Report

Luke McDougall

1. Input file conversion

The input file was converted to a coordinate system by storing the contents of the file in a two dimensional array of strings. The dimensions of this array are specified in the first line of the input file. The y coordinate is the index of the array of arrays and the x coordinate is the index of the array of strings. If there is a valid entry in any position in the input file the contents of that entry are stored as a string in the corresponding element of the array. If an entry is empty the corresponding element in the array is set to NULL.

The dimensions specified in the first line of the input file need to be validated. The c function fgetc is used extensively for this. After reading the first line of the file fgetc is called in a while loop. A variable called len is incremented every time fgetc returns a comma and a variable named height is incremented and len is reset to one(there is one less comma than the number of entries per line) every time fgetc returns the newline character. The loop is exited if len is not equal to the number of columns specified in the first line when a newline character is returned or when fgetc returns EOF. After the loop is exited the height variable is compared to the number of rows specified in the first line. If len always equals the number of columns and height is equal to the number of rows the file dimensions are valid.

After the dimensions have been validated the individual entries are validated and if they are all valid they are stored in the array. To find all of the entries the function read_line is used to return each line from the file as a string. Two char pointers start and end are used to find non empty entries. The value of start is the memory address of the first element in the string and end is the return value of the c function strchr being passed start and a comma. If the difference between end and start is greater than one the characters between start and end are validated and if they are valid copied into the array. Otherwise the corresponding array element is set to NULL. The last entry in a line will not have a comma at the end and there for strchr will return NULL. In this case all characters between start and the end of the string are copied into the array if there are more than one characters. The c function sscanf is used to separate a string to be validated into the components that will be stored in a struct. If sscanf fails to do this or any of the components are invalid eg a value being negative the entry is considered invalid. If any of the entries are invalid the array is freed an error message is printed to the console.

1.1. Alternative approach to input file conversion

An alternative method could be to "trust" the first line of the input file and only report failure if any row or column has less elements than the first line specifies ignoring any extra elements if they exist. Instead of validating the files dimensions first skip straight to the second stage of validation. An alternative to a two dimensional array of strings could be an array of pointers to structs. To avoid having an array of void pointers a generic "pick-up" struct would be created that contains a char field that specifies the type of pick-up it is and any of the fields that aren’t relevant to that pick-up type would be set to NULL.

This would avoid parsing the strings twice, once for validation and once for struct initialization, because both validation and initialization would be done at the same time. This would lessen the strictness of the criteria for a valid input file, any file that isn’t too small would be considered valid.

2. Demonstration

2.1. Demonstration of the contents of the array after loading a file.

```
TreasureHunter map.csv adventure.csv
```

Contents of file:

```
5,4
,,C 200,
G Iron Chestplate:chest:200,G Mithril Saber:hands:990,G Mithril Leggings:legs:950,C 50
M Healing Potion:85,M Defence Enchantment:360,G Iron Helmet:head:250,
,G Iron Leggings:legs:250,G Mithril Chestplate:chest:1050,
,G Mithril Helmet:head:950,G Rune Saber:hands:850,
```

Contents of array:

```
NULL,NULL,C 200,NULL,
G Iron Chestplate:chest:200,G Mithril Saber:hands:990,G Mithril Leggings:legs:950,C 50,
M Healing Potion:85,M Defence Enchantment:360,G Iron Helmet:head:250,NULL,
NULL,G Iron Leggings:legs:250,G Mithril Chestplate:chest:1050,NULL,
NULL,G Mithril Helmet:head:950,G Rune Saber:hands:850,NULL,
```

2.2.

