Name _____

CISC 3142 – Programming Paradigms in C++
Fall '22
Exam #2

Make sure you follow adhere to the following:

- proper parameter transmission return value transmission, and const receiver
- proper use of member/friend non-member functions.
- please do not write any inline functions… put all function bodies in the `.cpp` file.
- please use member initialization lists in your constructors.
- if writing a file, please be complete… #include's, using's etc

1. You are to code a module pair, `arr_utils.h/.cpp that` provides two non-member (i.e., C-like) functions:

   - `reverse`: accepts an array and the number of elements and reverses the array in-place.
   - `write`: accepts a file name, an array, and the number of elements, and writes the elements out to the file

   For extra credit, use pointer and pointer notation (rather than subscripts) in the headers and bodies

   a. Code the `arr_utils.h` file here:

```
#ifndef ARR_UTILS_H
#define ARR_UTILS_H

#include <string>

void reverse(int arr[], int n);
void write(std::string filename, int arr[], int n);

#endif


I did not deduct for the omission of a #include guard or #pragma once
```

b. Code the `arr_utils.cpp` file here:

```
#include <fstream>
#include <string>

#include "arr_utils.h"

using namespace std;

void reverse(int arr[], int n) {
	for (int i = 0; i < n/2; i++)
		int t = arr[i];
		arr[i] = arr[n-i-1];
		arr[n-i-1] = t;
	}
}

void write(string filename, int arr[], int n) {
	ofstream ofs(filename.c_str());
	if (!ofs) throw "File not found";

	for (int i = 0; i < n; i++)
		ofs << arr[i];
	ofs.close();
}
```

2. Write a C++ program that reads the file `numbers.text` (containing a sequence of integers) into an array, (using the >> operator) reverses the array by calling `reverse` of *question 1*, and then writes the resulting array to the file "`reversed.text`" using the `write` function of *question 1*. You can assume not more than 100 numbers.

```
#include <iostream>
#include "arr_utils.h"

using namespace std;

int main() {

   ifstream ifs("numbers.text");

   int arr[100] arr;

   int size = 0;

   ifs >> arr[size];
   while (ifs) {
         size++;
         ifs >> arr[size];
   }

   reverse(arr, size);
   write("reversed.text", arr, size);
}
```

3. You are to code a class pair, `person.h/.cpp`, for the definition of a `Person` class, containing the following:

   - two data members: a name (string) and an age (integer).
   - a constructor accepting a name (defaults to "John Doe") and age (defaults to 0)
   - a `<<` operator that prints out the right operand to the left operand; e.g.; `cout << p`, where `p` is declared to be a `Person` object
   - a `>>` operator that reads the right operand from the left operand; e.g. `cin >> p`, where `p` is declared to be a `Person` object.

   Don't forget the << and >> operator should both work for standard streams (`cout` and `cin` respectively) as well as files.

   a. Code the `person.h` file here:

```
class Person {
   friend operator <<(ostream &os, const Person &person);
   friend operator >>(istream &os, Person &person);
public:
   Person(string name="John Doe", int age=0);
private:
   String name;
   int age;
};
```

b. Code the `person.cpp` file here:

```
#include <iostream>
#include "person.h"

using namespace std;

ostream &operator <<(ostream &os, const Person &person)
   os << person.name << " " << person.age;
   Return os;
}

istream &operator >>(istream &os, Person &person) {
   is >> person.name >> person.age;
   return is;
}

Person::Person(string name, int age) : name(name), age(age) {}
```

4. Write a full class definition (`.h` and `.cpp` files) of a class named `Integer` with the following:

- an integer data member
- a constructor that accepts an integer used to initialize the data member. The parameter should have a default of 0.
- A `<<` operator that prints the value of the data member to the specified stream
- A `>>` operator that inputs the data member from the specified stream
- A `+` operator (adds two integers returning a third `Integer` object)
- A `+=` operator (in-place addition)
- A `-` (negation) unary operator; returns an `Integer` containing the negation of the receiver (which remains unchanged)

a. Code the `integer.h` file here

```
class Integer {
   friend ostream &operator <<(ostream &os, const Integer &i);
   friend istream &operator >>(istream &os, Integer &i);
   friend Integer operator +(const Integer &i1, const Integer &i2);
public:
   Integer(int val = 0);
   Integer &operator +=(const Integer &i);
   Integer operator -() const;
private:
   int value;
};
```

b. Code the `integer.cpp` file here

```
#include "iostream"
#include "integer.h"

Using namespace std;

ostream & operator <<(ostream &os, const Integer &i) {
   os << val;
   return os;
}

istream &operator >>(istream &os, Integer &i) {
   is >> val;
   return is;
}

Integer operator +(const Integer &i1, const Integer &i2) {
   Integer result = i1;
   return result += i2;
}

Integer::Integer(int val) : val(val) {}

Integer &Integer::operator +=(const Integer &i) {
   val += i.val;
   return *this;
}

Integer Integer::operator -() const {
   Integer result(-val);
   Return result;
}
```

5. Turn the functions of *question 1* into function templates

```
template <typename T>
void reverse(T arr[], int n) {
  for (int i = 0; i < n/2; i++)
        T t = arr[i];
        arr[I] = arr[n-i-1];
        arr[n-i-1] = t;
  }
}

template <typename T>
void write(string filename, T arr[], int n) {
  ofstream ofs(s.c_str());
  if (!ofs) throw "File not found";

  for (int i = 0; i < n; i++)
      ofs << arr[i];
  ofs.close();
}
```

6. Revisiting *question 2*…

   a. Show the lines of code that change if the program is supposed to read, reverse and write a file of `Person` data.

```
Person arr[100];

… and probably the name of the file (probably not “numbers.text”)
```

b. Suppose the program also printed out the sum of values in the files. Would that change anything for *part a*?

Will fail on compilation error when attempting to calculate sum… `Person` class has no + operator

c. Would *part b* be answered differently if *part a* was done for `Integer`

Would work fine even with the sum, since a `+` operator has been defined for `Integer`

(Optional) Turn the `Integer` class of *question 4* into a `Number` template where the data member is to be instantiated. That is, one should be able to instantiate the `Number` class to contain int values, or double values (or long int values, etc)

```
#ifndef NUMBER_H
#define NUMBER_H

#include <iostream>

template <typename T>
class Number {
     friend std::ostream &operator <<(std::ostream &os, const Number<T> &num) {
          os << num.val;
          return os;
     }
     friend std::istream &operator >>(std::istream &is, Number<T> &num) {
          is >> num.val;
          return is;
     }
     friend Number<T> operator +(const Number<T> &num1, const Number<T> &num2) {
          return Number(num1.val) += num2;
     }
public:
     Number(T val = 0) : val(val) {}
     Number &operator +=(const Number num) {
          val += num.val;
          return *this;
     }
     Number operator -() {
          return Number(-val);
     }
private:
     T val;
};

#endif
```

Does *question 2* – using a `Number` instantiated for an `int` still work for the `Number` class? What about if it is instantiated for a `double`?

Yes because there is a + operator