CS 201 Midterm Exam

Solomon Himelbloom

October 18, 2020

1 Answers to Questions

1.1 Convert the following for loop to a while loop.

- See the below example for code that transitions from a for loop to a while loop.

```cpp
// q01.cpp
// Solomon Himelbloom
// 18 October 2020
// CS 201 Midterm Exam

#include <iostream>
using std::cin;
using std::cout;
using std::endl;

// Question #1: Convert the following for loop to a while loop.
// for ( int i=-20 ; i<1999 ; i+=3 ) cout << i;

int main() {
    int i = -20;

    while ( i < 1999) {
        cout << i;
        i += 3;
    }
}
```

1.2 What is a range-based for loop? Give an example.

- A range-based for loop cycles through various items in a container. I have included an example below where a vector named 'range' is created and the compiler prints out the contents of the vector using the aforementioned looping sequence.
- Since the word range is synonymous with the phrase "sequence of elements," this means that we read the for *(int i : range)* loop like *for each int i in range*. We ultimately use range-based for loops to cycle over every element in a sequence–one at a time–until completed.

```cpp
// q02.cpp
// Solomon Himelbloom
// 18 October 2020
// CS 201 Midterm Exam

#include <iostream>
#include <stdio.h>
#include <string>
#include <vector>

using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;

// Question #2: What is a range-based for loop? Given an example.
int main() {
    cout << "Range-based for loop example:" << endl;

    std::vector<int> range{0, 25, 50};
    range.push_back(75);
    range.push_back(100);

    for (int i : range) {
        cout << i << endl;
    }
}
```

1.3 How does pass-by-value differ from pass-by-reference? When should you use const-pass-by-reference?

- There are various differences between **pass-by-value** and **pass-by-reference**. We use pass-by-value when the parameter is a copy of the argument. In other words, changes made inside the function are not reflected in comparison with the original value.
- This contrasts with **pass-by-reference** where no copy is made. We must also include an ampersand between the parameter's type and respective name. In other words, the parameter becomes an alias (instead of a copy) of the argument and can be considered another way to send information back to the caller and out of a function.
- The final example is **const-pass-by-reference** which is similar to the regular **pass-by-reference** since it avoids directly copying. The subtle change is that you are no longer able to make any changes inside the function – a great option if you were required to examine the value within the function without altering it.

1.4 What is the scope of an identifier?

- The scope of an identifier is the range of a program in which statements are recognized as valid names. Some rules to consider include: declaring types before they are used and recognizing that some identifiers have either a global or local scope. This means that we must be careful where we reference identifiers and dictate which functions they encompass.

1.5 Describe what is done during each step of the process: preprocessing, compiling, and linking.

- In **preprocessing**, we consider preprocessor directives throughout the source code as it is transitioned into source and header file(s). A crucial part of this step is preparation for the compiler.
- The next step is **compiling** where the C++ code is converted from one language to a generally lower-level programming language. This results in complied to machine-readable language which results in easy executables (stored in an object file as object code) to be used throughout the processor.
- We finally begin the third stage of the build process: **linking**. This important step ensures the conversion between compiled object code to code able to be executed. Another important note is that if linking did not take place, you might have multiple references to functions across other object files. Multiple objects are compiled with code into library files ensuring that the result is a single executable file with all required function references.

1.6 Why can conversion from a double to an int be a bad thing? Why is conversion from int to double ok?

- A conversion from a double to int can be considered 'a bad thing' or unsafe as your desired output should not sacrifice any data precision during this transition. Conversions from an int to a double are considered safe or 'ok' since the direction is from a smaller value to a bigger value – the type you are trying to convert to is smaller than your current type.

1.7 Write a C++ function print_range that takes two integers and prints each integer in the range between them. E.g. print_range(3, 8) would output 4 5 6 7.

- See the example below.

```cpp
// q07.cpp
// Solomon Himelbloom
// 18 October 2020
// CS 201 Midterm Exam

#include <iostream>
#include <stdio.h>
#include <string>

using std::cin;
using std::cout;
using std::endl;
using std::string;

void print_range(int lower, int higher) {
    int i = (lower + 1);
    while (i < higher) {
        cout << i << " ";
        i += 1;
    }

    cout << "\n" << endl;
    cout << "lower bound (excluded) = " << lower << endl;
    cout << "higher (excluded) = " << higher << endl;
}

// Question #7: Write a C++ function print_range that takes two integers
// and prints each integer in the range between them.
// E.g. print_range(3, 8) would output 4 5 6 7.
int main() {
    print_range(3, 8);
}
```

1.8 Write a C++ function isSorted that takes a vector of ints and returns true of a vector is sorted or false if is not.

- See the example below.

```cpp
// q08.cpp
// Solomon Himelbloom
// 18 October 2020
```

```cpp
// CS 201 Midterm Exam

#include <iostream>
#include <stdio.h>
#include <string>
#include <vector>

using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;

bool isSorted(vector<int> verify) {
    for (int i = 0; i < verify.size() - 1; i++) {
        if (verify[i] > verify[i+1]) {
            return false;
        }
        else {
            return true;
        }
    }
}

// Question #8: Write a C++ function isSorted that takes a vector of ints
// and returns true of a vector is sorted or false if is not.
int main() {
    vector<int> increasing { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    vector<int> decreasing { 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    vector<int> up { 1, 2, 3 };
    vector<int> down { 3, 2, 1 };

    // Boolean debug information:
    // 1 = sorted (true); 0 = not sorted (false)
    cout << "increasing vector: " << isSorted(std::vector(increasing)) << endl;
    cout << "decreasing vector: " << isSorted(std::vector(decreasing)) << endl;
    cout << "up vector: " << isSorted(std::vector(up)) << endl;
    cout << "down vector: " << isSorted(std::vector(down)) << endl;

    // Output information to the user based on if/else logic.
    if (isSorted(std::vector(increasing)) == 1) {
        cout << "The vector provided is sorted." << endl;
    }
    else {
        cout << "The vector provided is not sorted." << endl;
    }

    if (isSorted(std::vector(decreasing)) == 1) {
        cout << "The vector provided is sorted." << endl;
    }
    else {
        cout << "The vector provided is not sorted." << endl;
    }

    if (isSorted(std::vector(up)) == 1) {
        cout << "The vector provided is sorted." << endl;
    }
    else {
        cout << "The vector provided is not sorted." << endl;
    }

    if (isSorted(std::vector(down)) == 1) {
        cout << "The vector provided is sorted." << endl;
```

```
    }
    else {
        cout << "The vector provided is not sorted." << endl;
    }
}
```

1.9 Write a C++ function PrintSubStrings that take a string and prints all substrings in that string. E.g. PrintSubStrings("abc") would output a b c ab bc abc.

- See the example below.

```
// q09.cpp
// Solomon Himelbloom
// 18 October 2020
// CS 201 Midterm Exam

#include <iostream>
#include <stdio.h>
#include <string>
#include <vector>

using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;

void PrintSubStrings(std::string input, int permutations) {
    for (int i = 0; i < permutations; i++) {
        for (int length = 1; length <= permutations - i; length++) {
            cout << input.substr(i, length) << " ";
        }
    }
}

// Question #9: Write a C++ function PrintSubStrings that take a string
// and prints all substrings in that string.
// E.g. PrintSubStrings(\abc") would output a b c ab bc abc.
int main() {
    std::string output = "abc";
    PrintSubStrings(output, output.length());
    return 0;
}
```