Adventure game: header file documentation

```
// stringapps.h - John K. Estell - 5 February 2002
// extensions to the string class functions

#ifndef STRINGAPPS_H
#define STRINGAPPS_H

#include <string>
using namespace std;

string trim( const string s );
string toLowercase( const string s );

#endif
```

trim: eliminates all leading and trailing whitespace present in a string.

toLowercase: returns a string where all letter characters are in lowercase. Useful for when case insensitive matches are needed.

Adventure game: header file documentation

```
// room.h - John K. Estell - 31 January 2002
// header file for implementation of a room.  Implementation supports travel in six
// directions (north, south, east, west, up, down), items in room, and both brief and
// full text descriptions of the room.

#ifndef ROOM_H
#define ROOM_H

#include <string>
#include <vector>
#include "item.h"
#include "command.h"
using namespace std;


const int BLOCKED = -1;   // indicates travel is blocked in that direction

class Room {
   public:
      Room( int n, int s, int e, int w, int u, int d, int pts,
            string briefDescr, string longDescr );
      bool canTravelOnThis( direction heading ) const;
      bool hasVisitedRoom( bool isInRoom );
      int getRoomPoints() const;
      int getRoomTravelingOnThis( direction heading ) const;
      string getBriefRoomDescription() const;
      string getRoomDescription() const;
      vector<Item *> getRoomContents();
      bool isItemInRoom( string namedItem );
      Item *removeFromRoom( string namedItem );
      void addToRoom( Item *thisItem );

   private:
      int roomTravelDirection[ 6 ];  // entries correspond to order in direction enum type
      int roomPoints;
      bool hasVisitedFlag;
      string briefRoomDescription;
      string roomDescription;
      vector<Item *> roomContents;
};

vector<Room *> getRoomDataRecords( string roomDataFile );

#endif
```

Room: constructor - first six fields are used to indicate what room would be visited if one travels in that direction; BLOCKED (= -1) indicates that you can't travel in that direction. 'pts' is used to indicate the number of points scored for visiting a particular room. 'briefDescr' is a one-line string used to provide a brief description of a room. 'longDescr' is a string with embedded newlines for a formatted (i.e. how it appears in the data file is how it appears on the screen) full description of the room.

canTravelOnThis: returns true if the explorer can travel from the current room to another room on the provided directional heading. Routine returns false if blocked from doing so.

hasVisitedRoom: if explorer is currently in the room, then return the status and set the visitation flag to true. Passing false just checks the status of the room.

getRoomPoints: returns the number of points awarded for visiting this room.

Adventure game: header file documentation

getRoomTravelingOnThis: returns the room number for the room one would reach if traveling in the specified direction. Will return BLOCKED if there is nowhere to go in that direction. Should be used after a call to canTravelOnThis(heading).

getBriefRoomDescription: returns the short (one-line) description of a room, suitable for when one re-visits a room.

getRoomDescription: returns the full (multi-line) description of a room, suitable for the initial visit to a room and when user asks to LOOK at the room.

getRoomContents: returns a vector of pointers to Item that indicate what items are currently present in the room.

isItemInRoom: indicates whether or not the named item is present in the room. Can just pass the string as is - routine will convert to lowercase for performing the match. Currently does not handle compressing multiple whitespaces; if item is "`cat food`" and you check for "`cat  food`" (2 spaces) then routine will fail as currently implemented.

removeFromRoom: removes the named item from the room and returns a pointer to the item object. Returns the null pointer if item is not found. Ideally should be called after checking using isItemInRoom().

addToRoom: add the passed item to the current room.

getRoomDataRecords: reads in room information from specified data file. Rooms are stored as multiple-line records in the data file:

- Line 1: data in comment form containing room number. Rooms are sequentially enumerated beginning at zero.
- Usually used for documenting the data file only - can safely ignore within this routine once line is read in.
- Line 2: six integers indicating the ability to travel (in order) north, south, east, west, up, or down from the current room. The value -1 is used to indicate that one cannot travel in that direction; any other value indicates the room number of the room that would be visited if one travels in that direction.
- Line 3: points received for visiting this room.
- Line 4: name of image file used by your graphics package to illustrate where you are. Make sure there is no whitespace in the filename. Just read and ignore (or remove entirely) if not using graphics.
- Line 5: one line (i.e. short) description of where you are. Short descriptions are often used when revisiting a room.
- Line 6: start of (long) descriptive room text. Text will be formatted on the display exactly as it appears in the data file. Multiple lines are permitted; you may assume that a single line of input contains no more than 100 characters. The word `END` at the beginning of a line is used to indicate the end of the descriptive text.

