Ahmad Mohammad

c1001633 / asm6t

proj3

		deduction
TURNIN TIME	turned in on time.	0 %
SOURCE CODE SEARCH RESULTS	MISSING: None. FOUND: operator, friend, getArea, rectangles.txt, rectangle.h, tsllist.h, template, clearList, printAll, insertInOrder, deleteVal, deleteAllVal, struct TSLLNode	0

DEDUCTIONS	-0
FINAL GRADE	100

COMPILATION LOG

```
c1001633 MAKE PASS

MAKELOG OF STUDENT PROVIDED MAKEFILE:
g++ -c -g -O0 -std=c++11 -Wall rectangle.cpp -o rectangle.o -MMD -MF rectangle.d
g++ -c -g -O0 -std=c++11 -Wall proj3.cpp -o proj3.o -MMD -MF proj3.d
g++  -o proj3 rectangle.o proj3.o
```

c1001633 runlog begin:
Test 1:
Inputs:
7 2 a
3 4 a
4 5 a
3 5 d
2 6 a
4 4 a
12 1 D
2 8 a
8 2 d

Output:

a(7,2): ->[[L:7 W:2 (A 14)]]
a(3,4): ->[[L:3 W:4 (A 12)]]->[[L:7 W:2 (A 14)]]
a(4,5): ->[[L:3 W:4 (A 12)]]->[[L:7 W:2 (A 14)]]->[[L:4 W:5 (A 20)]]
d(3,5): Returned [L:0 W:0 (A 0)]->[[L:3 W:4 (A 12)]]->[[L:7 W:2 (A 14)]]->[[L:4 W:5 (A 20)]]
a(2,6): ->[[L:3 W:4 (A 12)]]->[[L:2 W:6 (A 12)]]->[[L:7 W:2 (A 14)]]->[[L:4 W:5 (A 20)]]
a(4,4): ->[[L:3 W:4 (A 12)]]->[[L:2 W:6 (A 12)]]->[[L:7 W:2 (A 14)]]->[[L:4 W:4 (A 16)]]->[[L:4 W:5 (A 20)]]
D(12,1): ->[[L:7 W:2 (A 14)]]->[[L:4 W:4 (A 16)]]->[[L:4 W:5 (A 20)]]
a(2,8): ->[[L:7 W:2 (A 14)]]->[[L:4 W:4 (A 16)]]->[[L:2 W:8 (A 16)]]->[[L:4 W:5 (A 20)]]
d(8,2): Returned [L:4 W:4 (A 16)]->[[L:7 W:2 (A 14)]]->[[L:2 W:8 (A 16)]]->[[L:4 W:5 (A 20)]]
Clearing [L:7 W:2 (A 14)]
Clearing [L:2 W:8 (A 16)]
Clearing [L:4 W:5 (A 20)]

