Adventureland Conversion and Translation

By: Naomi Burhoe

1. Introduction

Adventureland is a text-based game set in a command line-based environment that challenges players to explore a world rampant with magical creatures and many dangers in order to collect a variety of treasures. The version of the game I used was first written in C by Morten Lohre and my task was to convert as much as I could from C to C++ format. In order to do this, I studied object-oriented concepts such as inheritance through classes as well as more general features such as the use of enumerations. In the end, much was learned from this experience but some ideas were altered in order to incorporate them into the design.

2. Process

Before making changes to the program, I found it necessary to analyze the structure of it in order to determine exactly how it functioned. Due to the fact that the C translator made little use of commentary, this was definitely a challenge. Thus, my first order of business was to add as much commentary as necessary for me to understand exactly what was occurring in each phase and function of the program. Then, I separated some aspects of the program into multiple files. Finally, I added several abstraction elements to add clarity to the program.

3. Generally Unchanged Functionality

The user interface remains mostly unchanged from what it was before I began working on the program. In addition many functions implemented by the previous programmer retain the same logical structure. Such functions include:

```
void empty_keyboardbuffer()
int yes_no()
void welcome()
void look()
void turn()
void action(int, int *)
int get_input();
int get_item_string(int)
int get_action_variable(int *, int)
void carry_drop()
int length(const char *)
void copystring(char *, const char *)
int comparestring(const char *, const char *)
int check_logics()
```

Many functions did, however, require modification in order to allow for interaction with added abstractions. In addition, the get_input() function required modification in order to read in input from the console/user.

4. Abstractions:

a. Location enumeration

The Location enumeration consists of 34 different locations that are used throughout the game, represented by integers from -1 to 33.

Inventory = -1,
Unassigned = 0,
Swamp = 1,
Tree = 2,
Stump = 3,
Root = 4,
Hole = 5,
Hall = 6,
Cavern = 7,
EightRoom = 8,
Anteroom = 9,
Shore = 10,
Forest = 11,
Maze = 12-17,
Chasm = 18,
Ledge = 19,
RoyalChamber = 20,
LedgeThrone = 21,
ThroneRoom = 22,
Meadow = 23,
Trouble = 24,
Grove = 25,
Bog = 26,
PC = 27,
Branch = 28,
Empty = 29-32,
MistRoom = 33

The Empty locations may correspond to rooms found in other implementations of the Adventureland game.
The Location enumeration is used in both the Item and Room classes.

b. Item class
The Item class consists of a Location element and a description string. Functionality allows for == and != comparison, construction, copy construction, and viewing but not modification of elements.

```
class Item {
public :
        Item();
        Item(Location l, std::string d);

         Location getLocate() {return il; }
        void setLocate(Location);
        std::string getDescrip() {return id; }

        void operator=(const Item&);

        friend bool operator==(Item, Item);
        friend bool operator!=(Item, Item);

        ~Item();
private :
         Location il; // item location
         std::string id; // item description
};
```

c. Room class

The Room class consists of a Location element, description string, and a vector of directions that can be traveled in based on the Location. Functionality allows for == and != comparison, construction, copy construction, and viewing but not modification of elements.

```
class Room {
public :
             Room();
             Room(Location, std::string, std::vector<Location>);

             Location getRL() { return rl; }
             std::string getRD() { return rd; }
             std::vector<Location> getDirect() { return direct; }
             void operator=(const Room&);
            friend bool operator==(Room, Room);
            friend bool operator!=(Room, Room);

            ~Room();

private :
            Location rl; // room location
            std::string rd; // room description
            std::vector<Location> direct; // directions available
};
```

5. Other Additions and Modifications to Functions

a. Sleep() function

The Sleep function uses a clock/timer in order to slow the progress of welcome function printing. Before it was added, the user was not able to read game-play directions. This function was borrowed from the Ubuntu forums.

```
void Sleep ( int seconds )
{
             clock_t endwait;
             endwait = clock () + seconds * CLOCKS_PER_SEC ;
            while (clock() < endwait) {}
}
```

b. convertInt() function

The convertInt() functions returns a Location value based on a received integer value. The corresponding integer values are denoted in part 4.a of this article.

c. copystring() and comparestring() functions

These functions required modification of preconditions in order to allow for comparison of c_strings and std:strings. Thus, there are currently two versions of each function with the exact same logic for each. I could have simply changed compared std::strings to c_strings but I did not think of that before finishing the project.

6. File Separation

a. main.cpp

This file holds the major components of the game. All interaction and logic functions are defined here, except where otherwise specified. I attempted to separate the function definitions from the main program, but I gave up momentarily in order to work on other aspects of the game.

b. advland.h

This files only holds the Location enumeration. This is so it can interact with components of the game such as objects of types Item and Room.

c. advland.hpp

This files holds declarations of global variables and game-play functions.

I. Global Variables:

```
#define CL      151+1
#define NL      59+1
#define RL      33+1   // room limit
#define DL      6+1    // direction limit
#define ML      71+1   // message limit
#define AR      Forest     // start location
#define IL      60+1   // number of items (item limit)
#define MX      5      // max number of items allowed to carry
#define TT      13     // number of treasures
#define LN      3      // number of characters in commands in item strings
#define LT      125    // number of steps before light goes out
#define TR      Stump      // treasure depository location
#define MAXLINE 79     // max number of characters on one line

extern const unsigned int C[CL][8]
```

```
// NV$(59,1) commands
extern const char *NVS[2][NL];

// RM(33) rooms with location, description and travel
extern Room RM[RL];

// items(60) items with location and description
extern Item items[IL];

// MS$(71) messages
extern const char *MSS[ML];
```

II. Game-play functions

Game-play functions are specified in parts 3 and 5.

d. advland.cpp

This file holds definitions of global variables, which are designated by the keyword *extern.*

e. item.hpp

This file contains the Item class declaration. Details about the Item class can be found in part 4.b.

f. item.cpp

This file holds the implementation details of the Item class. Details about the Item class can be found in part 4.b.

g. room.hpp

This file contains the Room class declaration. Details about the Room class can be found in part 4.c.

h. room.cpp

This files holds the implementation details of the Room class. Details about the Room class can be found in part 4.c.

7. Conclusion

The Adventureland translation project required a lot of internal changes in order to begin the transition from programming in C to C++. Abstractions were added, language was updated, and formulas were modified in order to incorporate more modern programming techniques, such as object-oriented programming. While more could have been done in order to further expand the scope of the project, what was successfully done helped to build the knowledge of the student.