**2017 Winter**

Q1) Strictly speaking and unlike ISO C, ISO C++ only supports pass-by-value when calling

functions.

1. True (b) False

Q2) The **first** official ISO C++ standard is associated with the year \_\_\_\_.

1998

Q3) To compile **C++14** code using GCC’s g++, the compiler option one must use is: \_\_\_\_\_.

-std=c++14

Q4) In C++, one can declare a variable whose type is a reference to a reference.

1. True (b) False

Q5) The creator of C++ originally created the language while working at \_\_\_\_\_. (Provide company name.)

Bell labs

Q6) All C++ types that have as their rightmost decoration one or two ampersands are known as \_\_\_\_\_ types.

reference

Q7) If a C++ compiler chose to implement a variable of type **char const\*&** as a pointer, what would the pointer

variable’s (exact equivalent) type be?

char const \* \* const

Q8) If a C++ compiler chose to implement a variable of type **Foo&** as a pointer, what would the pointer

variable’s (exact equivalent) type be?

Foo \* const

Q9) If a C++ compiler chose to implement a variable of type **Bar\*&&** as a pointer, what would the pointer

variable’s (exact equivalent) type be?

Bar \* \* const

Q10) Given the declaration, int i;, what does i++; return?

1. an l-value (b) an r-value (c) neither of these

Q11) Write a C++ expression that declares a reference to a temporary int object whose value is 5.

int&& r = 5;

Q12) A variable can be easily determined to be an l-value when it \_\_\_\_\_\_.

has a name

Q13) A variable can be easily determined to be an r-value when it \_\_\_\_\_.

doesn’t have a name

Q14) ISO C++’s evolution is driven by theoretical problems not by real ones.

1. True (b) False

Q16) Match the appropriate term with the best definition. The possible terms to choose from are:

imperative, modular, object-based, object-oriented, generic, functional

Computation is defined in terms of patterns with placeholders that act on common aspects of those placeholder’s interfaces.

**generic**

Computation is defined in terms of state that determines the expression of the next operation to be performed.

**object-based**

Computation is defined in terms of programming statements that describe changes in state.

**imperative**

Computation is defined in terms of hierarchically organized state that determines the expression of the next operation to be performed.

**object-oriented**

Computation is defined in terms of operations that have no side effects.

**functional**

Computation is defined in terms of programming statements that describe changes in state with the additional ability to expose/hide functions and/or variables as seen from other translation units or namespaces.

**modular**

Q17) Per ISO C++ rules, to use C’s <ctype.h> header in a C++ program, **explain** what one has to do to #include

the header in C++. Be sure to also write the final C++ #include code for <ctype.h> in your answer.

1) delete the ‘.h’ suffix from the header name

2) prefix a ‘c’ to the header name

e.g., <ctype.h> => <cctype>

Q18) All C Standard Library functions are contained in the **cstd** namespace.

1. True (b) False (std)

Q19) All C++ Standard Library functions are contained in the **cxxstd** namespace.

1. True (b) False (std)

Q20) Briefly, what is the **difference** between **cout** and **cerr**?

cout outputs to standard output

cerr outputs to standard error; cerr is is unbuffered

Q21) The Standard Library uses the \_\_\_\_ bit shift operator to write formatted data out to an ostream.

<<

Q22) The Standard Library uses the \_\_\_\_ bit shift operator to read in formatted data from an istream.

>>

Q23) Originally in C++, the **auto** keyword prefixed in front of a variable declaration in a function meant that the

variable was declared and placed on a function’s \_\_\_\_\_.

call stack

Q24) In the current C++ standard, using the **auto** keyword to declare a variable allows a programmer to do what

in C++?

e.g., one of:

1) omit the type when declaring a variable

2) omit the return type of a function

Q25) Using the current C++ standard, will writing **auto i;** in **main()** compile? Answer yes or no. If no, explain

why it will not compile.

No. The compiler must be able to determine the exact type of i, e.g., via constructor syntax, code to the right of the ‘=’ after the variable, etc.

