

Professor Carl Eyer

SDEV425 6381: Mitigating Software Vulnerabilities

Homework 3

23 February 2024

University of Maryland Global Campus

This document will list and analyze the security issues that I will encounter in the C code given to me for Homework 3 of my SDEV425 course. I will be analyzing these issues based on materials from Weeks 5 and 6 of the course. Particularly, the CERT Coding Standard Recommendations. As instructed, before I can dive into figuring out the program's issues, I must demonstrate the proper functionality of my C environment. I decided to do this by showing a screenshot of my "Hello, World!" C program, which I designed in Week 5, with it running correctly.

### Environment Test:

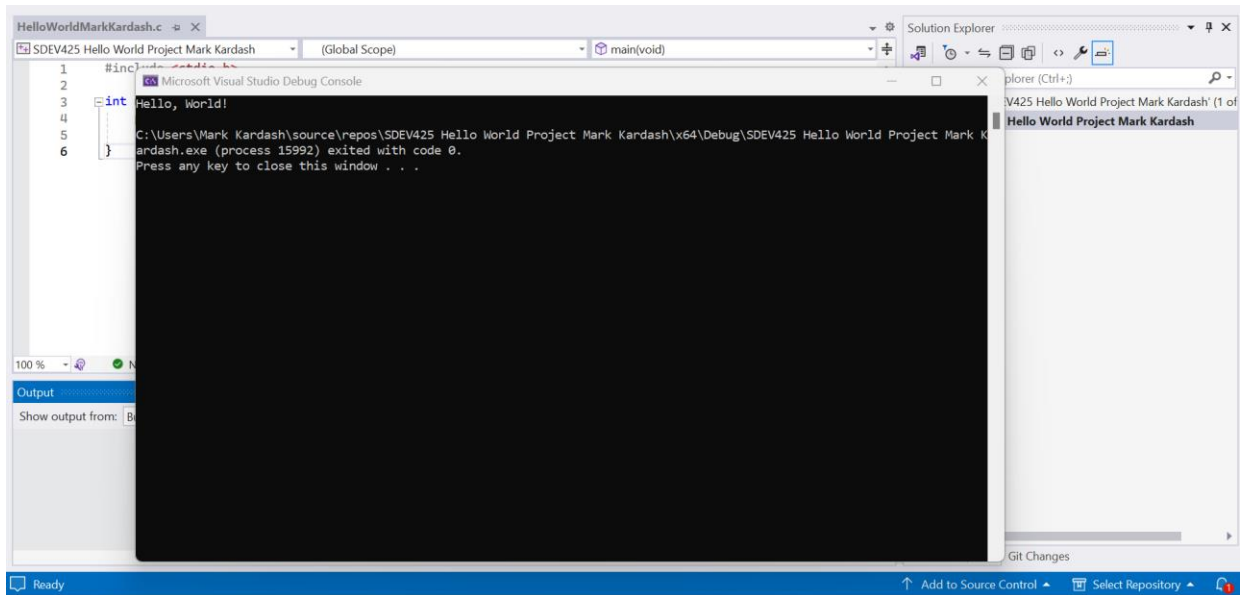


Figure 1: Environment Test

As can be seen in the figure above, the "Hello, World!" program from Week 5 is running successfully, proving the correct functionality of my environment.

I must first perform testing as to whether it works correctly. The test cases for this are described below. They will help me determine exactly what areas of the application don't function correctly.

Testing the Original Program:

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
1.	B, f, S, e	"Welcome to the Baseball Season" "Welcome to the Football Season" "Welcome to the Soccer Season" "Exiting the Menu system"	Each input causes the entire menu to be printed twice, with the user being asked for the next input. Only entering "e" causes the program to close correctly.	Fail
2.	"b", F, s, E (Ignore quotations on b. They are to signify that it is	"Welcome to the Baseball Season" "Welcome to the Football Season"	Each input causes the entire menu to be printed twice, with the user	Fail

	lowercase.  Otherwise  Microsoft Word  turns it into  uppercase).	“Welcome to the Soccer Season”  “Exiting the Menu system”	being asked for the next input.  Only entering “E” causes the program to close correctly.	
3.	Something	“Please enter a valid selection.”  Menu reprinted	The message “Welcome to the Soccer Season!” appears, after which the main menu, and “Enter a valid selection”, is printed four times. The program then exits and prints the password (A series of 1s).	Fail
4.	Hi	“Please enter a valid selection.”	The message “Please enter a	Fail

		Menu reprinted	valid selection:” appears, after which the menu is printed three times. The user is again expected to enter a selection.	
5.	Base	“Please enter a valid selection:” Menu reprinted	Although “Welcome to the Baseball Season” message appears, followed by the menu printed four times, and the program exiting with the exit message, and printing out the password.	Fail

```
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
B
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Welcome to the Football season!
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
```

Figure 2: Test 1 Output Part 1

Almost every selection results in the menu being printed twice.