Test 2:
Inputs:
8 2 a
10 3 a
2 2 a
4 1 d
6 5 a
2 15 a
5 6 D
1 4 a

Output:
a(8,2): ->[[L:8 W:2 (A 16)]]
a(10,3): ->[[L:8 W:2 (A 16)]]->[[L:10 W:3 (A 30)]]
a(2,2): ->[[L:2 W:2 (A 4)]]->[[L:8 W:2 (A 16)]]->[[L:10 W:3 (A 30)]]
d(4,1): Returned [L:2 W:2 (A 4)]->[[L:8 W:2 (A 16)]]->[[L:10 W:3 (A 30)]]
a(6,5): ->[[L:8 W:2 (A 16)]]->[[L:10 W:3 (A 30)]]->[[L:6 W:5 (A 30)]]
a(2,15): ->[[L:8 W:2 (A 16)]]->[[L:10 W:3 (A 30)]]->[[L:6 W:5 (A 30)]]->[[L:2 W:15 (A 30)]]
D(5,6): ->[[L:8 W:2 (A 16)]]
a(1,4): ->[[L:1 W:4 (A 4)]]->[[L:8 W:2 (A 16)]]
Clearing [L:1 W:4 (A 4)]
Clearing [L:8 W:2 (A 16)]

```
Manifest: proj3.cpp.lst
-----------------------
-----------------------
 Feb 25 10:11 proj3.cpp
```

```
    1   // Ahmad Mohammad
    2   // CSCI-3110-002
    3   // Proj 3
    4   // Due : 02/24/22
    5
    6   // This program will create objects representing rectangles and store them in ascendin
g are
    7   // a order it will also instantiate a list and apply operations read from the input fi
le.
    8
    9   #include<iostream>
   10
   11   #include<string>
   12   #include<fstream>
   13   #include "rectangle.h"
   14   #include "tsllist.h"
   15
   16   using namespace std;
   17
   18
   19
   20   int main()
   21   {
   22                   //Declaring input variables
   23       double length, width;
   24                   string letter;
   25                   // Instantiating Rectangle object "chick"
   26       Rectangle rectobj;
   27       // Instantiating TSLList object of type Rectangles
   28       TSLList<Rectangle> rectlist;
   29
   30                   // Open in file
   31       ifstream myin;
   32       myin.open("rectangles.txt");
   33
   34                   //loop through file
   35       while(myin >> length)
   36       {
   37           // pulling letter right after number from file
   38           myin >> width >> letter;
   39
   40                                rectobj.setLength(length);
   41                                rectobj.setWidth(width);
   42
   43           // if letter is 'a' then insert num and cout info in format
   44           if(letter == "a")
   45           {
   46                   cout<<letter<<'('<<rectobj.getLength()<<','<<rectobj.getWidth()<<"): "
;
   47                   rectlist.insertInOrder(rectobj);
   48                   rectlist.printAll();
   49           }
   50           // if letter is 'd' then delete num pulled from file that
   51           // occurs first in list  and cout info in format
   52           else if(letter == "d")
   53           {
   54                   cout<<letter<<'('<<rectobj.getLength()<<','<<rectobj.getWidth()<<"): "
;
   55                   cout << "Returned " << rectlist.deleteVal(rectobj);
   56                   rectlist.printAll();
   57
```

```
   58           }
   59           // if letter = 'D' then delete all numbers in list that = num pulled from file
   60           // and cout info in correct format
   61           else if(letter == "D")
   62           {
   63                   cout<<letter<<'('<<rectobj.getLength()<<','<<rectobj.getWidth()<<"): "
;
   64                   rectlist.deleteAllVal(rectobj);
   65                   rectlist.printAll();
   66
   67           }
   68
   69       }
   70       // end prog
   71       return 0;
   72   }
   73
   74
   75
   76
   77
---------------------- tsllist.h: ------------------------
    1  #ifndef T_LINKED_LIST
    2  #define T_LINKED_LIST
    3  #include<iostream>
    4
    5  template <typename T>
    6
    7  class TSLList {
    8      public:
    9
   10          // Constructor
   11          TSLList() {
   12              head = nullptr;
   13          }
   14
   15          //D Destructor
   16          ~TSLList() {
   17              clearList();
   18          }
   19
   20          // prints the info content and address of each node in the list
   21          void printAll() const {
   22              for (TSLLNode *tmp = head; tmp != nullptr; tmp = tmp->next)
   23                  std::cout << "->[" << tmp->info << "]";
   24              std::cout << std::endl;
   25          }
   26
   27          // Inserts node in order (see assignment specification for details)
   28          void insertInOrder(T number)
   29          {
   30              TSLLNode *newnode, *cur, *prev;
   31
   32              // dynamically allocating memory to new node and giving value passed in pa
ram.
   33              newnode = new TSLLNode(number);
   34              newnode -> info = number;
   35
   36              // preparing cur and prev for traversal
   37              cur = head;
   38              prev = NULL;
```

```
    39
    40              // checks if head is NULL and if so sets the newnode to be head (empty lis
t)
    41              if(!head)
    42                  {
    43                      head = newnode;
    44                      newnode->next = NULL;
    45                  }
    46              // otherwise traverse until we find valie
    47              else
    48                  {