Q26) Rewrite the function **double foo(int i) { return 3.14; }** to use the new C++ function suffix declarator

syntax. (Hint: Think “suffix” and know you need to use -> .)

auto foo(int i) **-> double** { return 3.14; }

Q27) Rewrite the function **double foo(int c) { return 3; }** so that it is a C++ lambda function. (Don’t overcomplicate

this!)

[](int c) **-> double** { return 3; }

NOTE: Unless 3 is changed to 3.0 the return type, i.e., -> double, must be present.

Q28) If a C++ I/O Stream type is cast to a **bool**, what do the **true** and **false** results mean?

true means no errors or EOF have occurred on the stream;

false means an error or EOF has occurred on the stream

Q29) The ISO C++ standards dictate complexity requirements for nearly everything in the C++

Standard Library.

(a) True (b) False

Q30) Write the code fragment to declare a variable called **values** whose type is a **std::vector** containing

**std::string** instances.

std::vector<std::string> values;

Q31) Write the code fragment to read in all **std::string** words from **std::cin** (using a single **while** loop) into the

variable **values** whose declaration you wrote in Q30. (An EOF or any occurrence of an error constitutes the end

of the words to be read in.) Remember to declare your **std::string** variable.

std::string word;

while (std::cin >> word)

values.push\_back(word);

Q32) Use the C++ **range-based for-loop** to output all words in the **values** variable (from Q30 and Q31) to

**std::cout**.

for (auto i : values)

std::cout << i << ‘\n’;

Q33) Call **<algorithm>’s std::for\_each** function passing the arguments required to output all words in the

**values** variable (from Q30 and Q31) to **std::cout** within a single C++ statement. Recall the prototype for

for\_each is:

template <typename Iter, typename UnaryPredicate>

UnaryPredicate for\_each(Iter first, Iter last, UnaryPredicate pred);

std::for\_each(values.begin(),values.end(),

[](auto e){std::cout << e;}

);

Q34) Given the **values** variable (from Q30 and Q31) write the complete code fragment that performs the

following:

• Starting at the beginning of **values**, search for the word “**the**” using **<algorithm>’s std::find**. Store the

result in a variable called “**the\_pos**”.

◦ NOTE: This must be done using a single C++ statement.

• Starting at “**the\_pos**”, search for the first word whose string length (i.e., string\_var.size()) is greater than

10 using <algorithms>’s **std::find\_if**. Store the result in a variable called “**big\_pos**”.

◦ NOTE: This must be done using a single C++ statement.

• As you did in Assignment 1, declare a **std::map** capable of representing a frequency histogram of words.

Call this variable “**hist**”.

◦ NOTE: This must be done using a single C++ statement.

• Using **<algorithm>’s std::for\_each**, add everything in the range **[the\_pos, big\_pos)** to the histogram

**hist** variable.

◦ NOTE: This must be done using a single C++ statement.

◦ Hint 1: Recall how the histogram was constructed in Assignment 1 which included using [].

◦ Hint 2: Use a lambda function to add the data to the histogram.

◦ Hint 3: Remember to capture something!

The prototypes for **std::find**, **std::find\_if** are as follows:

template <typename Iter, typename T>

Iter find(Iter first, Iter last, T const& value);

template <typename Iter, typename UnaryPredicate>

Iter find\_if(Iter first, Iter last, UnaryPredicate pred);

auto the\_pos = std::find(begin(values), end(values), “the”);

auto big\_pos = std::find\_if(the\_pos, end(values),

[](auto str) { return str.size() > 10; }

);

std::map<std::string, unsigned> hist;

std::for\_each(the\_pos, big\_pos,

[&hist](auto str) mutable { ++hist[str]; }

);

**2015 Winter**

Q1) The C++ is a **general-purpose**, **multi-paradigm**, dynamically-typed programming language.

1. True (b) False

Q3) A design objective of C++ is that it is more important to allow a useful feature than to prevent every misuse.

1. True (b) False

Q4) A design objective of C++ is to not allow any explicit violations of the static type system.

1. True (b) False

Q5) A design objective of C++ is to incur overhead for language features not used in a program.

