

1. (15 points) Pointers & Dynamic Arrays: Write a function that takes as parameters the pointers of two sorted integer arrays and the two array sizes, and returns the pointer of a new integer array that is the sorted merging of the two arrays. For example, if the two parameter arrays are [3, 7, 10, 16, 22] and [1, 2, 5, 10, 12, 17] the resulting array is [1, 2, 3, 5, 7, 10, 10, 12, 16, 17, 22].

Solution:

```
int *merge(int A[], int B[], int LenA, int LenB) {
    int *ret = new int[LenA + LenB];

    int aptr = 0;
    int bptr = 0;
    int i = 0;

    while (aptr < LenA && bptr < LenB)
        if (A[aptr] < B[bptr])
            ret[i++] = A[aptr++];
        else
            ret[i++] = B[bptr++];

    while (aptr < LenA)
        ret[i++] = A[aptr++];

    while (bptr < LenB)
        ret[i++] = B[bptr++];

    return ret;
}
```

2. (15 points) Basic Classes: Write the specification and implementation (no inline code) for a Rectangle class that stores a height and width (decimal values possible), constructor that takes the height and width as well as a default constructor that sets height and width to 1 each. The non-default constructor should error check the input so that any negative input is changed to 0. Accessors and mutators, mutators should do the same error checking. A member function for the area and a member function for the perimeter.

Solution:

```
1 #ifndef RECTANGLE_H_
2 #define RECTANGLE_H_
3
4 #include <iostream>
5
6 using namespace std;
7
8 class Rectangle {
9     private:
10         double height;
11         double width;
12
13     public:
14         Rectangle(double h = 1, double w = 1);
15         ~Rectangle();
16
17         void setHeight(double);
18         double getHeight();
19         void setWidth(double);
20         double getWidth();
21         double Area();
22         double Perimeter();
23 };
24
25 #endif
26
27 #include <iostream>
28
29 #include "Rectangle.h"
30
31 using namespace std;
32
33 Rectangle::Rectangle(double h, double w) {
34     if (h < 0)
35         h = 0;
36     height = h;
37
38     if (w < 0)
39         w = 0;
40     width = w;
41 }
42
43 Rectangle::~Rectangle() {}
44
45 void Rectangle::setHeight(double h) {
46     if (h < 0)
47         h = 0;
48     height = h;
49 }
50
51 double Rectangle::getHeight() { return height; }
52
53 void Rectangle::setWidth(double w) {
54     if (w < 0)
55         w = 0;
56     width = w;
57 }
58
59 double Rectangle::getWidth() { return width; }
60
61 double Rectangle::Area() { return height * width; }
62
63 double Rectangle::Perimeter() { return 2 * (height
64     + width); }
```

3. (20 points) Inheritance & Polymorphism: In this exercise we construct class structures, automobile, car, and truck. automobile is to be the base class and the other two will be derived classes off of this. For the mutators listed below you do not need to do any error checking on the values.

- automobile is to store the name of the auto that can be set by its constructor but defaults to “auto”. It also stores the mileage of the vehicle. It also has a destructor, member functions to access and change the name and mileage, and a virtual function toString that will print out the name and mileage.
- car inherits off of automobile, it further stores the number of doors the car has. It has a constructor allowing input of a name and a default that uses “car” as the name. It also has a destructor and accessor and mutator for the doors. Its toString function will, in addition to printing out the name and mileage, will print out the number of doors.
- truck inherits off of automobile, it further stores the length of the truck bed. It has a constructor allowing input of a name and a default that uses “truck” as the name. It also has a destructor and accessor and mutator for the bed length. Its toString function will, in addition to printing out the name and mileage, will print out the length of the bed.

With these classes and functions in place the following program will produce the output below.