    49                      // traversal loop which makes sure cur != NULL and stops when we r
each
    50                      // where we want to insert node.
    51                      while(cur != NULL && cur->info <= number)
    52                      {
    53                          prev = cur;
    54                          cur = cur->next;
    55                      }
    56                      // sets head to equal new node if both cur and prev are NULL
    57                      // ie. while loop did not run
    58                      if(prev == NULL)
    59                      {
    60                          head = newnode;
    61                          newnode->next = cur;
    62                      }
    63                      // otherwise set prev->next to newnode and newnode->next to cur
    64                      // in order to keep cur 2 nodes ahead of prev and not get seg faul
t
    65                      else
    66                      {
    67                          prev->next = newnode;
    68                          newnode->next = cur;
    69                      }
    70                  }
    71
    72  }
    73
    74          // Deletes an occurrence of argument (see assignment specification for details
)
    75          T deleteVal(T number)
    76          {
    77              TSLLNode *cur, *prev;
    78
    79                          T x, y;
    80
    81              cur = head;
    82              // checks if head = NULL and if so exits
    83              if(!head)
    84                          {
    85                                  T trash1;
    86                  return trash1;
    87                          }
    88
    89              // checks if head is node to be deleted if so deletes and returns cur whic
h = head
    90              if( head != NULL && head->info == number)
    91              {
    92                  cur = head;
    93                  head = head->next;
    94                  x = cur -> info;
```

```
    95                  cur -> next = NULL;
    96                                  delete cur;
    97                  cur = head;
    98                  return x;
    99
   100              }
   101              //other wise node to be deleted is in list somewhere...
   102              else
   103              {
   104
   105                  // traversal loop to locate node to be deleted
   106                  while(cur != NULL && cur->info != number)
   107                  {
   108                      prev = cur;
   109                      cur = cur->next;
   110                                  }
   111                  // makes sure cur data = number because if not we would get seg fault
   112                  // trying to delete NULL node
   113                  if(cur != NULL && cur -> info == number)
   114                  {
   115                      // makes sure all data from cur is wiped before delete and returns
 value we
   116                      // were looking for
   117                      prev-> next = cur->next;
   118                      cur-> next = NULL;
   119                                          prev = cur;
   120                                          y = prev-> info;
   121                                          delete cur;
   122                                          return y;
   123
   124                                  }
   125
   126                  }
   127                          // return default const obj.
   128              T trash2;
   129                          return trash2;
   130              }
   131
   132          // Deletes all occurrences of argument (see assignment specification for detai
ls)
   133          void deleteAllVal(T number)
   134          {
   135              TSLLNode *cur, *prev;
   136
   137              // checks if head = null.. if so exits
   138              if(!head)
   139                  return;
   140              // if head = number we want to delete we delete head and set head to cur
   141              // while head is = number we want gone
   142              if(head != NULL && head->info == number)
   143              {
   144                  while(head != NULL && head->info == number)
   145                  {
   146                      //cur = head->next;
   147                      //delete head;
   148                      //head = cur;
   149                                          cur = head;
   150                                          head = head->next;
   151                                          delete cur;
   152                  }
   153              }
```

```
   154              // otherwise the number we want to delete is in the list somewhere...
   155              else
   156              {
   157                  // set cur =  head for traversal
   158                  cur = head;
   159
   160                  // while list is a list and cur's data != number, traverse the list
   161                  while(cur != NULL && cur->info != number)