Demonstration of the output from TreasureHunter

Valid input files:

Contents of map.csv:

```
5,4
,,C 200,
G Iron Chestplate:chest:200,G Mithril Saber:hands:990,G Mithril Leggings:legs:950,C 50
M Healing Potion:85,M Defence Enchantment:360,G Iron Helmet:head:250,
,G Iron Leggings:legs:250,G Mithril Chestplate:chest:1050,
,G Mithril Helmet:head:950,G Rune Saber:hands:850,
```

Contents of adventure.csv:

```
DOWN 2
RIGHT 2
DOWN 2
LEFT 1
UP 3
RIGHT 2
```

```
TreasureHunter map.csv adventure.csv
```

Console ouput:

```
COMPLETE
COINS: 50
ITEMS: 445
GEAR: 3940
```

adventure.log output:

```
 ---
 COLLECT<GEAR, XLOC: 0, XLOC: 1, DESCRIPTION: Iron Chestplate, SLOT: chest, VALUE: 200>
 COLLECT<ITEM, XLOC: 0, YLOC: 2, DESCRIPTION: Healing Potion, VALUE: 85>
 COLLECT<ITEM, XLOC: 1, YLOC: 2, DESCRIPTION: Defence Enchantment, VALUE: 360>
 COLLECT<GEAR, XLOC: 2, XLOC: 2, DESCRIPTION: Iron Helmet, SLOT: head, VALUE: 250>
 DISCARD<GEAR, XLOC: 2, XLOC: 3, DESCRIPTION: Iron Chestplate, SLOT: chest, VALUE: 200>
 COLLECT<GEAR, XLOC: 2, XLOC: 3, DESCRIPTION: Mithril Chestplate, SLOT: chest, VALUE:
1050>
 COLLECT<GEAR, XLOC: 2, XLOC: 4, DESCRIPTION: Rune Saber, SLOT: hands, VALUE: 850>
 DISCARD<GEAR, XLOC: 1, XLOC: 4, DESCRIPTION: Iron Helmet, SLOT: head, VALUE: 250>
 COLLECT<GEAR, XLOC: 1, XLOC: 4, DESCRIPTION: Mithril Helmet, SLOT: head, VALUE: 950>
 COLLECT<GEAR, XLOC: 1, XLOC: 3, DESCRIPTION: Iron Leggings, SLOT: legs, VALUE: 250>
 DISCARD<GEAR, XLOC: 1, XLOC: 1, DESCRIPTION: Rune Saber, SLOT: hands, VALUE: 850>
 COLLECT<GEAR, XLOC: 1, XLOC: 1, DESCRIPTION: Mithril Saber, SLOT: hands, VALUE: 990>
 DISCARD<GEAR, XLOC: 2, XLOC: 1, DESCRIPTION: Iron Leggings, SLOT: legs, VALUE: 250>
 COLLECT<GEAR, XLOC: 2, XLOC: 1, DESCRIPTION: Mithril Leggings, SLOT: legs, VALUE: 950>
 COLLECT<COINS, XLOC: 3, YLOC: 1, VALUE: 50>
```

Invalid map, invalid dimensions:

Contents of map2.csv:

```
 5,5
 ,,C 200,
 ,G Vibranium Shield:hands:990,,C 50
 M Healing Potion:85,,M Defence Enchantment:360,
 ,,,
 ,,G Lightsaber:hands:850,
```

Contents of adventure.csv:
Same as above.

```
TreasureHunter map2.csv adventure.csv
```

Console output:

```
 Error: invalid file format at map2.csv!
 ABORTED
```

Invalid map, invalid entry:

Contents of map3.csv

```
 5,4
 ,,p 200,
 ,t Vibranium Shield:hands:990,,C 50
 M Healing Potion:85,,M Defence Enchantment:360,
 ,,,
 ,,G Light:sab:er:hands:850,
```

Contents of adventure.csv
Same as above.

Console output:

```
 Error: invalid file format at map3.csv!
 ABORTED
```

Invalid adventure, out of bounds:

Contents of map.csv
Same as above

Contents of adventure2.csv

```
 right 2
 left 1
 down 1
 right 12
 down 2
 down 3
```

Console output:

```
FAILED
```

adventure.log output:

```
 ---
 COLLECT<COINS, XLOC: 2, YLOC: 0, VALUE: 200>
 COLLECT<GEAR, XLOC: 1, XLOC: 1, DESCRIPTION: Mithril Saber, SLOT: hands, VALUE: 990>
```

Invalid adventure, out of bounds. In TreasureHunterAI mode.