Adventure game: header file documentation

```
// item.h - John K. Estell - 31 January 2002
// Implementation of an item object.

#ifndef ITEM_H
#define ITEM_H

#include <string>
#include <vector>
using namespace std;

class Item {
   public:
      Item( string name, int start, int target, int targetValue );
      int getStartingRoom();
      string getItemName();                  // used to display name
      string getLowercaseItemName();         // used for string comparisions
      int getTargetRoom();
      int getTargetRoomPoints();
   private:
      int startingRoom;
      string itemName;
      string lowercaseItemName;
      int targetRoom;
      int targetRoomPoints;
};

vector<Item *> getItemDataRecords( string itemDataFile );

#endif
```

Item: constructor used to create an instance of an item. Note that lowercase version of the name of the item should be constructed here for use in case-insensitive comparisons later on...

getStartingRoom: returns room number where item is initially located.

getItemName: returns the name of the item as formatted by the designer. Use this function when displaying item names to the player.

getLowercaseItemName: returns the name of the item, but all characters are in lowercase. Use this function when performing case-insensitive searches for a particular item.

getTargetRoom: returns room number where item is to be deposited in order to score points.

getTargetRoomPoints: returns the number of points scored if the particular item in question is located within the specified target room.

getItemDataRecords: reads the specified item data file to obtain the list of items to be used in the game. The format used in this routine consists of one record per line with the fields:

```
   startRm endRm pts itemName
```

where 'startRm' and 'endRm' are integers containing room numbers where the item is originally located and where it is to end up, 'pts' is an integer showing the number of points scored when item is placed in end room. First non-whitespace that follows up to end of line is the name of the item. Routine returns a vector of pointers to Item.

Adventure game: header file documentation

```
// command.h - John K. Estell - 1 February 2002
// command parser interface.  When a Command object is instantiated, the passed string is
// parsed into <verb, noun> format, which is defined as: first word is a verb or a
direction,
// and all remaining words comprise the noun.

#ifndef COMMAND_H
#define COMMAND_H

#include <string>
using namespace std;

// enumerated data types - used to indicate the supported directions and verbs
// for the program.  Supported nouns (other than directions) are obtained from
// the item data file.  The 'Unknown' values are used when there's no match.
enum direction { North, South, East, West, Up, Down, UnknownDirection };
enum verb { Move, Look, Score, Quit, Take, Drop, Inventory, Help, UnknownVerb };

class Command {
   public:
      Command( string input );
      verb getVerb();
      string getNoun();
      direction getDirection();
   private:
      verb commandVerb;                // store for the selected action
      string commandNoun;              // store for the noun
      direction commandDirection;      // store for the direction of travel
};


// auxiliary functions
verb parseVerb( string verbString );
direction parseDirection( string directionString );

#endif
```

Command: constructor takes input string and parses it into <verb, noun> format. Will also look for "action nouns" (i.e. one-word movement commands) and convert into <Move, direction>. If verb is Move (either directly or implicitly), then noun is processed for direction and stored as an enumerated direction type.

getVerb: returns the appropriate value from the enumerated verb type.

getNoun: return the trimmed string containing the noun. Note that nothing else is done to the noun other than trim (i.e. no change in case, whitespace compression, etc.)

getDirection: returns the appropriate value from the enumerated direction type.

parseVerb: processes string and looks for matching verb. Returns UnknownVerb if an unrecognized verb string is passed.

parseDirection: processes string and looks for matching direction. Returns UnknownDirection if an unrecognized direction string is passed.