Curtin University – Department of Computing

Declaration of Originality

You must fill out, sign and submit this declaration to the lecturer/unit coordinator **<u>in person</u>**, or your assignment will receive zero marks. **<u>Please print clearly</u>**.

Family name:	ONG	Student ID:	19287368
Given name(s):	MING HANG		
Unit name:	Unix and C Programming	Unit ID:	COMP1000
Lecturer:	Mark Upston	Tutor:	Hugh Flanigan
Date of submission:	26/05/2019		

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Plagiarism and collusion may be detected manually or by means of tools (such as Turnitin).
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to check that any files I submit are readable and not corrupted.

Signature: ONG MING HANG Date: 26/05/2019

Table of Contents

Functions and their purposes

TreasureHunter.c Functions

void main (void)

This main function takes in command-line arguments and initialises other functions to read, collect and display the adventure. It is the core function of every other functions, if an error occurs, the adventure will not begin. If the number of command-line inputs matches the condition, it will begin to read the given files and initiate a linked list. Else prints an error message to the terminal.

FileIO.c Functions

int mapFormat (char*)

This function reformats the given map file into a more readable state, it opens the original file and reads character by character, within the condition while reading, for every comma or newline character, place a dash behind it. The loop will continuously print out every single character into a new formatted map file, returns an int depending if an error occurs.

int advRead (char*, LinkedList*)

This function receives an adventure file to be opened for reading and storing its contents into a generic linked list, in the while-loop condition, while it isn't at the end-of-file and stop isn't declared as "1", memory allocate a Direction struct and using fscanf() function to read into the struct variables. The variable isCorrect keeps an eye out for incorrect format returned by fscanf(), if isCorrect is "2" or the number of step is greater or equals to zero, insert into linked list, else stop the loop and free the unwanted content, printing out an error message to the terminal. Returns an int depending if an error occurs.

void mapRead (char*, LinkedList*)

This function is forced to take in the formatted map file if and only if both mapFormat() and advRead() returns a "1", which means both function performed smoothly without any error. At the start of the function, some variables are declared to prevent the conditional jump errors showing up in valgrind, two FILE pointer are declared, one for the actual reading and the other to receives the number of lines to validate the map. Within the condition, if the file isn't empty, read the first line to initialise row and col variables, isCorrect variable is given the int value returned by sscanf() function to check if the content is being read correctly, it is expected to return two. If isCorrect contains "2", proceed to allocate sufficient memory using row and col variables to build a two-dimensional array of Map struct, in which the struct variable stores a string. After that, grab the number of lines that the actual file has to validate if the lines conform to the specified structure given by the first line of the file, which contains the information about rows and columns. Validate if the row matches, if it fails, prints error messages to the terminal, else using a single for-loop and while loop to store the content into the struct variable inside the two-dimensional array of Map struct with the assistant of strtok() function with comma as the delimiter, successfully stores the correct contents. Once the storage is filled, pass into itemProcessor() function to convert it into a two-dimensional array of Treasure struct alongside with the information of rows and columns, with the filled direction linked list. Freeing memory and closing the streams once it is done.

int getNumLines(FILE*)

This function takes in a FILE pointer to get the actual number of lines/rows the file has, using a do-while loop, reading character by character, for every newline character, increment by one and return the int value.

void dirFree (void* data)

This function takes in a void pointer to be converted into Direction struct and it is then freed, it is called when the program is freeing the linked list completely.

LinkedList.c Functions

LinkedList* listCreate (void)

This function creates an empty linked list through memory allocation alongside with initialisation of variables located inside the linked list struct, returns the newly created generic linked list.

