

Name _____

CISC 3142 – Programming Paradigms in C++
Fall '22
Exam #1
Solutions

1. **(15 points)** The file `numbers.data`, consists of groups of integers, each preceded by a header value. For example:

```
3  4 7 1
4  2 4 6 8
```

Write a complete C++ program that reads the file, prints out the maximum value in each group; when finished (i.e., when all the data has been read), the maximum value across all groups should be printed, as well as the number of groups processed. For example, given the above file, the output would be:

```
Max of group #1: 7
Max of group #2: 8
Overall max: 8
2 groups processed
```

- Your functions should come after your `main` function.
- Make sure you include all necessary `#includes`, etc.,
- Make sure you include the logic for testing the successful opening of the file-- if the file is not there... print out *"File not found"* and terminate the program.

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

int main() {
    ifstream infile("numbers.data");
    if (!infile) {
        throw string("File not found");
        exit(1);
    }

    int globalMax;
    bool first = true;

    int header;
    int numGroups = 0;
    infile >> header;
    while (infile) {
        numGroups++;
        int max;
        infile >> max;
        for (int i = 1; i < header; i++) {
            int n;
            infile >> n;
            if (n > max) max = n;
        }
        cout << "max: " << max << endl;
        if (first || max > globalMax) globalMax = max;
        if (first) first = false;
        infile >> header;
    }
    cout << "global max: " << globalMax << endl;
    cout << "groupsProcessed: " << numGroups << endl;
}
```

2. **(15 points)** Write a Java class named `PosInt`, that represents positive integers, i.e., integers greater than 0, with the following behavior:

- An integer instance variable to contain the value
- A constructor that accepts an integer argument that is assigned to the instance variable. If the argument passed is not positive, an `Exception` is thrown with the message *"Constructor passed non-positive value"*.
- A default constructor that initializes the value of the instance variable to 1. For full credit, you must use the integer-argument constructor when implementing this method.
- A copy constructor that accepts a `PosInt` object and copies its values to the new (receiver) object.
- A `copy` method that accepts a(nother) `PosInt` argument and copies the argument to the receiver. The receiver is returned as the value of the function.
- An `addInPlace` method that accepts a(nother) `PosInt` argument and adds the argument to the receiver. The receiver is returned as the value of the function.
- An `add` method that accepts a(nother) `PosInt` argument and returns the sum as a new `PosInt` object.

For full credit, you must leverage:

- either `addInPlace` using `add`, or `add` using `addInPlace`.
- the copy constructor using the `copy` method

```
class PosInt {
    PosInt(int val) throws Exception {
        if (val <= 0) throw new Exception("Negative");
        this.val = val;
    }
    PosInt() throws Exception {this(1);}
    PosInt(PosInt posInt) {copy(posInt);}

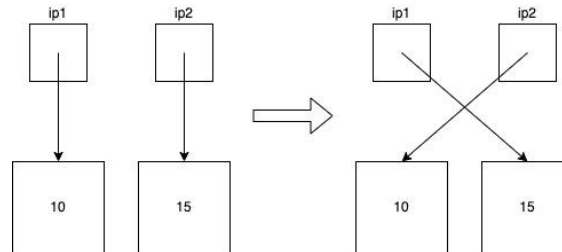
    void copy(PosInt posInt) { val = posInt.val;}

    PosInt addInPlace(PosInt posInt) {
        val += posInt.val;
        return this;
    }
    PosInt add (PosInt posInt) {
        PosInt result = new PosInt(posInt);
        return result.addInPlace(posInt);
    }

    private int val;
}
```

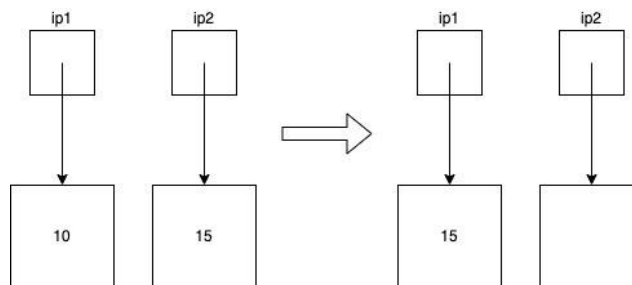
3.

- a. **(5 points)** The variables `ip1` and `ip2` have both been declared as pointers to integers, and have been assigned values (i.e., they are each pointing to an integer value). Write the code to exchange the values of these two variables (so that after the swap `ip1` points to what `ip2` originally pointed to and vice-versa-- in other words, in this exercise, you are swapping the pointers). Declare any necessary variables.