Contents of map.csv
Same as above

Contents of adventure2.csv
Same as above

Console output:

```
 CORRECTED
 COINS: 250
 ITEMS: 0
 GEAR: 1940
```

adventure.log output:

```
 ---
 COLLECT<COINS, XLOC: 2, YLOC: 0, VALUE: 200>
 COLLECT<GEAR, XLOC: 1, XLOC: 1, DESCRIPTION: Mithril Saber, SLOT: hands, VALUE: 990>
 COLLECT<GEAR, XLOC: 2, XLOC: 1, DESCRIPTION: Mithril Leggings, SLOT: legs, VALUE: 950>
 COLLECT<COINS, XLOC: 3, YLOC: 1, VALUE: 50>
```

Valid files, TreasureHunterLog mode.

Contents of map.csv
Same as above

Contents of adventure.csv
Same as above

Console output:

```
---
COLLECT<GEAR, XLOC: 0, XLOC: 1, DESCRIPTION: Iron Chestplate, SLOT: chest, VALUE: 200>
COLLECT<ITEM, XLOC: 0, YLOC: 2, DESCRIPTION: Healing Potion, VALUE: 85>
COLLECT<ITEM, XLOC: 1, YLOC: 2, DESCRIPTION: Defence Enchantment, VALUE: 360>
COLLECT<GEAR, XLOC: 2, XLOC: 2, DESCRIPTION: Iron Helmet, SLOT: head, VALUE: 250>
DISCARD<GEAR, XLOC: 2, XLOC: 3, DESCRIPTION: Iron Chestplate, SLOT: chest, VALUE: 200>
COLLECT<GEAR, XLOC: 2, XLOC: 3, DESCRIPTION: Mithril Chestplate, SLOT: chest, VALUE: 
1050>
COLLECT<GEAR, XLOC: 2, XLOC: 4, DESCRIPTION: Rune Saber, SLOT: hands, VALUE: 850>
DISCARD<GEAR, XLOC: 1, XLOC: 4, DESCRIPTION: Iron Helmet, SLOT: head, VALUE: 250>
COLLECT<GEAR, XLOC: 1, XLOC: 4, DESCRIPTION: Mithril Helmet, SLOT: head, VALUE: 950>
COLLECT<GEAR, XLOC: 1, XLOC: 3, DESCRIPTION: Iron Leggings, SLOT: legs, VALUE: 250>
DISCARD<GEAR, XLOC: 1, XLOC: 1, DESCRIPTION: Rune Saber, SLOT: hands, VALUE: 850>
COLLECT<GEAR, XLOC: 1, XLOC: 1, DESCRIPTION: Mithril Saber, SLOT: hands, VALUE: 990>
DISCARD<GEAR, XLOC: 2, XLOC: 1, DESCRIPTION: Iron Leggings, SLOT: legs, VALUE: 250>
COLLECT<GEAR, XLOC: 2, XLOC: 1, DESCRIPTION: Mithril Leggings, SLOT: legs, VALUE: 950>
COLLECT<COINS, XLOC: 3, YLOC: 1, VALUE: 50>
COMPLETE
COINS: 50
ITEMS: 445
GEAR: 3940
```

