“I certify that I have neither received nor given un-permitted aid on this examination and that I have reported all such incidents observed by me in which un-permitted aid is given.”

Signature SOLUTIONS
Name ___________________________
Student ID _______________________

Question 01 ------------ / [4]
Question 02 ------------ / [8]
Question 03 ------------ / [8]
Question 04 ------------ / [6]
Question 05 ------------ / [4]
Question 06 ------------ / [3]
Question 07 ------------ / [3]
Question 08 ------------ / [6]
Question 09 ------------ / [4]
Question 10 ------------ / [6]
Question 11 ------------ / [5]
Question 12 ------------ / [15]
Question 13 ------------ / [18]
Question 14 ------------ / [10]

Total: ___________/100
**Q 1:** You are to find the average of some integers. Assume that you store the sum of those integers in an `int` variable `sum` and another `int` variable `n` contains the total number of integers. You want to store the average in a `float` `f`. What statement in old C would accomplish this? What statement would you use if you were to use C++?

Statement in C: \( f = \frac{\text{float} \sum}{n}; \)

Statement in C++: \( f = \text{static\_cast\_float} \frac{\sum}{n}; \)

**Q 2:** Consider the following code. Write the output in the box provided on the right.

```cpp
#include <iostream>
using namespace std;

void foo(int x)
{
    int& y = x;
    y++;
}

void bar(int& x)
{
    int& y = x;
    y++;
}

void foobar(int& x)
{
    int y = x;
    y++;
}

int main()
{
    int i = 291;
    cout << i << endl;
    foo(i);
    cout << i << endl;
    bar(i);
    cout << i << endl;
    foobar(i);
    cout << i << endl;
    return 0;
}
```

```
291
291
292
292
```
Q 3: Consider the following two functions:

```cpp
int& foo() {
    int i = 3;
    int& j = i;
    return j;
}

int bar() {
    int i = 3;
    int& j = i;
    return j;
}
```

Answer the following questions:

I) Is it okay to define the function `foo()` as it is done? If not, then why?

   It is not okay. This is because we are returning a reference to a local variable. We should point out that it will be okay to receive the return value in an `int` rather than an `int&`. However, since the function prototype indicates a return by reference, the clients can not be restricted to receive the value in an `int` and they will be able to receive the return value in a reference causing runtime errors.

II) Is it okay to define the function `bar()` as it is done? If not, then why?

   It is okay. This is because we are returning from the function by value. Once again, the client may receive the return value in a reference in which case, that reference will refer to an unnamed temporary but the two reasons it is still okay to define the function as is are:

   a) The function prototype dictates a return by value suggesting clients to receive the return value not as a reference.

   b) Even if the client receives the return value in a reference which would refer to an unnamed temporary, the compiler will force the client to receive the value as a `const` reference.

III) Is it okay to do each of the following in a `main()` function? If not, then why?

   (a) `int x = foo();`

      Okay but not suggested/encouraged by the function prototype.

   (b) `int& x = foo();`

      Not okay for the reason given in (I).

   (c) `int x = bar();`

      Okay.

   (d) `int& y = bar();`

      Not okay. The reference `y` would refer to an unnamed temporary and must be initialized as a `const` reference.
Q 4: In our popular Student class, the name of the student is stored as a C++ string. Not as a pointer and not as a reference; just a plain C++ string. Assume that the class provides a getter as:

```cpp
const string& getName() const;
```

(a) The function could have returned a non-reference; what is the motivation in making the return type a reference? State the reason in one short sentence, no stories!

The sole reason for returning a reference is to avoid copying of the return object. Thus, returning a reference would be more efficient.

(b) Assume that we decided that the return type will be a reference for the reason you stated in the above part. Is it absolutely necessary to make the return type a `const`? Why? Why not?

It is absolutely necessary to make the return type a `const`, the compiler will complain otherwise. This is because the method is declared as a `const` method and must not allow modification of the object on which it is called. If the reference is returned without `const` qualifier, the client would have the ability to modify the object. Thus, if a `const` method returns data by reference, it must return the reference as a `const`.