Node* getNewNode (void*)

This function creates an empty list node through memory allocation alongside with initialisation of variables located inside the list node struct and storing the data it was given, returns the newly created list node.

void insertFirst (void*, LinkedList*)

This functions adds a new node into an existing linked list structure, starts by getting the current head previous pointer to point at the new node and have the new node next pointer to point at the current head, then the list head pointer now points to the new node and the new node previous pointer points to nothing. This makes the new node as the new head of the list.

void removeLast (Linkedlist*)

This function uses the existing linked list, remove the last element and depends on three cases which are if the list is empty, list has only one element and list has more than one element. When called, if the list is empty just return NULL, else if there is only one element in the list, store the tail into temp, grabs the data from temp, makes the list head and tail points to NULL and frees temp which is the lost tail pointer. Else if there is more than one element in the list, store the tail into temp, grabs the data from temp, makes the list tail previous pointer to point at the previous element and set the current tail next pointer to remove the reference to previous tail, frees temp which is the lost tail pointer.

void freeLinkedList (LinkedList*, StructKiller)

This function grabs the head to set a condition for the while-loop, while there exist some data, make nextNode to point at the next element of node, frees the malloc'd data inside node and frees the node itself, node now points at the next element, continue loop until there is nothing left.

ItemProcessor.c Functions

void itemProcessor (Map**, int, int, LinkedList*)

This functions takes in the two-dimensional array of Map struct alongside with rows and columns that has been initialised, also takes in a filled linked list. Starts by allocating sufficient memory using the given rows and columns to construct a two-dimensional array of Treasure struct, once successfully allocated with the correct size, fill up the newly created two-dimensional array using sscanf() function with isCorrect to validate the number of item scanned correctly. Scanning the type of item so each item is correctly stored in-place, if the item being read is wrong, render it as incorrect. For each gear, store a function pointer into the Gear struct variable compareStr depending on the slot given by the string. It is then passed into movementProcessor() function to begin the real adventure process, where it moves the player through the two-dimensional array of Treasure struct to collect items one after another. Freeing memory once it is done.

char* capitalizer (char*)

This function checks each and every character of the given string, using a for-loop, iterates through the array of characters check if any of them are lowercase in terms of ASCII values, if any of the character is larger than 'Z', it is classified as a lowercase character, in ASCII table lowercase is usually higher than uppercase. Subtract by 32 to get the uppercase character and combined it into a new string, it is then returned to where it is called.

MovementProcessor.c Functions

void movementProcessor (Treasure**, int, int, LinkedList*)

The function uses the filled linked list to move the player, remove the last element from the list once to set the while-loop condition, for every element removed from the list, it checks if the instruction matches any of the conditions, once matched, using a for-loop to move the player one pace at a time and in every for-loop calls itemCollector() function to collect the items and stored inside the player inventory. If the step exceeds the boundary, it will stop the program and prints "STATUS: FAILED" to the terminal while freeing any unwanted memory. Else if AI has been defined through conditional compilation, for every step stepping out of map boundary, it redirects the step to stop at the edge instead thus preventing the program to access restricted memory. Once the instructions are fully fulfilled, if it is valid, proceed to call displayStats() to show the final statistic of the player with conditions.

void itemCollector (Treasure**, int, int, Equip*)

This function collects item to store into the player's inventory, it is called multiple times to move one pace at a time, if the item is a coin, store the coin value into the player's inventory and if LOG has been defined, prints the process to the terminal and write to "adventure.log" file. If the item is a magic, it behaves the same as when the player discovered coin. If the item is a gear, immediately calls the function pointer to compare current gear with a new gear, if the current gear is weaker than the new gear it is effectively discard. Else if the current gear is still stronger, nothing will happen, it also depends on slot as the function pointer contains four address to four compare functions.

void displayStats (Equip*, char*)

This function displays the final statistic to the terminal once the adventure ends, include but not limited to the status of the adventure, total value of gears, total value of magics and total value of coins.

void equipFree (void*)

This function takes in a void pointer to be converted into Equip struct and it is then freed, it is called when the adventure ends and the statistic displayed.

CompareGear.c Functions

void compareHead (Gear*, Equip*, int, int)

This function takes in a pointer to the new head equipment and a pointer to the current head equipment, alongside with the location of x and y on the map. If the slot of head equipment is empty, equip as it is by pointer to the new head equipment. Else if the current head equipment is weaker than the new head, replace the current head equipment with the new head equipment by pointing at the new head equipment. Nothing will occur if new head equipment is weaker than current head equipment. If LOG has been defined, prints the process and write it into “adventure.log” file.

void compareChest (Gear*, Equip*, int, int)