1. True (b) False

Q7) The \_\_\_\_\_ programming paradigm uses encapsulation via translation units, namespaces, structs, unions,

and/or classes in C++ but it is limited in that there can **only be one run-time** instance of it.

modular

Q8) The \_\_\_\_\_ in C++ enables very powerful forms of compile-time polymorphism with the focus placed on

algorithms and type abstractions as patterns.

generic programming paradigm

Q9) The \_\_\_\_\_ programming paradigm defines computation in terms of programming statements that describe

changes in state.

Procedural (imperative)

Q10) Write the C++ symbol name(s) that stderr is conceptually equivalent to.

std::cerr std::clog

Q11) Write the C++ symbol name(s) that stdout is conceptually equivalent to.

std::cout

Q12) Write the C++ symbol name(s) that stdin is conceptually equivalent to.

std::cin

For Q13a through Q13f, your boss has asked you to transliterate a Java program into C++. Full full marks, ensure

your answer uses correct C++ terminology and is appropriately detailed / specific.

Q13a) All inheritance in Java would be implemented as \_\_\_\_\_ inheritance in C++.

virtual public

Q13b) All static methods in Java would be implemented as \_\_\_\_\_ member functions in C++.

static

Q13c) All non-static methods in Java would be implemented as \_\_\_\_\_ member functions in C++.

virtual

Q13d) All interfaces in Java would be implemented as \_\_\_\_\_ in C++.

abstract classes

Q13e) All abstract classes in Java would be implemented as \_\_\_\_\_ using \_\_\_\_\_ \_\_\_\_\_ functions in C++.

First blank: abstract classes

Second & third blanks: pure virtual

Q13f) Most enumerations in Java would be implemented as \_\_\_\_\_ in C++11. (Be specific!)

enum class

Q14) Public inheritance should **only** be used to model \_\_\_\_\_ \_\_\_\_\_ relationships.

is-a

Q15) In C++, the this symbol is always a \_\_\_\_\_ \_\_\_\_\_ to the type it represents.

constant pointer

Q16) In C++, by default all **struct inheritance** is \_\_\_\_\_ and all **members** are declared with \_\_\_\_\_ access.

public public

Q17) In C++, by default all **class inheritance** is \_\_\_\_\_ and all **members** are declared with \_\_\_\_\_ access.

private private

Q18) The acronym’s **RAII** expansion in words is: \_\_\_\_\_.

Resource Acquisition Is Initialization

Q19) Most C++ compilers implement exceptions in a way that the **cost** is \_\_\_\_\_ as long as exceptions are **not**

thrown.

essentially zero

Q20) Complete the sentence: The C++ language does **not** need the equivalent of Java’s finally clause **because**

\_\_\_\_\_. (Ensure your answer also justifies the statement.)

C++ has destructors which (by convention) clean up object-held resources when the call stack is unwound

Q21) Indicate whether or not each of the following **expressions** are **L-values** or **R-values** or an error (e.g., is

illegal in the language).

**C++11 Expression L(value) or R(value) or E(rror)**

int(5)

**R**

double d = 3; (Answer with respect to d.)

**L**

float f = 2.3F;

float& r = f;

(Answer with respect to r.)

**L**

long double a = -3.2e10L;

long double&& b = a;

(Answer with respect to b.)

**E**

short&& m = 345;

(Answer with respect to m.)

**L**

Q22) In Assignment 2, you manipulated an anonymous **union**:

class parse\_value

{ private:

parse\_value\_type type\_;

**union**

**{**

**char charval\_;**

**std::string stringval\_;**

**long long integerval\_;**

**long double realval\_;**

**};**

// ...

};

Explain (i) what the differences are between a union and a struct and (ii) what that allowed you to accomplish

in Assignment 2.

Answer (i): A union’s members all overlap in memory with each member’s starting address being the same as all other members; struct members are non-overlapping wrt memory. Only one member of a union can be stored at a time. All members of a struct are stored simultaneously.

Answer (ii): Using a union allowed one to store zero or one **value** of a set of types i.e., char, string, long long, and long double, within any instance of parse\_value. (This was accomplished without using pointers, inheritance, etc. using placement new instead.)

