// and contents

v1.push\_back(111);

cout << "V1: " << v1 << " (With " << v1.size() << "/" << v1.capacity() << "/"

<< v1.empty() << ")\n";

//--- 7 --- Statements to investigate how capacities grow

// Add statements to append 222, 333, 444, and 555 to v1 and output

// v1's capacity, size, and contents after each value is appended

for (int i = 222; i <= 555; i += 111) {

v1.push\_back(i);

cout << "V1: " << v1 << " (With " << v1.size() << "/" << v1.capacity()

<< "/" << v1.empty() << ")\n";

}

//--- 8 --- Statements to investigate how capacities grow

// Remove the comment delimiters from the following:

int oldCapacity = v1.capacity();

for (int i = v1.size() + 1; i <= 2500; i++) {

v1.push\_back(999);

if (v1.capacity() == v1.size())

cout << "\n\*\*\* v1 is full with " << v1.size() << " elements\n";

if (v1.capacity() > oldCapacity) {

cout << "Adding an element increases capacity from " << oldCapacity

<< " to " << v1.capacity() << endl;

oldCapacity = v1.capacity();

}

}

//--- 9 --- Statements to see if element type affects how capacities grow

// Add:

// A declaration of an empty vector<double> v0;

// A loop like the preceding but with v1 replaced by v0

//

// Then change double to char and run it again.

{

vector<char> v0;

int oldCap = v0.capacity();

for (int i = 1; i <= 2500; i++) {

v0.push\_back(i);

if (v0.capacity() == v0.size())

cout << "\n REACHED v0 CAP WITH " << v0.size() << endl;

if (v0.capacity() > oldCap) {

cout << "Increasing capacity from " << oldCap << "->"

<< v0.capacity() << endl;

oldCap = v0.capacity();

}

}

}

//--- 10 --- Statements to see how initial capacity affects

// how capacities grow

// Uncomment the following line:

cout << "Initial capacity of v4 is " << v4.capacity() << endl;

// Add a loop like that in 9 but output changes in v4's capacity

int oldCap4 = v4.capacity();

for (int i = 1; i <= 2500; i++) {

v4.push\_back(i);

if (v4.capacity() == v4.size())

cout << "\n REACHED v0 CAP WITH " << v4.size() << endl;

if (v4.capacity() > oldCap4) {

cout << "Increasing capacity from " << oldCap4 << "->" << v4.capacity()

<< endl;

oldCap4 = v4.capacity();

}

}

//--- 11 --- Statements to access the ends of a vector

// Uncomment the following line:

cout << "Original contents of v5: " << v5 << endl;

// Add statements to:

// Output the first and last elements of v5

// Change the first element to 77 and the last element to 88

// Output the contents of v5

cout << "First: " << v5.front() << " Last: " << v5.back() << endl;

v5.front() = 77;

v5.back() = 88;

cout << "New contents of v5: " << v5 << endl;

//--- 12 --- Statements to demonstrate correct and incorrect

// use of the subscript operator

// Add statements that try using the subscript operator to:

// change the value in location 1 of v2 to 2222

// append the value 3333 to v2

// append a value to empty vector v6

v2[1] = 2222;

cout << v2;

v2[v2.size()] = 3333;

cout << "\n\n" << v2 << endl;

for (int i = 0; i <= v2.size(); i++)

cout << v2[i] << " ";

cout << endl;

cout << "Size: " << v2.size() << " Capacity: " << v2.capacity() << endl;

// v6[0] = 1107;

//--- 13 --- Add statements that:

// assign v5 to v3

// check if they are equal

// check if v5 is less than v2

// swap contents of v5 and v2

// check if v5 is less than v2

v3 = v5;

cout << (v5 == v3 ? "true" : "false") << endl;

cout << (v5 > v2 ? "true" : "false") << endl;

v5.swap(*v2*);

cout << (v5 > v2 ? "true" : "false") << endl;

}

**What to turn in:**

1. Printouts of your word document and your source code.
2. A hard copy of the questions and answers from your lab manual.
3. Compress the word document and the source code from all of your programs and submit them to the D2L assignment folder called Lab Experience Five.