This function takes in a pointer to the new chest equipment and a pointer to the current chest equipment, alongside with the location of x and y on the map. If the slot of chest equipment is empty, equip as it is by pointer to the new chest equipment. Else if the current chest equipment is weaker than the new chest, replace the current chest equipment with the new chest equipment by pointing at the new chest equipment. Nothing will occur if new chest equipment is weaker than current chest equipment. If LOG has been defined, prints the process and write it into “adventure.log” file.

void compareHands (Gear*, Equip*, int, int)

This function takes in a pointer to the new hands equipment and a pointer to the current hands equipment, alongside with the location of x and y on the map. If the slot of hands equipment is empty, equip as it is by pointer to the new hands equipment. Else if the current hands equipment is weaker than the new hands, replace the current hands equipment with the new hands equipment by pointing at the new hands equipment. Nothing will occur if new hands equipment is weaker than current hands equipment. If LOG has been defined, prints the process and write it into “adventure.log” file.

void compareLegs (Gear*, Equip*, int, int)

This function takes in a pointer to the new legs equipment and a pointer to the current legs equipment, alongside with the location of x and y on the map. If the slot of legs equipment is empty, equip as it is by pointer to the new legs equipment. Else if the current legs equipment is weaker than the new legs, replace the current legs equipment with the new legs equipment by pointing at the new legs equipment. Nothing will occur if new legs equipment is weaker than current legs equipment. If LOG has been defined, prints the process and write it into “adventure.log” file.

LogWriter.c Functions

void writeFile (char*)

This function opens a FILE pointer to write into the file without overwriting the file, takes in a single string to be written into the file using fprintf() function, closes the stream afterwards.

Input to Coordinate Conversion

Input file conversion are made into two different function, one for map and one for direction. The map conversion first reformats the file into a more readable state, adding dashes for every comma and newline character, by doing this, the formatted map can effectively be treated as a valid map in comparison to the original map. The adventure function will read in each line into a generic linked list, by storing the instructions into a linked list, it can be treated as a queue abstract data type, it is noted that each node is processed using fscanf() function which takes in a string and an int which is then inserted first into the list. Since the linked list is known for not having any limitation of how much it can store, instructions can be written indefinitely. Both function returns a status, which is an int, if the status return 1, it means that no error occurred and everything went smoothly, else if it return 0, it means an error occurred and the real map conversion function will not be called. To call the third function, which is the real map conversion, both function must return 1 consecutively.

The third function is responsible converting the valid file content into a coordinate system in the structure of a two dimensional array, or a map grid. This function is forced to receive the formatted file written by the first map conversion function so the player could not control if the file were to be invalid. It first reads the first line of the formatted file, detecting if there exist two ints that represents rows and columns, if it exist, dynamically allocate enough memory for the two dimensional array, else throws an error message. The rows and columns are only defined by the given file, if the two variables does not conform the real map size, it will be effectively discarded. Else it would, using a string tokenizer function like strtok() with the delimiter "," or comma, read what is in between the delimiter regardless of string length as the program should also consider empty spaces. Once the two-dimensional array is filled, it is then passed into a function that converts it from a string two-dimensional array to a struct two-dimensional array which actually stores the information correctly. Once all of it is done, it will proceed to the last and final function that is responsible for the coordinate system. The instruction is passed alongside with the two-dimensional struct array, rows and columns to move the character around the map grid. Instructions are then remove in FILO order, or first-in-last-out from the linked list to move the player around the map grind.

Alternatively, instead of using a map formatter to convert the file character by character, it is possible to using strstr() to achieve the same outcome without writing another function for it but the implementation is currently unknown. Another implementation would be just using fgets(), although it would be unusual to describe how it works but it is still possible to achieve the same outcome. Lastly, writing a completely new tokenizing method that is designed specifically for this map conversion is another way to convert it.

