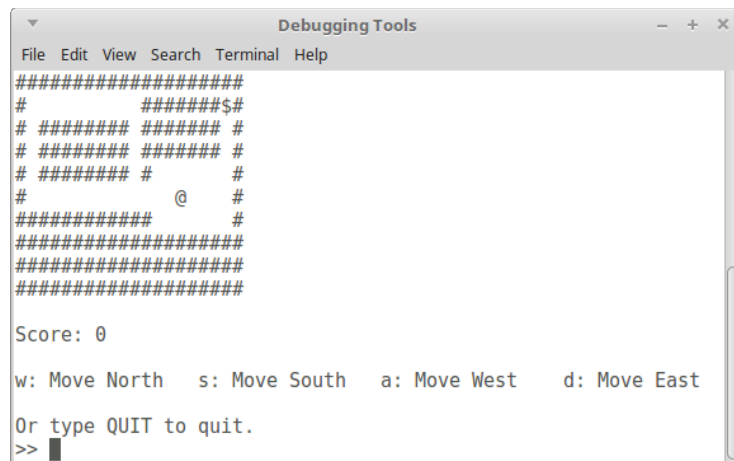


Lab: Debugging Tools

About

For this lab, download the example code. As you follow this document, it will step through how to use the debugging tools in Visual Studio. Write in your answers for the questions.

Introduction: Play the game



```
Debugging Tools
File Edit View Search Terminal Help
#####
#          #####$#
# #####  ##### #
# #####  ##### #
# #####  #      #
#          @      #
#####
#####
#####
#####

Score: 0

w: Move North   s: Move South   a: Move West   d: Move East

Or type QUIT to quit.
>>
```

First start up the game and try it out – see how it plays, notice the commands. You’re the @ symbol and your goal is to collect the \$ symbol.

Task 1: MovePlayer function – Variable values

Set a breakpoint on line 246, the first line in the MovePlayer function.

Run the program, and choose “1” to start a new game. Notice where your player is (the @ symbol), and try to walk to an empty space (# are walls, use W, A, S, or D and then ENTER to move.) The program will pause after you hit ENTER and you can view the MovePlayer function. It should be stopped at this line:

```
char firstLetter = direction[0];
```

In the locals, auto, or watch window, locate the following variables and fill in their values:

direction	
firstLetter (will be “default” until you move to next line)	
player	
board	

The player and board parameters are passed by reference. You may have a hexadecimal value, which is a memory address.

Next, click on the “Step over” button to move to the next line of code. The program execution should be stopped at this line:

```
firstLetter = tolower( firstLetter );
```

Now record the following value:

firstLetter	
-------------	--

- How has `firstLetter` changed?

Next, hover your mouse over the `MOVE_UP` named constant, it should tell you its value. Also type its name in the “Watches” window. Fill out the values for each of these named constants:

<code>MOVE_UP</code>	
<code>MOVE_DOWN</code>	
<code>MOVE_LEFT</code>	
<code>MOVE_RIGHT</code>	

Click on “Step into” several times until you enter one of these:

- `if (firstLetter == MOVE_UP)`
- `else if (firstLetter == MOVE_DOWN)`
- `else if (firstLetter == MOVE_RIGHT)`
- `else if (firstLetter == MOVE_LEFT)`

In your auto/locals/watch window, the `player` variable should be expandable. Expand it to view its member variables. Fill out all of the member variables of the struct, and its current values.

Member variable name	Value

Keep using “Step into” to move into the internal statement, where the `x` or `y` variable of the `player` struct is being changed. When the pause cursor is *on* the line of code, it has paused prior to its execution. In the auto/locals/watch window, watch the value of `x` or `y` change once you step over that line of code.

Once you’re done, clear this breakpoint.

Task 2: YouWin function – Step Into

Next, set a breakpoint at line 280, inside the `YouWin()` function. This function is called once the `@` character’s (`x`, `y`) coordinates matches the (`x`, `y`) coordinates of the `$` character.

Run the program and navigate your character to the `$`. Then the program execution should pause.

Step to the `ClearScreen()` function, then use the “Step into” button. It will move the pause cursor into this function.

- As you step through this function, which line of code is called: `system("cls");` or `system("clear");` ?

Click on “Step over” several times until the end of the `ClearScreen()` function. It will return back to the `YouWin()` function.

Continue using the “Step into” button through the `cout` statements. Once you get to this line of code:

```
string input = GetStringInput();
```

it should then step into the `GetStringInput()` function.

Continue clicking “Step into” as you go through the `GetStringInput()` function. Once you move past the `cin` statement, return to the program window – it’s expecting some input. Type something and hit enter, and then the program execution will pause again. Continue stepping thru past the `return val;` statement and back to the `YouWin` function.

Finally, press the “Continue” button. The program will keep executing until the next time you finish a stage of the game.

Task 3: Call stack

There is a debug pane that will tell you which functions has been called to reach the point where you are at.

Set a breakpoint at line 178, in the `GenerateMap(...)` function. Run the program and choose Play. The program will pause. Log the following, using the Call Stack:

1. What was the function that called `GenerateMap()` ?
2. What was the function that called the function that called `GenerateMap()` ?