1 Short Answer

1. (*10 Points*) Write a function that takes in three parameters, the bill amount at a restaurant (before tax), the percentage of tax for the state, and the tip percentage that is applied to the pre-tax bill. Have the function print out the bill, tax amount, tip amount and total as in the example below. The following line in the main,

   ```
   printBill(125.67, 6, 17);
   ```

   Will produce the following output.

   ```
   Bill      125.67
   Tax         7.54
   Tip        21.36
   Total     154.57
   ```

   Solution:

```
void printBill(double amount, double taxper, double tipper) {
    double tax = amount * taxper / 100;
    double tip = amount * tipper / 100;
    double total = amount + tax + tip;

    cout << setprecision(2) << fixed;
    cout << "Bill    " << setw(8) << amount << endl;
    cout << "Tax     " << setw(8) << tax << endl;
    cout << "Tip     " << setw(8) << tip << endl;
    cout << "Total   " << setw(8) << total << endl;
    cout << endl;
}
```

2. (*10 Points*) Create a function called `sum` that will take in a single long parameter named `stop`, which will be your sentinel value. The function will continually ask the user for a number until the user types in this sentinel value. The function will return a long that is the sum of the user inputs excluding the sentinel value. For example, the lines in the main.

   ```
   long total = sum(-1);
   cout << total << endl;
   total = sum(12345);
   cout << total << endl;
   ```

   Will produce the following output. Your prompts must match those below.

   ```
   Input Number (-1 to stop): 2
   Input Number (-1 to stop): 3
   Input Number (-1 to stop): 1
   Input Number (-1 to stop): 7
   Input Number (-1 to stop): -1
   13
   Input Number (12345 to stop): -5
   Input Number (12345 to stop): 6
   Input Number (12345 to stop): -45
   Input Number (12345 to stop): 57
   Input Number (12345 to stop): 99
   Input Number (12345 to stop): 12345
   112
   ```

   Solution:

```
long sum(long stop) {
    long sum = 0;
    long input = 0;
    do {
        cout << "Input Number (" << stop << " to stop): ";
        cin >> input;
        if (input != stop)
            sum += input;

    } while (input != stop);

    return sum;
}
```

3. (*15 Points*) Write a function that sorts an integer array in ascending order using the selection sort.

   Solution:

```
void selectionSort(int array[], int size) {
    int minIndex, minValue;

    for (int start = 0; start < (size - 1); start++) {
        minIndex = start;
        minValue = array[start];
        for (int index = start + 1; index < size; index++) {
            if (array[index] < minValue) {
                minValue = array[index];
                minIndex = index;
            }
        }
        swap(array[minIndex], array[start]);
    }
}
```

4. (*15 Points*) Write a function that will take as parameters an array of integers, the size of the array, and a target value to search for. Have the function do a binary search for the target and return the index of the target in the array if found and -1 if not. You may assume that the array is already sorted in ascending order.

   Solution:

```
int binarySearch(const int A[], int size, int value) {
    int first = 0;
    int last = size - 1;
    int middle = 0;

    while (first <= last) {
        middle = (first + last) / 2;
        if (A[middle] == value) {
            return middle;
        } else if (A[middle] > value)
            last = middle - 1;
        else
            first = middle + 1;
    }
    return -1;
}
```

5. (*10 Points*) The D sequence is defined to be $D(1) = 1$, $D(2) = 1$, and

$$D(n) = D(D(n-1)) + D(n-1-D(n-2))$$

Write a recursive function D that takes in a single long parameter n and returns a long that is the value of $D(n)$.

Solution:

```
long D(long n) {
    if (n <= 0)
        return 0;

    if (n == 1 || n == 2)
        return 1;

    return D(D(n - 1)) + D(n - 1 - D(n - 2));
}
```

6. (*10 Points*) Write a recursive function that prints out the solution to the three-peg Towers of Hanoi puzzle given n disks as a parameter.

Solution:

```
void hanoi(int n, char initNeedle, char endNeedle, char tempNeedle) {
    if (n == 1)
        cout << "Move " << initNeedle << " to " << endNeedle << endl;
    else {
        hanoi(n - 1, initNeedle, tempNeedle, endNeedle);
        cout << "Move " << initNeedle << " to " << endNeedle << endl;
        hanoi(n - 1, tempNeedle, endNeedle, initNeedle);
    }
}
```

7. (*20 Points*) This exercise is to write a function that will manipulate an array using pointer notation only, no use of [ ] notation is allowed, and a function that will print out an array also using only pointer notation, specifics are below.