(c) In part (b) above, whether necessary or not, is there any motivation in making the return type a `const`?

The other motivation to return a `const` reference is to prohibit the clients to modify our local data. Thus, if the method were not declared as a `const` method, we would have been able to return a non-`const` reference but we would not want to do that to avoid accidental damage to our local data.

Q 5: Give at least two restrictions that C++ places on operator overloading?

- Each overloaded operator must include at least one user-defined class
- You can not define new operators
- You can not change the order of evaluation and precedence of operators
- You can not overload ::, ., *., and ternary operators

Q 6: Can we overload `+` operator as a class method? If not, why?

Yes. There is no reason why not!

Q 7: Can we overload insertion operator ("<<") as a class method? If not, why?

No, we can not. This is because the function which is called for each overloaded operator must either be a global function or the class method of the object that appears on the left hand side of the operator. In case of insertion operator, the left hand object is not defined by the user (e.g., cout and cin are predefined), and writing a class method corresponding to the insertion operator would require us adding a class method to the classes to which cin and cout belong – we certainly would not want to do that.
Q 8: Increment operator can be overloaded either in the prefix form or in the postfix form. This means that the compiler will call two different functions: one for prefix-increment as in \texttt{++obj} and the other for postfix-increment as in \texttt{obj++}. How does the compiler know which function to call? Give an example using \texttt{BigNum} class. Just provide the declaration of the two functions. We don’t need definitions.

The compiler uses an additional [dummy] int parameter to distinguish between the two function calls.

For the \texttt{BigNum} class, when implemented as class method:

\begin{verbatim}
const BigNum& operator++(); // prefix form, called for ++obj
const BigNum operator++(int); // postfix form, called for obj++
\end{verbatim}

When implemented as a global function, each form takes an additional parameter which would be reference to the \texttt{BigNum}

\begin{verbatim}
const BigNum& operator++(BigNum&); // prefix form, global
const BigNum operator++(BigNum&, int); // postfix form, global
\end{verbatim}

Q 9: Come up with an indigenous definition of generic programming? You just have two sentences. Answers longer than two sentences will not receive any credit.

- Programming which allows users to write programs without worrying about the type of the objects.
- A way of programming in which generic/arbitrary data types are passed as parameters to constructs.
- Programming independent of the representation details of objects.

Q 10: Name three big things which make up the standard template library (STL)?

- Containers
- Algorithms
- Iterators

Q 11: Overloading of insertion operator ("<<") was shown to be taking an \texttt{ostream} reference as a parameter. Is it also okay to pass the \texttt{ostream} by value? Why? Why not?

Since the overloaded function will modify the \texttt{ostream} object (otherwise what is the point of writing such a function!), the corresponding function must take a non-\texttt{const} reference to the object which will be modified.
Q 12: Consider the following code that tries to keep track of the total number of objects of a class present at a given time. The code compiles and runs but there are logical errors.

```
// file foo.h
#ifndef _FOO_H_
define _FOO_H_

class Foo
{
public:
  Foo();
  ~Foo();
  static int getNumInstances()
  { return numInstances; }

private:
  static int numInstances;
};

#endif /* _FOO_H_ */

// file foo.cc
#include "foo.h"

int Foo::numInstances = 0;

Foo::Foo()
{
  numInstances++;
}

Foo::~Foo()
{
  numInstances--;
}
```

(a) What is the logical flaw in the above code?

The code does not give consideration to those objects which are created using copy constructor. Thus, the counter variable `numInstances` will only include those objects which are not constructed by the copy constructor. Recall that a copy constructor is provided by the compiler and we must explicitly write that constructor to obtain the desired results.