Q23) Your boss has asked you to convert his C program:

#include <stdio.h>

#include <stdlib.h>

int value = 0;

int func(int\* const p)

{

\*p += value;

value = \*p;

return abs(\*p \* 2);

}

int main()

{

int i;

while (scanf(“%i”, &i) == 1)

printf(“%i “, func(&i));

printf(“\n”);

return 0;

}

into a C++11 program replacing everything in C that has suitable C++ equivalents. Note that this means:

• you cannot use the C library at all in your answer,

• you need to replace everything that can be to a suitable C++ language construct –provided it is

semantically correct to do so and it simplifies the code/syntax, and,

• everything else must remain as-is.

Your answer must be a **full and complete valid C++11 program.**

#include <**iostream**>

#include <**cstdlib**>

int value**{}**;

int func(int**&** p){

**p += value;**

value = **p;**

return **std::**abs(**p** \* 2);

}

int main(){

int i;

while (**cin >> i**)

**cout << func(i);**

**cout << ‘\n’;**

return 0;

}

Q24a through Q24f make use of the following given struct:

struct X

{

double \*n; // Must be dynamically allocated below

};

All answers must be written as if there were **written inside** X’s definition. All dynamic allocation manipulations

must be appropriately loaded, stored, or applied to the **member variable** n.

You may **assume** “using namespace std;” and any and all necessary #include files **appear elsewhere**. **Only** write

the necessary code in your answer.

Q24a) Write the default constructor for X. It must **dynamically** allocate a **single double** variable set to **3.14**.

Absolutely no code can appear between the constructor’s braces (i.e., { and }).

X() :

n(new double(3.14))

{}

Q24b) Write the copy constructor for X. It must dynamically allocate a single double variable whose value is

copied from the object passed to the constructor. Absolutely no code can appear between the constructor’s braces

(i.e., { and }).

X(X const& x) :

n(x.n != nullptr ? new double(\*x.n) : nullptr)

{}

Q24c) Write the copy assignment operator for X with the correct semantics. It does not have to be exception-safe.

// NOTE: An exception safe version follows (other solutions possible)...

X& operator =(X const& x){

if (this != &x)

{

X tmpX = x; // Make copy

// Swap tmpX.n with this->n...

double \*tmp = tmpX.n;

tmpX.n = n;

n = tmp;

// and tmpX destructor will destroy its n (which is what was in X)

}

return \*this;

}

Q24d) Write the move constructor for X with the correct semantics.

X(X&& x) :

n(x.n)

{

x.n = nullptr;

}

Q24e) Write the move assignment operator for X with the correct semantics.

X& operator =(X&& x)

{

if (this != &x)

{

double \*tmp = n; // Save old n pointer value

n = x.n; // Copy x.n into n

x.n = nullptr; // Null out x.n

delete tmp; // Free memory tied to tmp

}

return \*this;

}

Q24f) Write the destructor for X with the correct semantics.

~X()

{

delete n;

}

**2014 Winter**

Q1) The C++ programming language was created by \_\_\_\_\_ (full name).

Bjarne Stroustrup

Q2) C++ was originally called \_\_\_\_\_.

C with classes

Q4) The current C++ standard in effect was standardized in what year?

2017

Q7) Briefly describe the zero overhead language design rule in C++.

You don't pay for what you don't use. If you don't use a language feature, you don't incur run-time costs for such features.

Q8) The imperative programming paradigm defines computation in terms of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

programming statements that describe changes in state

Q9) The modular programming paradigm allows \_\_\_\_\_: the ability to expose/hide functions/types defined in

different modules.

encapsulation

Q10) The difference between the object-based and the object-oriented paradigms is that the object-based

paradigm does not use/support \_\_\_\_\_\_\_.

inheritance

Q11) Match the following descriptions with the programming paradigm that best applies.

**Letter Description**

A "Decide which classes you want; provide a full set of operations for each class; make

commonality explicit using inheritance."

B "Decide which procedures you want; use the best algorithms you can find."

C "Decide which algorithms you want; parameterize them so that they work for a variety suitable

types and data structures.

D "Decide which modules you want; partition the program so that data is hidden within

modules."

