**#include <iostream>**

**#include <fstream>**

**#include <iomanip>**

**struct** Node {

**friend** **class** **List**;

**private**:

**double** first;

**double** second;

Node\* nextNode;

};

**class** **List** {

**public**:

List();

**void** add(**double** f, **double** s);

**void** sort(**int** data);

**void** next();

**void** toStart() {current = first;}

**int** length() {**return** size;}

**double**\* currentValue();

**private**:

**void** swap(Node\* llNode, Node\* lNode, Node\* rNode);

**int** size;

Node\* first;

Node\* current;

};

List::List() {

size = 0;

Node\* node = **new** Node;

node->first = 0;

node->second = 0;

node->nextNode = NULL;

first = node;

current = node;

}

**void** List::next() {

//if(current->nextNode == NULL)

//current = first;

//else

current = current->nextNode;

}

**void** List::add(**double** f, **double** s) {

**if**(size == 0)

{

first->first = f;

first->second = s;

}

**else**

{

Node\* node = **new** Node;

node->first = f;

node->second = s;

node->nextNode = first;

first = node;

current = node;

}

size++;

}

**double**\* List::currentValue() {

**double**\* value = **new** **double**[2];

value[0] = current->first;

value[1] = current->second;

**return** value;

}

**void** List::sort(**int** data) {

**double** lowerValue, higherValue;

**bool** swapped = **true**;

**while** (swapped)

{

Node\* lowerNode = first;

Node\* llowerNode = NULL;

Node\* higherNode = first->nextNode;

swapped = **false**;

**while**(higherNode != NULL)

{

**if**(data == 0)

{

higherValue = higherNode->first;

lowerValue = lowerNode->first;

}

**else** **if** (data == 1)

{

higherValue = higherNode->second;

lowerValue = lowerNode->second;

}

**else**

{

std::cout << "Error: invalid agrument. Must be 0 or 1.\n";

**return**;

}

**if**(lowerValue > higherValue)

{

swap(llowerNode, lowerNode, higherNode);

llowerNode = higherNode;

swapped = **true**;

higherNode = lowerNode->nextNode;

}

**else**

{

llowerNode = lowerNode;

higherNode = higherNode->nextNode;

lowerNode = lowerNode->nextNode;

}

}

}

toStart();

//delete higherNode;

//delete lowerNode;

}

**void** List::swap(Node\* llNode, Node\* lNode, Node\* hNode) {

Node\* nexthNode = hNode->nextNode;

**if**(nexthNode == NULL)

lNode->nextNode = NULL;

**else**

lNode->nextNode = nexthNode;

**if**(llNode != NULL)

llNode->nextNode = hNode;

hNode->nextNode = lNode;

**if**(first == lNode)

first = hNode;

//delete nexthNode;

}

**int** main() {

std::ifstream in;

in.open("input.txt");

List mylist;

**double** temp1, temp2;

in >> temp1 >> temp2;

**while** (in.good())

{

mylist.add(temp1, temp2);

in >> temp1 >> temp2;

}

in.close();

**double**\* value;

std::cout << "myList:\n";

**for**(**int** i = 1; i <= mylist.length(); i++)

{

value = mylist.currentValue();

mylist.next();

std::cout << std::setw(6) << value[0] << std::setw(6) << value[1] << std::endl;

}

mylist.sort(0);

std::cout << "List values sorted by first value:\n";

**for**(**int** i = 1; i <= mylist.length(); i++)

{

value = mylist.currentValue();

mylist.next();

std::cout << std::setw(6) << value[0] << std::setw(6) << value[1] << std::endl;

}

mylist.sort(1);

std::cout << "List values sorted by second value:\n";

**for**(**int** i = 1; i <= mylist.length(); i++)

{

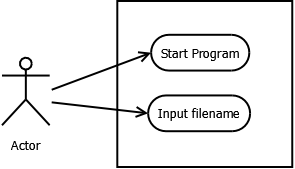
value = mylist.currentValue();

mylist.next();

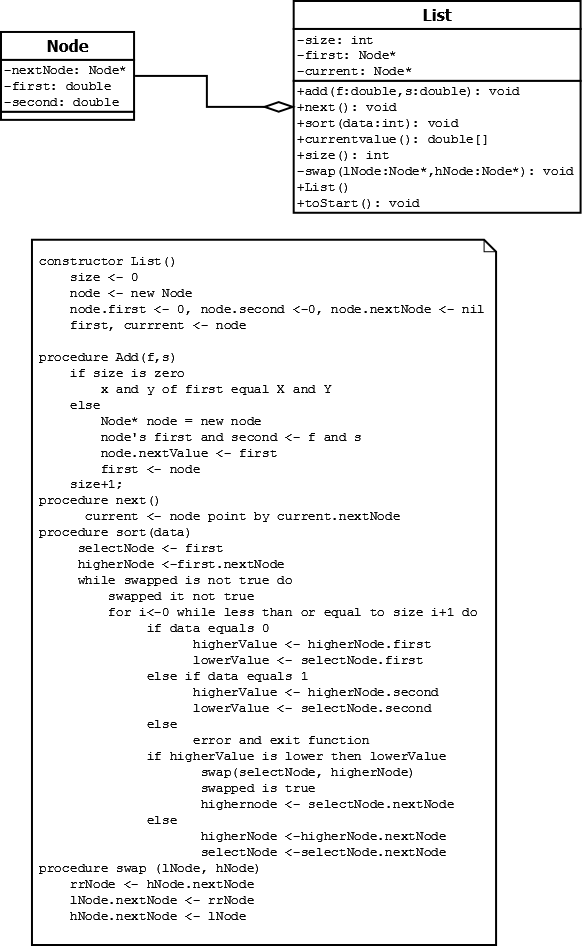
std::cout << std::setw(6) << value[0] << std::setw(6) << value [1] << std::endl;