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

#include "auto.h"
#include "car.h"
#include "truck.h"

using namespace std;

int main() {
    vector<automobile *> autos;
    car *carptr = new car;
    carptr->setDoors(4);
    carptr->setMiles(80000);
    autos.push_back(carptr);

    truck *truckptr = new truck("Sport Pickup");
    truckptr->setBedLength(6);
    truckptr->setMiles(35000);
    autos.push_back(truckptr);

    truckptr = new truck;
    truckptr->setBedLength(8);
    truckptr->setMiles(125300);
    autos.push_back(truckptr);

    carptr = new car("Corvette");
    carptr->setDoors(2);
    carptr->setMiles(15200);
    autos.push_back(carptr);

    for (auto v : autos)
        cout << v->toString() << endl;

    return 0;
}
```

Output:

```
car Miles: 80000 Doors: 4
Sport Pickup Miles: 35000 Bed Length: 6
truck Miles: 125300 Bed Length: 8
Corvette Miles: 15200 Doors: 2
```

Solution:

```

1  #ifndef AUTO_H_
2  #define AUTO_H_
3
4  #include <iostream>
5
6  using namespace std;
7
8  class automobile {
9      protected:
10         string name;
11         int miles;
12
13     public:
14         automobile(string n = "auto");
15         virtual ~automobile();
16         string getName();
17         void setName(string n);
18         int getMiles();
19         void setMiles(int m);
20         virtual string toString();
21 };
22
23 #endif

```

```

1  #include <iostream>
2
3  #include "auto.h"
4
5  automobile::automobile(string n) { name = n; }
6  automobile::~~automobile() {}
7  string automobile::getName() { return name; }
8  void automobile::setName(string n) { name = n; }
9  int automobile::getMiles() { return miles; }
10 void automobile::setMiles(int m) { miles = m; }
11 string automobile::toString() { return name + " Miles: " + to_string(miles); }

```

```

1  #ifndef CAR_H_
2  #define CAR_H_
3
4  #include <iostream>
5
6  #include "auto.h"
7
8  using namespace std;
9
10 class car : public automobile {
11     protected:
12         int doors;
13
14     public:
15         car(string n = "car");
16         ~car();
17         int getDoors();
18         void setDoors(int m);
19         string toString();
20 };
21
22 #endif

```

```

1  #include <iostream>
2
3  #include "car.h"
4
5  using namespace std;
6
7  car::car(string n) : automobile(n) {}
8  car::~~car() {}
9  int car::getDoors() { return doors; }
10 void car::setDoors(int m) { doors = m; }

```

```
11 string car::toString() {  
12     return automobile::toString() + "  Doors: " + to_string(doors);  
13 }
```

```
1 #ifndef TRUCK_H_  
2 #define TRUCK_H_  
3  
4 #include <iostream>  
5  
6 #include "auto.h"  
7  
8 using namespace std;  
9  
10 class truck : public automobile {  
11     protected:  
12         int bedlength;  
13  
14     public:  
15         truck(string n = "truck");  
16         ~truck();  
17         int getBedLength();  
18         void setBedLength(int m);  
19         string toString();  
20 };  
21  
22 #endif  
  
1 #include <iostream>  
2  
3 #include "truck.h"  
4  
5 truck::truck(string n) : automobile(n) {}  
6 truck::~~truck() {}  
7 int truck::getBedLength() { return bedlength; }  
8 void truck::setBedLength(int m) { bedlength = m; }  
9 string truck::toString() {  
10     return automobile::toString() + "  Bed Length: " + to_string(bedlength);  
11 }
```

4. (20 points) Operator Overloading: Say we have a class structure called `Point` that represents a point (x, y, z) in three dimensions. The data, x , y , and z are stored in an array of three doubles, specifically, `double P[3];` is its declaration. Write the following overloads for the `+`, `-` and `*` operators. Give both the specification and implementation of each with no inline code. You may assume that the `Point` class has a non-default constructor that takes three double parameters, specifically, `Point p(x, y, z);` is a valid creation of the point `p`.

- The addition of two points is another point and is done by adding the respective x , y , and z values. So in mathematical notation, $(x_1, y_1, z_1) + (x_2, y_2, z_2) = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$.
- The subtraction of two points is another point and is done by subtracting the respective x , y , and z values. So in mathematical notation, $(x_1, y_1, z_1) - (x_2, y_2, z_2) = (x_1 - x_2, y_1 - y_2, z_1 - z_2)$.
- Multiplying a number and a point (called scalar multiplication) is just multiplying each entry by that number. So in mathematical notation, $a \cdot (x_1, y_1, z_1) = (a \cdot x_1, a \cdot y_1, a \cdot z_1)$. This can also be done on either side of the point, that is, $(x_1, y_1, z_1) \cdot a = (a \cdot x_1, a \cdot y_1, a \cdot z_1)$.

Solution:

```

1 Point operator+(const Point&);
2 Point operator-(const Point&);
3 Point operator*(double);
4 friend Point operator*(double, const Point&);

1 Point Point::operator+(const Point &pt) {
2     Point p(P[0] + pt.P[0], P[1] + pt.P[1], P[2] + pt.P[2]);
3     return p;
4 }
5
6 Point Point::operator-(const Point &pt) {
7     Point p(P[0] - pt.P[0], P[1] - pt.P[1], P[2] - pt.P[2]);
8     return p;
9 }
10
11 Point Point::operator*(double a) {
12     Point p(a * P[0], a * P[1], a * P[2]);
13     return p;
14 }
15
16 Point operator*(double a, const Point &rhs) {
17     Point p = rhs;
18     return p * a;
19 }

```

5. (30 points) Linked Lists:

- (a) Write an `appendNode` function for the templated linked list. The function will take in a single parameter of the templated type, create the node of `ListNode` type and attach it to the end of the list. The start of the list is a pointer called `head`. This implementation of the linked list does not have a tail pointer.

Solution:

```

1  template<class T> void LinkedList<T>::appendNode(T newValue) {
2      ListNode<T> *newNode;
3      ListNode<T> *nodePtr;
4
5      newNode = new ListNode<T>(newValue);
6
7      if (!head)
8          head = newNode;
9      else {
10         nodePtr = head;
11
12         while (nodePtr->next)
13             nodePtr = nodePtr->next;
14
15         nodePtr->next = newNode;
16     }
17 }
```

- (b) Write a copy constructor for the templated linked list class. Assume the class name is `LinkedList`.

Solution:

```

1  template <class T> LinkedList<T>::LinkedList(const LinkedList &obj) {
2      head = nullptr;
3      ListNode<T> *ptr = obj.head;
4
5      while (ptr){
6          appendNode(ptr->value);
7          ptr = ptr->next;
8      }
9  }
```

- (c) Write a `deleteNode` function for the templated linked list. The function will take in a single parameter of the templated type and remove the first occurrence of that value from the list. If the value is not in the list then the list is unaltered. The start of the list is a pointer called `head`. This implementation of the linked list does not have a tail pointer.

Solution:

```

1  template<class T>
2  void LinkedList<T>::deleteNode(T searchValue) {
3      ListNode<T> *nodePtr;
4      ListNode<T> *previousNode;
5
6      if (!head)
7          return;
8
9      if (head->value == searchValue) {
10         nodePtr = head->next;
11         delete head;
12         head = nodePtr;
13     } else {
14         nodePtr = head;
15         while (nodePtr != nullptr && nodePtr->value != searchValue) {
16             previousNode = nodePtr;
17             nodePtr = nodePtr->next;
18         }
19         if (nodePtr) {
20             previousNode->next = nodePtr->next;
21             delete nodePtr;
22         }
23     }
24 }
```

6. (10 points) Stacks, Queues, and STL: Write a function called `balanced` that takes in a string (which represents a mathematical expression) and uses an STL stack to determine if the parentheses are balanced. The function will return `true` if the parentheses are balanced and `false` otherwise. For example, an input of `(x+4/(x-1))*(y+7)/(z*(x + 4*y))` would return `true` and an input of `x+4/(x-1))` would return `false`.

Solution:

```
1 bool balanced(string exp) {
2     stack<char> perst;
3
4     for (unsigned long i = 0; i < exp.length(); i++) {
5         char ch = exp[i];
6         if (ch == '(')
7             perst.push(ch);
8         else if (ch == ')') {
9             if (perst.empty())
10                return false;
11             char top = perst.top();
12             if (top != '(')
13                return false;
14             perst.pop();
15         }
16     }
17
18     if (perst.empty())
19         return true;
20     else
21         return false;
22 }
```

7. (20 points) Recursion:

- (a) 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:

```

1 long D(long n) {
2     if (n <= 0)
3         return 0;
4
5     if (n == 1 || n == 2)
6         return 1;
7
8     return D(D(n - 1)) + D(n - 1 - D(n - 2));
9 }
```

- (b) Write a recursive function that calculates n choose k ,

$$\binom{n}{k}$$

The mathematical recursive definition is,

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

If n or k is 0 then the value is 1, if $n = k$ the value is 1, and we will define anything outside these ranges the value is 0. That is, if n or k are negative or if $k > n$ we will simply have the program return 0.

Solution:

```

1 long comb(long n, long k) {
2     if (k == 0)
3         return 1;
4     else if (k == n)
5         return 1;
6     else if (k > n)
7         return 0;
8     else
9         return comb(n - 1, k - 1) + comb(n - 1, k);
10 }
```

- (c) In a standard templated linked list class with external templated `ListNode` class we will rewrite the display list function to be recursive. The private (or protected) section contains the specification,

```
void displayListRec(ListNode<T> *) const;
```

The public section contains the specification,

```
void displayList() const { displayListRec(head); }
```

Write the `displayListRec` function recursively, that is, no loops allowed.

Solution:

```

1 template <class T> void LinkedList<T>::displayListRec(ListNode<T> *ptr) const
2 {
3     if (ptr) {
4         cout << ptr->value << " ";
5         displayListRec(ptr->next);
6     }
7 }
```


8. (25 points) Binary Search Trees:

- (a) Write the recursive insert function for the templated binary search tree class. This function is to take a pointer to a tree node and a pointer to a new node and insert the new node in the correct position in the BST. That is, if we were to run the following code, the insert function will put this new node in the correct position in the tree.

```
1 TreeNode *newNode = new TreeNode;
2 newNode->value = item;
3 newNode->left = newNode->right = nullptr;
4 insert(root, newNode);
```

Solution:

```
1 template<class T>
2 void BinaryTree<T>::insert(TreeNode *&nodePtr, TreeNode *&newNode) {
3     if (nodePtr == nullptr)
4         nodePtr = newNode;           // Insert the node.
5     else if (newNode->value < nodePtr->value)
6         insert(nodePtr->left, newNode); // Search the left branch
7     else
8         insert(nodePtr->right, newNode); // Search the right branch
9 }
```

- (b) Write an in-order traversal function for the templated binary search tree class that displays the node value to the console screen.

Solution:

```
1 template<class T>
2 void BinaryTree<T>::displayInOrder(TreeNode *nodePtr) const {
3     if (nodePtr) {
4         displayInOrder(nodePtr->left);
5         cout << nodePtr->value << endl;
6         displayInOrder(nodePtr->right);
7     }
8 }
```

(c) Write the three templated functions,

```
1 void deleteNode(T, TreeNode *&);
2 void makeDeletion(TreeNode *&);
3 void remove(T);
```

That together will delete an item from a binary search tree.

Solution:

```
1 template <class T> void BinaryTree<T>::remove(T item) {
2     deleteNode(item, root);
3 }
4
5 template <class T> void BinaryTree<T>::deleteNode(T item, TreeNode *&nodePtr)
6 {
7     if (!nodePtr)
8         return;
9     if (item < nodePtr->value)
10        deleteNode(item, nodePtr->left);
11    else if (item > nodePtr->value)
12        deleteNode(item, nodePtr->right);
13    else
14        makeDeletion(nodePtr);
15 }
16
17 template <class T> void BinaryTree<T>::makeDeletion(TreeNode *&nodePtr) {
18     TreeNode *tempNodePtr = nullptr;
19
20     if (nodePtr == nullptr)
21         cout << "Cannot delete empty node.\n";
22     else if (nodePtr->right == nullptr) {
23         tempNodePtr = nodePtr;
24         nodePtr = nodePtr->left;
25         delete tempNodePtr;
26     } else if (nodePtr->left == nullptr) {
27         tempNodePtr = nodePtr;
28         nodePtr = nodePtr->right;
29         delete tempNodePtr;
30     }
31     else {
32         // Move one node the right.
33         tempNodePtr = nodePtr->right;
34         while (tempNodePtr->left)
35             tempNodePtr = tempNodePtr->left;
36         tempNodePtr->left = nodePtr->left;
37         tempNodePtr = nodePtr;
38         nodePtr = nodePtr->right;
39         delete tempNodePtr;
40     }
41 }
```

9. (20 points) Complexity:

- (a) State the precise mathematical definitions of Big- O , Big- Ω , and Big- Θ . Also give the common meaning of each, specifically, what bound does it indicate?

Solution:

- A function $g(n)$ is $O(f(n))$ if there exist a constants $c > 0$ and n_0 such that, for every $n > n_0$, $|g(n)| \leq cf(n)$. This is an Upper Bound on the complexity.
- A function $g(n)$ is $\Omega(f(n))$ if there exist a constants $c > 0$ and n_0 such that, for every $n > n_0$, $|g(n)| \geq cf(n)$. This is a Lower Bound on the complexity.
- A function $g(n)$ is $\Theta(f(n))$ if there exist a constants $c_1 > 0$, $c_2 > 0$, and n_0 such that, for every $n > n_0$, $c_1f(n) \leq |g(n)| \leq c_2f(n)$. This is a Tight Bound on the complexity.

- (b) Fill out the time complexity table below.

Solution:

Algorithm	Best	Average	Worst
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$
Merge Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$
Tree Sort with BST	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$
Linear Search on Array	$\Omega(1)$	$\Theta(n)$	$O(n)$
Binary Search on Sorted Array	$\Omega(1)$	$\Theta(\log(n))$	$O(\log(n))$

- (c) Prove that $T(n) = 2^{\sqrt{\lg n}}$ is $O(\sqrt[5]{n})$.

Solution: We need to show that there are constants $c > 0$ and n_0 such that, for every $n > n_0$, $|g(n)| \leq cf(n)$. That is, $2^{\sqrt{\lg n}} \leq c\sqrt[5]{n}$.

$$\begin{aligned}
 2^{\sqrt{\lg n}} &\leq c\sqrt[5]{n} \\
 \lg\left(2^{\sqrt{\lg n}}\right) &\leq \lg\left(cn^{1/5}\right) \\
 \sqrt{\lg n} \lg(2) &\leq \frac{1}{5} \lg(n) + \lg(c) && \text{Let } c = 1. \\
 \sqrt{\lg n} &\leq \frac{1}{5} \lg(n) \\
 5 &\leq \sqrt{\lg(n)} \\
 25 &\leq \lg(n) \\
 n &\geq 2^{25}
 \end{aligned}$$

So let $c = 1$ and $n_0 = 2^{25} = 33,554,432$.

10. (15 points) Sets and Multisets: Write a program that will ask the user to input the number of dice to roll, the number of sides of the dice (assume all die are the same), and the number of rolls. Have the program roll all the dice that many times and store each result in a multi set. After all the data is stored, have the program print out the counts of each roll. Make sure that each roll total appears only one time. Example runs are below.

```
Input number of dice to roll: 2
Input number of sides for each die: 6
Input number of rolls of dice: 10
Results
4 1
6 3
7 3
8 1
9 1
10 1
```

```
Input number of dice to roll: 2
Input number of sides for each die: 4
Input number of rolls of dice: 1000000
Results
2 61954
3 124703
4 187578
5 250120
6 187825
7 125267
8 62553
```

Solution:

```
1  int main() {
2      int numDice = 0;
3      int numRolls = 0;
4      int sides = 6;
5      multiset<int> Rolls;
6      srand(time(0));
7
8      cout << "Input number of dice to roll: ";
9      cin >> numDice;
10     cout << "Input number of sides for each die: ";
11     cin >> sides;
12     cout << "Input number of rolls of dice: ";
13     cin >> numRolls;
14
15     for (int i = 0; i < numRolls; i++) {
16         int totalRoll = 0;
17         for (int j = 0; j < numDice; j++) {
18             totalRoll += rand() % sides + 1;
19         }
20         Rolls.insert(totalRoll);
21     }
22
23     cout << "Results" << endl;
24
25     set<int> KeySet;
26     KeySet.insert(Rolls.begin(), Rolls.end());
27     for (auto it = KeySet.begin(); it != KeySet.end(); it++) {
28         cout << *it << " " << Rolls.count(*it) << endl;
29     }
30
31     return 0;
32 }
```

11. (10 points) Sorts: Write the templated Quick Sort functions to sort an array.

Solution:

```
1  template<class T>
2  void quickSort(T A[], int left, int right) {
3      int i = left;
4      int j = right;
5      int mid = (left + right) / 2;
6
7      T pivot = A[mid];
8
9      while (i <= j) {
10         while (A[i] < pivot)
11             i++;
12
13         while (A[j] > pivot)
14             j--;
15
16         if (i <= j) {
17             T tmp = A[i];
18             A[i] = A[j];
19             A[j] = tmp;
20             i++;
21             j--;
22         }
23     }
24
25     if (left < j)
26         quickSort(A, left, j);
27
28     if (i < right)
29         quickSort(A, i, right);
30 }
31
32 template<class T>
33 void quickSort(T A[], int size) {
34     quickSort(A, 0, size - 1);
35 }
```

12. (10 points) Short Answer:

- (a) Assuming that ptr is a pointer to a double, what memory address is (ptr + 10)?

Solution: 10*sizeof(double) bytes larger than ptr, probably 80 bytes more than ptr.

- (b) Describe the difference between an instance member variable and a static member variable.

Solution: An instance member variable is different for all instances of a class, that is, are stored in different locations in memory. A static member variable is shared between all instances of the class.

- (c) In what circumstances is the copy constructor called?

Solution: When construction is done and the new object is initialized with another object. In addition, when a function parameter is a class structure and is sent by value.

- When an object of the class is returned by value: return obj;
- When an object of the class is passed to a function by value as an argument: fct(MyObj obj).
- When an object is constructed based on another object of the same class: MyObj obj = obj2.
- When compiler generates a temporary object.

- (d) What is the difference between a protected class member and a private class member?

Solution: protected members can be seen by derived classes whereas private members cannot.

- (e) Which destructor is called first, that of the derived class or the base class? What must be done to ensure that the entire chain of destructors in an inheritance chain are called?

Solution: The derived class constructor is called first. You must make the destructor virtual.

- (f) What is an iterator? Write the definition of one for a vector of longs.

Solution: An iterator is an object that behaves like a pointer. It is used to access the individual data elements in a container.

```
1 vector<long>::iterator iter;
```

- (g) Write a complete make file for compiling the following code project. The compilation must do the minimal amount of recompiling dependent on the changes in the files. Also, the recompilation should track changes in all dependent files. The project to be compiled has five files in it, main.cpp, IntBST.h, IntBST.cpp, NumberList.h and NumberList.cpp. The main.cpp and NumberList.cpp both include NumberList.h and the main.cpp and IntBST.cpp both include IntBST.h. There is no templating in these class structures so the specification and implementation of the NumberList class is divided between the two files. Use an arrow “→” to denote a tab character.

Solution:

```

PROG = prog
CC = g++
CPPFLAGS = -g -Wall -std=c++11
OBJS = main.o NumberList.o IntBST.o

$(PROG) : $(OBJS)
    $(CC) -o $(PROG) $(OBJS)

main.o : main.cpp NumberList.h IntBST.h
    $(CC) $(CPPFLAGS) -c main.cpp

NumberList.o : NumberList.h NumberList.cpp
    $(CC) $(CPPFLAGS) -c NumberList.cpp

IntBST.o : IntBST.h IntBST.cpp
    $(CC) $(CPPFLAGS) -c IntBST.cpp

clean:
    rm -f core $(PROG) $(OBJS)

rebuild:
    make clean
    make
```