E "Decide which types you want; provide a full set of operations for each type."

Write the letter of the most appropriate description corresponding to the programming paradigm below:

**E** Object-Based Paradigm

**C** Generic Paradigm

**D** Modular Paradigm

**A** Object-Oriented Paradigm

**B** Procedural Paradigm

Q12) C++'s template mechanism is Turing-complete.

1. True (b) False

Q14) Clearly explain what the differences, if any, are between T \*const and T const\*.

T \* const is a constant read-only pointer to a read-write instance of T.

T const\* is a read-write pointer to a constant read-only instance of T.

Q15) Write a C++11 lambda function that accepts a double as an argument and returns its value multiplied by

3.14.

[](double d) { return d\*3.14; }

Q16) The mathematician that designed key portions of the STL is \_\_\_\_\_ (full name).

Alexander Stepanov

Q18) Briefly describe what a C++ Standard Library iterator represents.

An iterator in C++ represents a pointer to an object/value stored within a container, or, it represents the one-past-the-end iterator position of the container. Iterators permit through the elements/objects stored in the container.

Q19) C++ Standard Library's iterators were modeled upon which C language construct?

pointers (and pointer arithmetic)

Q20) Briefly explain what is meant by a predicate in the C++ Standard Library.

A predicate is a function that returns a bool () value.

Q21) In C++ a Standard Library function requiring a predicate allows the predicate to be stateful.

1. True (b) False

Q22) In the C++ Standard Library, all sorting operations rely on a \_\_\_\_\_ (3 words; Hint: SWO) to sort things.

The default overloaded operator to perform such is \_\_\_\_\_.

3-word strict weak order

Operator less than

Q23) Your boss has just let go of an employee that wrote an insertion sort routine, called insertion\_sort(),

function using std::vector because he had asked for it to be written with std::list. Your boss thought he

could just change the typedef from vector<int> to list<int> but got all kinds of compiler errors! He has

**asked you** to **rewrite insertion\_sort()** function to **use iterators** so that it **works properly for all**

**bidirectional iterator inputs and random-access iterators**. He included a sample main() so you could see

how it will be invoked. Pseudocode for the insertion sort is as follows, given an array A whose first index is zero:

for i from 1 to length(A) {

j = i

while (j != 0 and A[j-1] > A[j]) {

swap(A[i], A[j-1])

j = j – 1

}

}

Your solution must only use valid bidirectional iterator operations. All comparisons must be done using < (i.e.,

less than). You must only use the for loop construct –not the while, do..while, or goto constructs.

**NOTE:** Write your answer on the next page. insertion\_sort() will sort [first,last) in ascending order.

#include <iostream>

#include <list>

#include <algorithm>

using namespace std;

template <typename Iter>

void insertion\_sort(Iter first, Iter last)

{

**// YOUR ANSWER GOES HERE.**

}

int main()

{

typedef list<int> LIST;

LIST stuff{ 32, 14, 45, -24, 6543, 7635, 2, -5, -23, -25, -242 };

insertion\_sort(begin(stuff), end(stuff));

for (auto const& i : stuff)

cout << i << ' ';

cout << endl;

}

Q23 Answer - Write **ONLY** the code **INSIDE** the provided **insertion\_sort()** function above. (If you do

otherwise then marks will be deducted.) You cannot change the function prototype.

// No need to sort if empty

if (first == last)

return;

// No need to sort empty list...

auto second = next(first);

if (second == last)

return;

// Above code is needed; i.e., some code to ensure that one does not go past

// the end when determining second.

for i from 1 to length(A) {

j = i

while (j != 0 and A[j-1] > A[j]) {

swap(A[j], A[j-1])

j = j – 1

}

}

for (auto i=second; i != last; ++i)

{

for (auto j=i; j != first && \*j < \*prev(j); --j)

iter\_swap(j, prev(j));

}

// NOTE: Typo in pseudocode A[i] should be A[j]. This affects

// iter\_swap() code. A[i] results in iter\_swap(i, prev(j));

**2013 Winter**

Q4) Explain the key differences between (i) modular and object-based, and, (ii) object-based and object-oriented

programming.