(b) Write a `main()` function that will expose the flaw in above code:

```
#include "foo.h"
#include <iostream>
using namespace std;

int main()
{
  Foo f;
  cout << Foo::getNumInstances() << endl; // output is 1 – correct
  Foo g;
  cout << Foo::getNumInstances() << endl; // output is 2 – correct
  Foo h(f); // copy constructor called
  cout << Foo::getNumInstances() << endl; // output is 2 – wrong!!!
  return 0;
}
```

(c) How the flaw can be fixed? Provide code that should go in foo.h and/or foo.cc file.

In your header file foo.h, in the public section of the class definition, add the following:

```
  Foo(const Foo& other);
```

In your implementation file foo.cc, add the following:

```
  Foo::Foo(const Foo& other)
  {
    numInstances++;
  }
```

----- OR ----- 

Just add the following in the header file:

```
  Foo(const Foo& other) (numInstances++;
```
Q 13: Consider the following class definition which holds an array of integers:

```cpp
class IntArray
{
public:
    IntArray(int size);
    IntArray(const IntArray& source);
    const IntArray& operator=(const IntArray& rho);
    int operator[](int index) const;
    int& operator[](int index);
    ~IntArray();
private:
    int* elements;
    int size;
};
```

We have declared six class methods in the class definition. Above definition appears in a header file. You are asked to provide the code that must go in the implementation file. That is, write the code for six class methods declared in above class definition. Make sure you include all header files necessary within the implementation file.

```cpp
#include <cassert>
IntArray::IntArray(int size)
{
    this->size = size;
    this->elements = new int[this->size];
    for (int i = 0; i < this->size; i++)
        this->elements[i] = 0;   // initialization
}
IntArray::IntArray(const IntArray& array)
{
    this->size = array.size;
    this->elements = new int[this->size];
    for (int i = 0; i < this->size; i++)
        this->elements[i] = array.elements[i];
}
IntArray::~IntArray()
{
    delete[] elements;
}
const IntArray& IntArray::operator=(const IntArray& src)
{
    if (this != &src) {
        delete this->elements;
        this->size = src.size;
        this->elements = new int[this->size];
        for (int i = 0; i < this->size; i++)
            this->elements[i] = src.elements[i];
    }
    return *this;
}
int IntArray::operator[](int i) const
{
    assert(i >= 0 && i < size);
    return elements[i];
}
int& IntArray::operator[](int i)
{
    assert(i >= 0 && i < size);
    return elements[i];
}
```
Q 14: Consider the following piece of code which depends upon the C++\texttt{string} class:

```
#include <iostream>
#include <string>

using namespace std;

class A
{
    public:
        static const string *vote(string party, string governor,
                        const string *vpp, const string& vp)
        {
            string democrat(*vpp);
            party = democrat;
            governor[1] = governor[0];
            return &vp;
        }

    static void candidates()
    {
        string gore("Al");
        string bush = gore;
        string& nader = bush;
        string buchanan;
        buchanan = gore;
        nader = *vote("Republican", bush, &gore, gore);
    }
};

int main()
{
    A::candidates();
    return 0;
}
```

On the next page, you are required to specify the ordering of all the calls to the \texttt{string} memory management functions. Here are the ones that you will consider:

1. The default constructor
2. The \texttt{char*} constructor
3. The copy constructor
4. The \texttt{operator=} assignment method
5. The destructor

Important:

1. Consider parameter setup when making function calls.
2. Assume that method arguments are always evaluated from left to right. That is, in a call to \texttt{foo(CustomString cs, Student s, int i)}, \texttt{cs} is constructed first, followed by \texttt{s}, followed by \texttt{i}.
3. On the next page, we have provided the start of your solution…
Answer:
Order of calls……
1. char* constructor for gore
2. copy constructor for bush
3. default constructor for buchanan
4. operator= method for assigning gore to buchanan
5. char* constructor for party
6. copy constructor for governor
7. copy constructor for democrat
8. operator= for assigning democrat to party
9. destructor for democrat
10. destructor for governor
11. destructor for party
12. operator= for assigning gore to nader
13. destructor for buchanan
14. destructor for bush
15. destructor for gore