   162                  {
   163                      prev = cur;
   164                      cur = cur->next;
   165                  }
   166                  // if cur -> info = number then enter while loop and delete nodes
   167                  // until it is not equal to number
   168                  if(cur != NULL && cur -> info == number)
   169                  {
   170                      while(cur != NULL && cur->info == number)
   171                      {
   172                          prev->next = cur->next;
   173                          delete cur;
   174                          cur = prev->next;
   175
   176                      }
   177                  }
   178              }
   179          }
   180
   181          // Clears the list (deallocates memory - see assignment specification for deta
ils)
   182          void clearList()
   183          {
   184              TSLLNode *cur;
   185              while(head)
   186                  {
   187                      // clearing message
   188                      std::cout << "Clearing " << head->info << std::endl;
   189                      //setting cur to head to delete bc we cant delete head
   190                      cur = head;
   191                      // setting head to next node in list
   192                      head = head -> next;
   193                      // deleting old head
   194                      delete cur;
   195          }
   196          }
   197
   198      private:
   199          // Node stored in linked list
   200          struct TSLLNode {
   201              TSLLNode(T el = T()) {
   202                  info = el;
   203                  next = nullptr; }
   204                  T info;
   205                  TSLLNode *next; };
   206
   207
   208          TSLLNode *head;   // head of the list
   209  };
   210
   211
   212  #endif
   213
```

```
   214
---------------------- rectangle.h: ------------------------
     1  #ifndef RECTANGLE_H
     2  #define RECTANGLE_H
     3
     4  #include <iostream>
     5  #include "tsllist.h"
     6
     7
     8  using std::ostream;
     9
    10  class Rectangle
    11  {
    12    public:
    13
    14      Rectangle(int l = 0, int w = 0)                                        // default con
structor
    15                  { length = l; width = w; area = length * width; }
    16
    17      void setLength(int l)                                                  // len
gthgth mutator (setter) - updates area member
    18                  { length = l; area = length * width; }
    19
    20      void setWidth(int w)                                                   // wid
th mutator (setter) - updates area member
    21                  { width = w; area = length * width; }
    22
    23          int getLength() const                                              // len
gthgth accessor (getter)
    24                  { return length; }
    25
    26          int getWidth() const                                               // wid
th accessor (getter)
    27                  { return width; }
    28
    29          int getArea() const
// area accessor (getter)
    30                  { return area; }
    31
    32          friend ostream& operator << (ostream& os, const Rectangle & rect)   // outputs
 a Rectangle object
    33          {
    34                  os << "[L:" << rect.length << " W:" << rect.width << " (A " << rect.ar
ea << ")]";
    35                  return os;
    36          }
    37
    38
    39          // implement overloads below
    40          bool operator<(const Rectangle &);
    41
    42          bool operator<=(const Rectangle &);
    43
    44          bool operator>(const Rectangle &);
    45
    46          bool operator>=(const Rectangle &);
    47
    48          bool operator==(const Rectangle &);
    49
    50          bool operator!=(const Rectangle &);
    51
```

```
    52    private:
    53            int length;
// length data member
    54
    55            int width;
// width data member
    56
    57            int area;
// area data member
    58
    59  };
    60
    61  #endif
    62
\---------------------- rectangle.cpp: ------------------------
     1  #include "rectangle.h"
     2  #include "iostream"
     3
     4  using namespace std;
     5
     6  bool Rectangle::operator<(const Rectangle & rObj)
     7  {
     8      return area < rObj.area;
     9  }
    10  bool Rectangle::operator<=(const Rectangle & rObj)
    11  {
    12      return area <= rObj.area;
    13  }
    14  bool Rectangle::operator>(const Rectangle & rObj)
    15  {
    16      return area > rObj.area;
    17  }
    18  bool Rectangle::operator>=(const Rectangle & rObj)
    19  {
    20      return area >= rObj.area;
    21  }
    22  bool Rectangle::operator==(const Rectangle & rObj)
    23  {
    24      return area == rObj.area;
    25  }
    26  bool Rectangle::operator!=(const Rectangle & rObj)
    27  {
    28      return area != rObj.area;
    29  }
```