(i) Modular has only one instance of the module whereas in object-based the module is the object and therefore one can have multiple instances.

(ii) Object-based programming does not allow/permit one to use inheritance; object-oriented programming does.

Q6) Briefly explain **why** the C programming language can **only** be said to **truly** support pass-by-value.

C does not support references. Although C does support pointers, they are always copied when passed to functions and the programmer must also explicitly specify the address as a value when passing that argument. Thus, each pointer argument is pass-by-value.

Q7) C++11 supports two types of references. What are they called?

Answer A: L-value references (i.e., & as right-most symbol in a type declaration)

Answer B: R-value references (i.e., && as right-most symbol in a type declaration)

Q9) If the reference to some type, T, where to be written as T&, what would the semantically equivalent pointer

type declaration to that reference be written as?

T \* const \*

Q10) Using big-O, little-o, and omega complexity symbols, explain what the **cost of moving data** is in terms of

**copying data** and **copying pointers to the data**. Also, is the cost of **moving data** ever zero?

O(moving data) = o(copying data), i.e., moving is no worse than copying

Ω(moving data) = Ω(copying pointers to the data), i.e., moving is at least copying pointers

Ω(moving data) is never zero, i.e., moving is never free of cost (clearly assuming that there some data to actually move!).

Q11) One should view a \_\_\_\_\_ [1] operation as an optimized \_\_\_\_ [2] operation. (Hint: Q10.)

Answer 1: moving

Answer 2: copying

Q12) Write a C++11 lambda function that accepts an int as an argument and returns twice its value (as an int).

[](int i){return i\*2;}

Q14) Briefly describe what a C++ Standard Library container represents.

A container is an entity / a data structure that can hold many values all of the same type.

Q17) In C++, object-oriented programming provides run-time polymorphism whereas its \_\_\_\_\_ programming

provides compile-time polymorphism.

generic (template)

Q21) Why is C++'s use of the three-word answer with only a single operator (i.e., in Q20) better to use for sorting

than using two operators. Briefly explain.

Using one operator ensures consistent semantics for the equivalence and less than operations since both are defined in terms of the one operator. If two operators are used, it is more easily possible for their return values to violate the ordering semantics required for the sort and therefore result in an invalid sort / corrupted data structure.

Q22) Why is C++'s use of the three-word answer with only a single operator (i.e., in Q20) enable one to do that

cannot be done with a total order? Briefly explain.

Strict weak order (SWO) semantics allow both (a < b) and (b < a) to both return false. This is not possible with total order semantics since (a < b) => not (b < a). In effect, a SWO permits one to compare things that cannot be compared. Since equivalence is defined to be !(a < b) && !(b < a), this places all incomparable elements into the same equivalence class.

Q23) Briefly explain what the differences between a forward iterator and a bidirectional iterator are in terms of

their permitted operations.

A bidirectional iterator also permits traversing in reverse order (from any position) using the decrement operators (prefix and postfix) in addition to the in-order traversal of forward iters.

Q24) Clearly describe what the C++ compiler adds to a C++ class/struct when one declares a virtual member

function in it. Briefly what allows the correct virtual member function to be correctly invoked from all derived

classes.

The C++ compiler will internally add a function pointer for each virtual member function. These function pointers are typically placed into an array pointed to by a internally added pointer in the class/struct since this array is identical for all instances of that class/struct. This array is often called the "vtable". The compiler tracks which function signature is placed at which offset into the array since all types derived from this type will invoke the offset in the array when accessing a virtual function with the same signature. If a derived class overrides a virtual member function, then its vtable has a function pointer to its overridden version instead of the base version. Since the same offset is called for the same signature code having a pointer/reference to a base type will always call the correct virtual function via its vtable.

Q25) For a user-defined struct/class type T, write the member prototypes for the following:

**Member Prototype**

Default constructor

T();

Copy constructor

T(T const&);

Copy assignment operator

T& operator =(T const&);

Move constructor

T(T&&);

Move assignment operator

T& operator =(T&&);

Destructor

~T();

Q26) Briefly explain how a programmer would (conceptually and) properly use the Resource Acquisition Is

Initialization (RAII) design pattern.