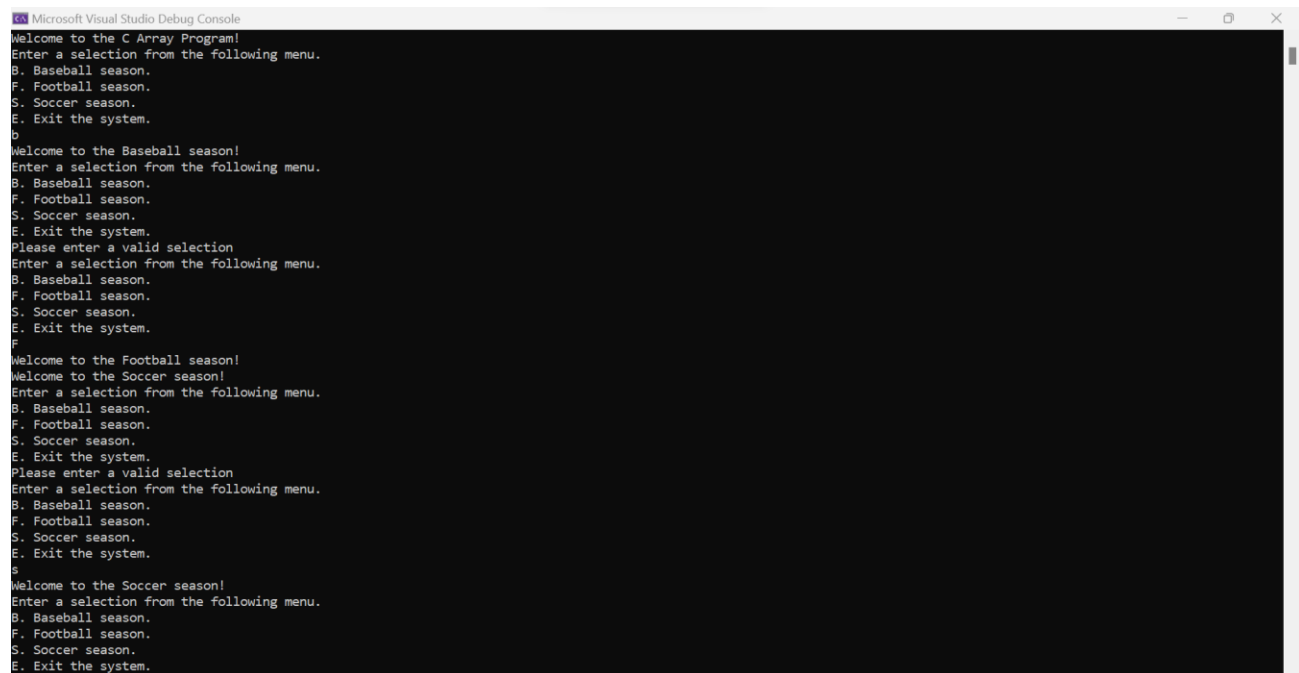
```

Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!
C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 4460) exited with code 0.
Press any key to close this window . . .

```

Figure 3: Test 1 Output Part 2

Only inputting the letter “e” results in the intended output, printing out the password and the exit message.



```

Microsoft Visual Studio Debug Console
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
b
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
F
Welcome to the Football season!
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
s
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.

```

Figure 4: Test 2 Output Part 1

As with the previous test, almost every selection results in the main menu being printed twice.

```

Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
E
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!
C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 16828) exited with code 0.
Press any key to close this window . . .