adventure.log output:

```
---
COLLECT<GEAR, XLOC: 0, XLOC: 1, DESCRIPTION: Iron Chestplate, SLOT: chest, VALUE: 200>
COLLECT<ITEM, XLOC: 0, YLOC: 2, DESCRIPTION: Healing Potion, VALUE: 85>
COLLECT<ITEM, XLOC: 1, YLOC: 2, DESCRIPTION: Defence Enchantment, VALUE: 360>
COLLECT<GEAR, XLOC: 2, XLOC: 2, DESCRIPTION: Iron Helmet, SLOT: head, VALUE: 250>
DISCARD<GEAR, XLOC: 2, XLOC: 3, DESCRIPTION: Iron Chestplate, SLOT: chest, VALUE: 200>
COLLECT<GEAR, XLOC: 2, XLOC: 3, DESCRIPTION: Mithril Chestplate, SLOT: chest, VALUE: 
1050>
COLLECT<GEAR, XLOC: 2, XLOC: 4, DESCRIPTION: Rune Saber, SLOT: hands, VALUE: 850>
DISCARD<GEAR, XLOC: 1, XLOC: 4, DESCRIPTION: Iron Helmet, SLOT: head, VALUE: 250>
COLLECT<GEAR, XLOC: 1, XLOC: 4, DESCRIPTION: Mithril Helmet, SLOT: head, VALUE: 950>
COLLECT<GEAR, XLOC: 1, XLOC: 3, DESCRIPTION: Iron Leggings, SLOT: legs, VALUE: 250>
DISCARD<GEAR, XLOC: 1, XLOC: 1, DESCRIPTION: Rune Saber, SLOT: hands, VALUE: 850>
COLLECT<GEAR, XLOC: 1, XLOC: 1, DESCRIPTION: Mithril Saber, SLOT: hands, VALUE: 990>
DISCARD<GEAR, XLOC: 2, XLOC: 1, DESCRIPTION: Iron Leggings, SLOT: legs, VALUE: 250>
COLLECT<GEAR, XLOC: 2, XLOC: 1, DESCRIPTION: Mithril Leggings, SLOT: legs, VALUE: 950>
COLLECT<COINS, XLOC: 3, YLOC: 1, VALUE: 50>
```

3. Documentation

3.1. adventure.h

void explorer_init(explorer*)

The explorer_init function is used to allocate memory for and initialize an explorer struct. The coins, pos_x and pos_y fields are set to zero because the adventure always starts at the top left corner of the map (0,0) and the explorer starts with zero coins. An array of pointers to gear structs is initialized with a length of NUM_GEAR. NUM_GEAR is a macro that is defined as 4. The explorer can only have 4 pieces of gear at one time but if someone wanted to change this they could change the definition of NUM_GEAR. The items field is set to a pointer to an empty linked list using the createList function. A pointer to the new explorer is returned.

void explorer_add_item(explorer*, item*)

The explorer_add_item function is used to insert a pointer to an item struct into the passed explorers items linked list. The insertFirst function is used to do this.

void explorer_gear_compare(explorer*, gear*)

The explorer_gear_compare function is used to compare and add a new gear struct to the explorers equipment array. If the index in the equipment array that corresponds to the gears slot is NULL then the gear struct is added to the array at that index. If it is not NULL then the two gear structs are passed into the function pointed to by the passed gear structs compare function pointer. If the compare function returns 1 the gear struct currently stored in the array is freed and the passed gear struct is stored in it's place. Otherwise the passed struct is freed. Because this function is responsible for the logic of collected and discarding gear this function also calls the write_gear function.

void free_explorer(explorer*)

The free_explorer function is used to free all memory allocated for the passed explorer struct. The structs stored in the explorer struct are all freed by calling the corresponding free function for that struct.

void print_explorer(explorer*)

The function print_explorer is used to print the value of the coins field the sum of the values of gear structs stored in the equipment array and the sum of the values of all item structs in the items linked list of the passed explorer struct.

gear* gear_init(char*, enum gear_slot s, int v)

The gear_init function is used to allocate memory for and initialize a gear struct. The value of s is used to determine the function pointer that is stored in the struct.

void free_gear(gear*)

The free_gear function is used to free any memory allocated to store the passed gear struct.

void free_item(item*)

The free_item function is used to free any memory allocated to store the passed item struct.

move* move_init(int, char)

The move_init function is used to allocate memory for and initialize a new move struct and return a pointer to it.

char compare_head/chest/legs/hands

Compares the values of the two passed gear structs and returns 1 if the first is greater than the second otherwise returns 0.

Table of Contents