Demonstration

Map file: map.csv

Adventure file: adventure.csv, intothevoidyougo.csv (for AI)

Contents of map.csv as follow:

```
5,4
START,,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 as follow:

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

Contents of intothevoidyougo.csv

```
DOWN 2
RIGHT 2
DOWN 2
LEFT 1
UP 3
RIGHT 2
...
DOWN 2
LEFT 1
UP 3
RIGHT 2 <--- Line number 136
```

Command-Line Input and Output of Normal mode

Input line: `./TreasureHunter map.csv adventure.csv`

Output:

```
Welcome to Treasure Hunter!
Select one of the option:
1. Start game
2. Exit
User input: 1
Game started!
STATUS: COMPLETE
GEAR: 3940
MAGIC: 445
COIN: 50
Select one of the option:
1. Start game
2. Exit
User input: 2
...Exiting Treasure Hunter...
```

Command-Line Input and Output of Log mode

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

Output:

```
Welcome to Treasure Hunter!
Select one of the option:
1. Start game
2. Exit
User input: 1
Game started!
```

```
COLLECT<ITEM:GEAR, XLOC:0, YLOC:1, DESCRIPTION:Iron Chestplate,
SLOT:CHEST, VALUE:200>
COLLECT<ITEM:MAGIC, XLOC:0, YLOC:2, DESCRIPTION:Healing Potion,
VALUE:85>
COLLECT<ITEM:MAGIC,    XLOC:1,    YLOC:2,    DESCRIPTION:Defence
Enchantment, VALUE:360>
COLLECT<ITEM:GEAR,  XLOC:2,  YLOC:2,  DESCRIPTION:Iron  Helmet,
SLOT:HEAD, VALUE:250>
COLLECT<ITEM:GEAR,    XLOC:2,    YLOC:3,    DESCRIPTION:Mithril
Chestplate, SLOT:CHEST, VALUE:1050>
DISCARD<ITEM:GEAR, XLOC:2, YLOC:3, DESCRIPTION:Iron Chestplate,
SLOT:CHEST, VALUE:200>
COLLECT<ITEM:GEAR,  XLOC:2,  YLOC:4,  DESCRIPTION:Rune  Saber,
SLOT:HANDS, VALUE:850>
COLLECT<ITEM:GEAR, XLOC:1, YLOC:4, DESCRIPTION:Mithril Helmet,
SLOT:HEAD, VALUE:950>
DISCARD<ITEM:GEAR,  XLOC:1,  YLOC:4,  DESCRIPTION:Iron  Helmet,
SLOT:HEAD, VALUE:250>
COLLECT<ITEM:GEAR,  XLOC:1,  YLOC:3,  DESCRIPTION:Iron  Leggings,
SLOT:LEGS, VALUE:250>
COLLECT<ITEM:GEAR,  XLOC:1,  YLOC:1,  DESCRIPTION:Mithril  Saber,
SLOT:HANDS, VALUE:990>
DISCARD<ITEM:GEAR,  XLOC:1,  YLOC:1,  DESCRIPTION:Rune  Saber,
SLOT:HANDS, VALUE:850>
COLLECT<ITEM:GEAR, XLOC:2, YLOC:1, DESCRIPTION:Mithril Leggings,
SLOT:LEGS, VALUE:950>
DISCARD<ITEM:GEAR,  XLOC:2,  YLOC:1,  DESCRIPTION:Iron  Leggings,
SLOT:LEGS, VALUE:250>
COLLECT<ITEM:COINS, XLOC:3, YLOC:1, VALUE:50>
STATUS: COMPLETE
GEAR: 3940
MAGIC: 445
COIN: 50
Select one of the option:
1. Start game
2. Exit
User input: 2
...Exiting Treasure Hunter...
```

Command-Line Input and Output of AI mode

```
./TreasureHunter map.csv intothevoidyougo.csv
```

Output:

```
Welcome to Treasure Hunter!
Select one of the option:
1. Start game
2. Exit
User input: 1
Game started!
STATUS: CORRECTED
GEAR: 3940
MAGIC: 445
COIN: 50
Select one of the option:
1. Start game
2. Exit
User input: 2
...Exiting Treasure Hunter...
```

Inside adventure.log:

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

Known Issues and Bugs

When reading in a map with invalid format\, the program still ignores the validation and attempts to access memory not within its boundary, resulting in a segmentation fault.

When reading in an adventure with invalid format, the program still ignores the validation and attempts to read it anyways, it doesn't cause any memory leaks but it displays conditional jump errors.

The validation of columns has not yet been implemented due to the lack of time, so it will only validate rows for now.