}

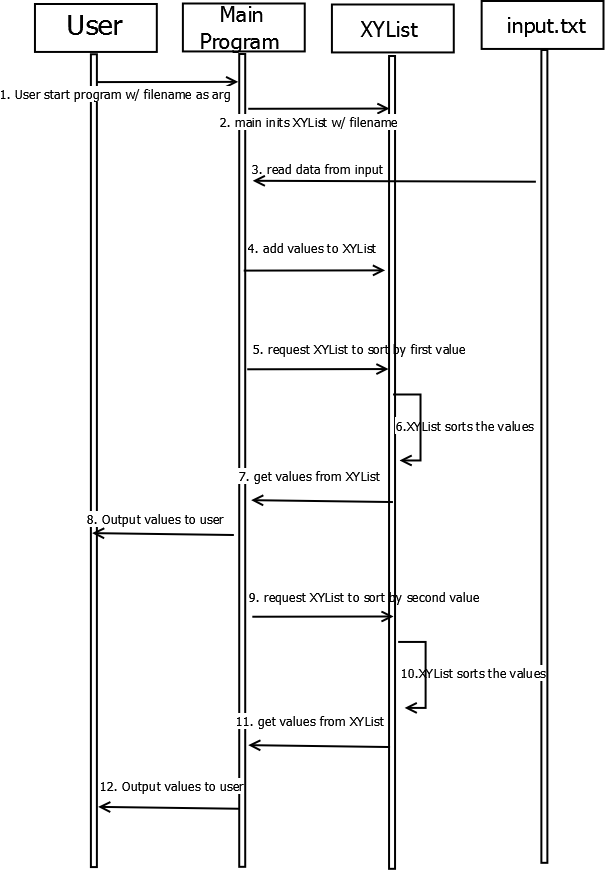
}

**Use-Case Diagram:**

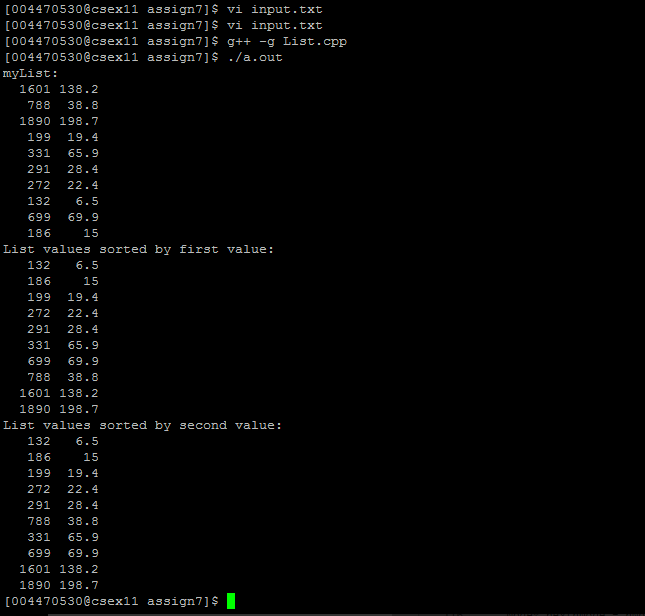
**UML Diagram and PseudoCode:**

****

**Sequence Diagram:**

****

**Screen cap of Compilation and Run:**

****

**Input Table:**

|  |  |  |
| --- | --- | --- |
| Item Number | Actual New and Changed LOC | Development Hours |
| n | x | y |
| 1 | 186 | 15.0 |
| 2 | 699 | 69.9 |
| 3 | 132 | 6.5 |
| 4 | 272 | 22.4 |
| 5 | 291 | 28.4 |
| 6 | 331 | 65.9 |
| 7 | 199 | 19.4 |
| 8 | 1890 | 198.7 |
| 9 | 788 | 38.8 |
| 10 | 1601 | 138.2 |

**Output Table:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Unsorted List** | | **Sorted by First Value** | | **Sorted by Second Value** | |
| **index** | **first** | **second** | **1st** | **value** | **index** | **value** |
| **1** | **1601** | **138.2** | **132** | **6.5** | **132** | **6.5** |
| **2** | **788** | **38.8** | **186** | **15** | **186** | **15** |
| **3** | **1890** | **198.7** | **199** | **19.4** | **199** | **19.4** |
| **4** | **199** | **19.4** | **272** | **22.4** | **272** | **22.4** |
| **5** | **331** | **65.9** | **291** | **28.4** | **291** | **28.4** |
| **6** | **291** | **28.4** | **331** | **65.9** | **331** | **65.9** |
| **7** | **272** | **22.4** | **699** | **69.9** | **699** | **69.9** |
| **8** | **132** | **6.5** | **788** | **38.8** | **788** | **38.8** |
| **9** | **699** | **69.9** | **1601** | **138.2** | **1601** | **138.2** |
| **10** | **186** | **15** | **1890** | **198.7** | **1890** | **198.7** |

**Note: The list adds values buy making each new value the first value, so the data is read in correctly but the list has the last value as the first value and vice versa.**