   (a) Write a function called `rotate` that will accept two integer pointers, start and end. The function will "rotate" the elements to the right (with wrapping) by one entry. The function will not return anything. So the element in the start position will be moved one to the right, as will the second and third and so on. The last element will be moved to the start. Note that the end pointer will be pointing to where the values stop, that is, after all the elements in the array. For example, if the array is

   `1 2 3 4 5 6 7 8 9 10`

   the call `rotate(A, A + size);` will alter the array to

   `10 1 2 3 4 5 6 7 8 9`

   and the call `rotate(A + 2, A + 7);` will alter the array to

   `1 2 7 3 4 5 6 8 9 10`

   (b) Write a function called `print` that will accept two integer pointers, start and end. The function will print the elements between start and end, again the end pointer will be pointing to where the values stop. For example, if the array is

   `1 2 3 4 5 6 7 8 9 10`

   the call `print(A, A + size);` will print the entire array,

   `1 2 3 4 5 6 7 8 9 10`

   and the call `print(A + 3, A + 8);` will print,

   `4 5 6 7 8`

Solution:

```
void rotate(int *start, int *end) {
    int last = *(end - 1);
    int *p;
    p = end - 2;

    while (p >= start) {
        *(p + 1) = *p;
        p--;
    }
    p++;
    *p = last;
}

void print(int *start, int *end) {
    int *p = start;
    while (p < end)
        cout << *p++ << " ";
    cout << endl;
}
```

2 Programming

1. (*15 Points*) Write a function called `minmax` that takes as its only parameter a vector of doubles. The function is to find the minimum and maximum of the values in the vector and return a vector with two entries, the first is the minimum and the second is the maximum obtained from the input vector. Then write a main that will create an vector 1,000 random doubles between 0 and 1. The main will then call the `minmax` function to get the minimum and maximum. Finally, from the result of `minmax`, have the main print out the minimum and maximum to the console on a single line.

   Solution:

```
#include <iostream>
#include <vector>

using namespace std;

vector<double> minmax(vector<double> A) {
    double min = A[0];
    double max = A[0];

    for (int i = 0; i < A.size(); i++) {
        if (A[i] < min)
            min = A[i];
        if (A[i] > max)
            max = A[i];
    }
    vector<double> ret;
    ret.push_back(min);
    ret.push_back(max);
    return ret;
}

int main() {
    srand(time(0));
    vector<double> D;

    int sz = 1000;

    for (int i = 0; i < sz; i++)
        D.push_back(static_cast<double>(rand()) / RAND_MAX);

    vector<double> MM = minmax(D);
    cout << "Min = " << MM[0] << "    Max = " << MM[1] << endl;

    return 0;
}
```

2. (*20 Points*) Write a function called `copyArray` that takes as parameters an integer array and the size of the array. The function is to create a copy of the array and return a pointer to the copy. Then write a main that will create an array A of 1,000,000 random integers between 1 and 1000, stored in the heap. The main will then call the `copyArray` function to create a copy of the array that it will store in the array B. Finally have the main print out the contents of B to the console on a single line with a space between the entries. The program should have no memory leaks and not read or write outside allocated memory.

   Solution:

```cpp
#include <iostream>

using namespace std;

int* copyArray(int *A, int size) {
    int *B = new int[size];

    for (int i = 0; i < size; i++)
        B[i] = A[i];

    return B;
}

int main() {
    srand(time(0));
    int sz = 1000000;
    int *A = new int[sz];

    for (int i = 0; i < sz; i++)
        A[i] = rand() % 1000 + 1;

    int *B = copyArray(A, sz);

    for (int i = 0; i < sz; i++)
        cout << B[i] << " ";
    cout << endl;

    delete[] A;
    delete[] B;

    return 0;
}
```

3. (*20 Points*) This program will read in student data (names and GPA) from a file, find the maximum GPA, and then print out all students with that GPA. Data storage will be using parallel vectors. The program will do this in three functions and a main, as described below.

   (a) A function `loadData` will open a text file named `students.txt` for input and load the students' names into a vector of strings and the GPA into a vector of doubles. These vectors are to be reference parameters so that the calling function can retrieve the information from the file. The names and GPA are stored as one student per line, name first then GPA separated by a tab character, as in the example below. You will not know the length of the file.