```
int *tp = ip1;
ip1 = ip2;
ip2 = tp;
```

- b. **(5 points)** The variables `ip1` and `ip2` have both been declared as pointers to integers and have been assigned addresses. Write the code to exchange the two integers (so that after the swap `ip1` still points at the same location, but it now contains the integer value originally contained in the location pointed to by `ip2`; and vice versa-- in other words, in this exercise you are swapping the integers, not the pointers). Declare any necessary variables.



```
int t = *ip1;
*ip1 = *ip2;
*ip2 = t;
```

- c. **(5 points)** Write a function that swaps two integers using call-by-reference

```
void swap(int &x, int &y) {  
    int t = x;  
    x = y;  
    y = t;  
}
```

- d. **(3 points)** Write the code to call the function of *part c* for the integer variables `x` and `y`

```
swap(x, y);
```

- e. **(5 points)** Write a function that swaps two integers by passing pointers to the integers (i.e., ‘*call-by-pointer*’)

```
void swap(int *xp, int *yp) {  
    int t = *xp;  
    *xp = *yp;  
    *yp = t;  
}
```

- f. **(3 points)** Write the code to call the function of *part e* for the integer variables `x` and `y`

```
swap(&x, &y);
```

4.

- a. **(10 points)** Write a function, `find`, that accepts an array of integers, the number of elements in the array, and an integer value, and returns the position (index) of the value in the array, if the integer value appears in the array, and `-1` if it is not in the array.

```
int find(int arr[], int n, int val) {  
    for (int i = 0; i < n; i++)  
        if (arr[i] == val) return i;  
    return -1;  
}
```

- b. **(5 points)** Write some code that declares, and initializes, an array to the values `1...10`, prompts the user for a number, searches the array (using `find`) for the number, and prints the result. You do not have to write a full program, just the code to accomplish the this.

```
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
  
cout << "##? ";  
cin >> val;  
  
cout << find(a, 10, val) << endl;
```

5.

a. **(8 points)** Recode the `find` function of *question 4* as a function template

```
template <typename T>
int find(T arr[], int n, T val) {
    for (int i = 0; i < n; i++)
        if (arr[i] == val) return i;
    return -1;
}
```

b. **(Optional 5 points)** *Part b* of *question 4* and *part b* of *question 6* ask you to write code that uses the function and class template of their *part a*'s respectively. Why am I not asking that of you here?

```
Instantiating a function template by calling it with arguments looks
just like a function call itself (unless there's an ambiguity and one
must provide an explicit instantiation). So this would look just like
part b of #4
```

6.

a. **(12 points)** Code a `Pair` class template, but unlike the one presented in class, the types of the two elements of this `Pair` can be different. The template should include the following member functions:

- Two data members of (possibly) different types
- A constructor that accepts two parameters and assigns them to the first and second data members respectively.
- `getFirst` and `getSecond` member functions that return the corresponding data member
- a `print` function that prints the pair in the format:
(*first*, *second*)

```
template <typename F, typename S>
class Pair {
public:
    Pair(F f, S s) : first(f), second(s) {}

    F getFirst() {return first;}
    S getSecond() {return second;}

    void print() {
        cout << "(" << first << ", " << second << ")";
    }

private:
    F first;
    S second;
};
```


b. **(4 points)** Write some code that instantiates and initializes a `Pair` object containing:

- the integer (17) as the first element and the string ("hello") as the second
- the doubles 2.1, and 5.4 as the first and second elements respectively

```
Pair<int, string> p1(17, "hello");  
Pair<double, double> p2(2.1, 5.4);
```

c. **(5 points)** The `Pair` template presented in class had a `swap` function, but this one does not. Why not?

Can't swap the first and second data members if their types are different