He/she would ensure that the constructor allocated/acquired any needed resources and the destructor would deallocate/release any resources allocated/acquired during the lifetime of the object.

Q20) Write the full-and-complete program that would be placed inside main() (include any variable declarations

needed) to read in from standard input an **unknown** number of ints and output to standard output their sum.

You are not allowed to use any containers. You are only allowed to declare **at most two** variables and can only

use **one loop** construct. Your program must work correct even in the presence of input failures and errors (i.e.,

stop summing on the first failure, error, or EOF). You do not need to worry about integer overflow.

#include <iostream>

int main()

{

using namespace std;

int sum = 0;

for (int i; cin >> i; sum += i)

;

cout << sum << '\n';

return 0;

}

Q13) The Standard Library’s IOStream classes (e.g., istream and ostream) support the ability to throw exceptions

but such is disabled by default.

1. True (b) False

Q19) Explain what the expression **decltype(foo(‘a’)) value = foo(‘a’);** does and what is the type of value?

decltype(expression) is used to take the variable type of the expression and assign it to a new variable. In this case, value has the same variable type as the return type of foo().

Q20) For each table row, indicate if the run-time type result of the expression is a lvalue (L), rvalue (R), or a

syntax error (E). **[5 marks]**

**C++ Expression lvalue (L), rvalue (R), error (E)**

int{};

**R**

decltype(x+y) z;

**L**

auto x = 5;

**L**

q += 5;

**L**

++a;

(a is an int variable.)

**L**

b++;

(b is an int variable.)

**R**

int&& q = 56;

**L**

double&& r = z;

**E**

int& s = 56;

**E**

double& t = z;

**R**

Q23) Your boss has asked you to convert his C99 program into either a C++11 or C++14 program as specified

below. Although the C99 program:

◦ reads the data into an array, you must read all data into a linked list (i.e., use std::list),

◦ reads up to 5 elements into the data array, you must read in as many elements as possible from standard

input,

◦ uses pointers with the bubble\_sort function, you must change bubble sort to use iterators instead of

pointers, i.e., the prototype for your bubble\_sort() will become:

▪ template <typename Iter>

void bubble\_sort(Iter first, Iter last);

▪ NOTE: You are not allowed to called std::swap() in bubble\_sort(). You are also not allowed to use std::list::sort() in main() --bubble\_sort() must be preserved and implemented.

You must preserve all remaining aspects of program logic converting C99 code to use appropriate equivalent C+

+ language constructs and C++ Standard Library functions where possible. You don’t need to rewrite any

comments in your answer –just translate the code.

#include <stddef.h> // For size\_t

#include <stdio.h>

void bubble\_sort(int\* first, int\* last){

bool swapped;

do{

swapped = false;

int \*prev = first;

int \*cur = first; ++cur; // cur is one ahead of prev

for (; cur != last; ++prev, ++cur){

if (\*cur < \*prev)

{

int tmp = \*cur; \*cur = \*prev; \*prev = tmp; // Swap values

swapped = true;

}

}

}

while (swapped);

}

int main(){

// Declare the data container...

int data[5];

size\_t num\_read\_in = 0;

// Read in the data...

for (int i; scanf(“%i”, &i) == 1 && num\_read\_in < 5; ++num\_read\_in)

data[num\_read\_in] = i;

// Sort the data...

bubble\_sort(data, data+num\_read\_in);

// Output the data...

for (size\_t i=0; i != num\_read\_in; ++i)

printf(“%i ”, data[i]);

printf(“\n”);

return 0;

}

Important:

• You cannot use the C Standard Library at all in your answer unless there is no C++ equivalent for such.

• You need to replace everything that can be to a suitable C++ language or C++ Standard Library

construct –provided it is semantically correct to do so and it simplifies the code/syntax unless otherwise

instructed.

• Except when implementing the required changes, the steps the code performs else must remain as-is or

be equivalent to the original program code, i.e., don’t transform the program logic to use different

algorithms, optimize the code, or “make the program better”, etc.