   ```
   Jane Doe 3.5
   John Doe 3.1
   John Smith   2.5
   Jack Frost   3.5
   ```

   (b) A function `maxGPA` will bring in as a parameter (by reference but uneditable) the vector of GPAs and return the maximum value.

   (c) A function `display` will take in as parameters the two vectors of names and GPAs and a single (target) gpa value. The function will print out all student's names that have the target gpa parameter. The output from the file above using 3.5 as the target value would be,

   ```
   Jane Doe
   Jack Frost
   ```

   (d) The main will consist of 6 lines, two lines to declare the two vectors, a call to `loadData`, a call to `maxGPA`, a call to `display`, and finally a `return 0;`.

Solution:

```
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

void loadData(vector<string> &names, vector<double> &gpa) {
    fstream dataFile;
    dataFile.open("students.txt", ios::in);

    if (!dataFile) {
        cerr << "Error opening file.\n";
        return;
    }

    string line;
    while (getline(dataFile, line)) {
        int pos = line.find('\t');
        names.push_back(line.substr(0, pos));
        gpa.push_back(stod(line.substr(pos)));
    }
}

double maxGPA(const vector<double> &gpa) {
    double max = gpa[0];

    for (int i = 0; i < gpa.size(); i++)
        if (gpa[i] > max)
            max = gpa[i];

    return max;
}

void display(const vector<string> &names, const vector<double> &gpa,
        double val) {
    for (int i = 0; i < gpa.size(); i++)
        if (gpa[i] == val)
            cout << names[i] << endl;
}

int main() {
    vector<string> names;
    vector<double> gpa;

    loadData(names, gpa);
    double maxgpa = maxGPA(gpa);
    display(names, gpa, maxgpa);

    return 0;
}
```

4. (*25 Points*) This exercise is to create a rectangle class structure that defines the geometric shape of a rectangle. Write both the Rectangle.h and Rectangle.cpp files for this class, no inline code may be used.

   The class is to store the width and height of the rectangle, have a default constructor that setts the width and height both to 1, a non-default constructor that brings in the width and height as parameters, accessor functions to return the width and height, a set function that resets the width and height, a function that returns the area, a function that returns the perimeter, and a function that determines if the rectangle is a square. Make sure that the width and height are never negative. The following program will produce the following output.

```cpp
#include <iostream>
#include "Rectangle.h"

using namespace std;

int main() {
    Rectangle r1;
    Rectangle r2(2.7, 3.5);

    cout << r1.area() << endl;
    cout << r1.perimeter() << endl;
    cout << r1.isSquare() << endl;
    cout << endl;

    cout << r2.getWidth() << endl;
    cout << r2.getHeight() << endl;
    cout << r2.area() << endl;
    cout << r2.perimeter() << endl;
    cout << r2.isSquare() << endl;
    cout << endl;

    r2.set(5.675, 17.123);
    cout << r2.getWidth() << endl;
    cout << r2.getHeight() << endl;
    cout << r2.area() << endl;
    cout << r2.perimeter() << endl;
    cout << r2.isSquare() << endl;

    return 0;
}
```

Output

```
1
4
1

2.7
3.5
9.45
12.4
0

5.675
17.123
97.173
45.596
0
```

Solution:

```cpp
#ifndef RECTANGLE_H_
#define RECTANGLE_H_

class Rectangle {
private:
    double width;
    double height;

public:
    Rectangle(double w = 1, double h = 1);

    double getWidth();
    double getHeight();
    void set(double, double);

    double area();
    double perimeter();
    bool isSquare();
};

#endif
```

```cpp
#include "Rectangle.h"

Rectangle::Rectangle(double w, double h) {
    if (w < 0)
        w = 0;
    if (h < 0)
        h = 0;

    width = w;
    height = h;
}

double Rectangle::getWidth() {
    return width;
}

double Rectangle::getHeight() {
    return height;
}

void Rectangle::set(double w, double h) {
    if (w < 0)
        w = 0;
    if (h < 0)
        h = 0;

    width = w;
    height = h;
}

double Rectangle::area() {
    return width * height;
}

double Rectangle::perimeter() {
    return 2 * (width + height);
}

bool Rectangle::isSquare() {
    return width == height;
}
```

5. (*30 Points*) This exercise is to create a safe array class structure that defines an integer array but in addition handles resizing and memory management as well as not allowing reading or writing outside the allocated memory space. Write both the SafeArray.h and SafeArray.cpp files for this class, no inline code may be used.

   The class is to store the size of the array and a pointer to the array contents.

   - A default constructor that has the size of the array as 0 with no allocated memory.
   - A non-default constructor that takes an initial size of the array, creates it and populates the array with zeros.
   - A copy constructor.
   - A destructor.
   - A function that returns the current size of the array.
   - A function that resizes the array to a specified size. If the given size is less than or equal to 0 then the function should remove the array from memory and set the size to 0. If the new size is larger than the old the new array will have the contents of the old one and the extra entries are to be set to 0. If the new array is smaller then the old the new one will have the values of the old but truncated to the new array size.
   - A clear function that sets the size to 0 and removes the array.
   - Overloaded assignment operator.
   - Overloaded access operator, that is the bracket operator [ ]. This should return or update the entry in the given index and if the index is out of the array bounds it should report an index out of bounds error and halt the program.
   - Overloaded stream out operator that prints the array on a single line with a space between each element.

   The following program will produce the following output.

```cpp
#include <iostream>
#include "SafeArray.h"

using namespace std;

int main() {
    SafeArray A;
    A.resize(10);
    cout << A << endl;
    for (int i = 0; i < A.get_size(); i++)
        A[i] = i+1;
    cout << A << endl;

    SafeArray B = A;
    cout << B << endl;
    B[3] = -5;
    B[5] = 25;
    B[8] = 123;
    cout << A << endl;
    cout << B << endl;

    SafeArray C(8);
    cout << C << endl;
    C = B;
    cout << C << endl;
    C[1] = 21;
    C[7] = 42;
    cout << C << endl;
    cout << B << endl;
    B.resize(5);
    cout << B << endl;
    B.resize(7);
    cout << B << endl;

    return 0;
}
```

Output

```
0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 -5 5 25 7 8 123 10
0 0 0 0 0 0 0 0
1 2 3 -5 5 25 7 8 123 10
1 21 3 -5 5 25 7 42 123 10
1 2 3 -5 5 25 7 8 123 10
1 2 3 -5 5
1 2 3 -5 5 0 0
```

Solution:

```cpp
#ifndef SAFEARRAY_H_
#define SAFEARRAY_H_

#include <iostream>
using namespace std;

class SafeArray {
private:
    int size = 0;
    int *list = nullptr;

public:
    SafeArray();
    SafeArray(int);
    SafeArray(const SafeArray&);
    ~SafeArray();

    int get_size();
    void resize(int);

    void clear();
    const SafeArray operator=(const SafeArray&);

    int& operator[](const int&);
    friend ostream& operator <<(ostream&, const SafeArray&);
};

#endif
```

```cpp
#include "SafeArray.h"

SafeArray::SafeArray() {
    size = 0;
    list = nullptr;
}

SafeArray::SafeArray(int sz) {
    resize(sz);
}

SafeArray::~SafeArray() {
    delete[] list;
}

SafeArray::SafeArray(const SafeArray &right) {
    size = right.size;

    list = new int[size];
    for (int i = 0; i < size; i++)
        list[i] = right.list[i];
}

int SafeArray::get_size() {
    return size;
}

void SafeArray::resize(int sz) {
    if (sz <= 0) {
        clear();
        return;
    }

    int copysize = (size > sz) ? sz : size;
    int *newarray = new int[sz];

    for (int i = 0; i < sz; i++)
        newarray[i] = 0;

    for (int i = 0; i < copysize; i++)
        newarray[i] = list[i];

    delete[] list;
    list = newarray;
    size = sz;
}

void SafeArray::clear() {
    delete[] list;
    size = 0;
    list = nullptr;
}

const SafeArray SafeArray::operator=(const SafeArray &right) {
    delete[] list;
    size = right.size;

    list = new int[size];
    for (int i = 0; i < size; i++)
        list[i] = right.list[i];

    return *this;
}

int& SafeArray::operator[](const int &i) {
    if (i < 0 || i >= size) {
        cerr << "Index out of bounds." << endl;
        exit(1);
    } else
        return list[i];
}

ostream& operator <<(ostream &strm, const SafeArray &obj) {
    for (int i = 0; i < obj.size; i++)
        strm << obj.list[i] << " ";

    return strm;
}
```