```

Figure 5: Test 2 Output Part 2

Only entering “E” gives the desired result.

```

Microsoft Visual Studio Debug Console
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
something
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!
C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 21368) exited with code 0.
Press any key to close this window . . .

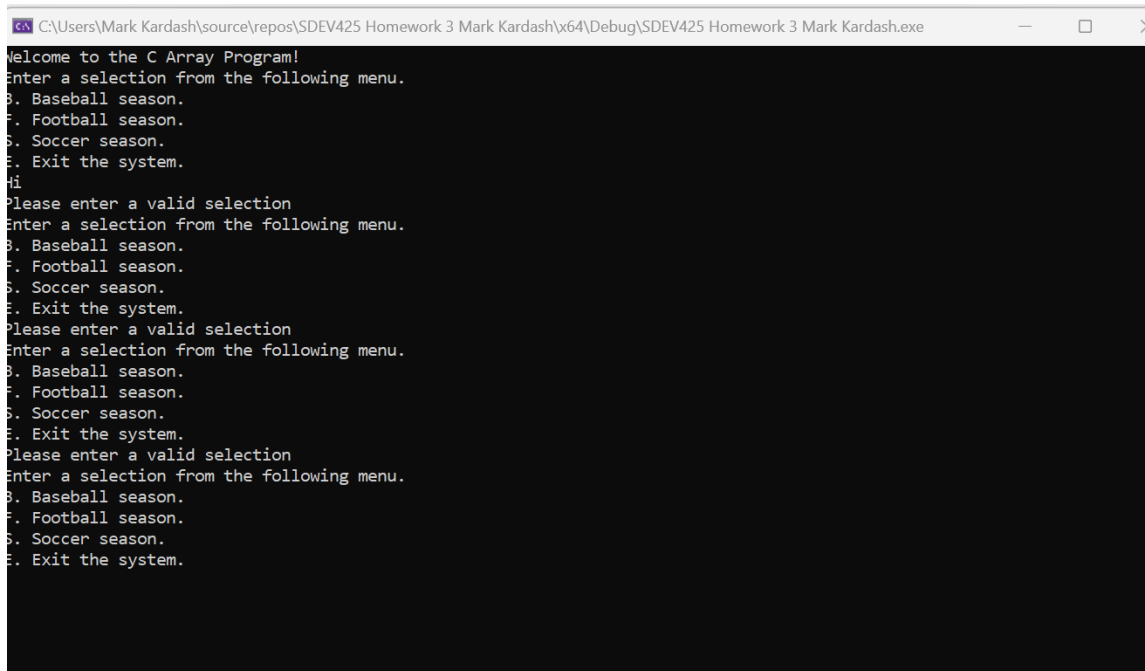
```

Figure 6: Test 3 Output

Entering the word “something” as input resulted in a printing out of the message “Welcome to the Soccer Season”, followed by the main menu, and “Please enter a valid selection”, being printed four times. After this, the program exits, printing out the password. My guess is that the



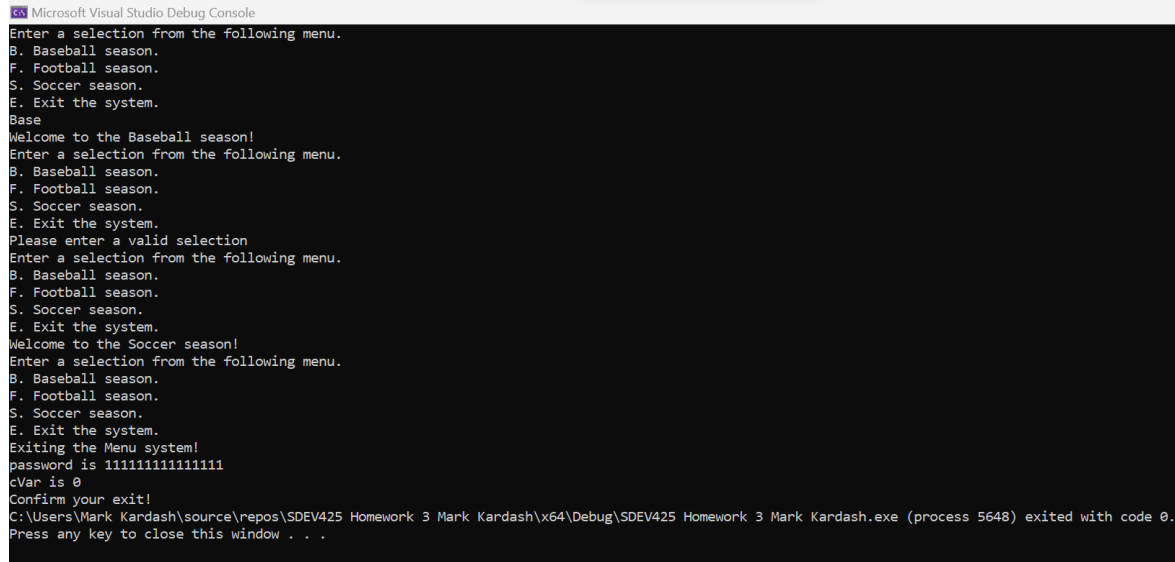
program was triggered to display the “Welcome to the Soccer Season” message because the word “something” contains the letter “s”, which the program somehow misinterpreted as a valid input.



```
C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Hi
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
```

Figure 7: Test 4 Output

Although entering “Hi” as an input does result in the expected “Please enter a valid selection” message, the menu is then printed three times, and the program still asks the user for input at the end, just like it normally would in the beginning.



```

Microsoft Visual Studio Debug Console
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Base
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!
C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 5648) exited with code 0.
Press any key to close this window . . .

```

Figure 8: Test 5 Output

I wanted to test my theory that messages in the program are somehow triggered by “correct” letters within an incorrect input. To do this, I inputted a word that begins with the letter “B” (“Base”) this time intentionally uppercase. As I expected, this triggered the “Welcome to the Baseball Season!” message to be displayed. However, similar to Test 3, it also caused the message “Please enter a valid selection”, and the main menu, to be printed four times, after which the program exited, printing an exit message, and the password.

### Repairing the Code:

The very first thing that seemed off to me when I looked at the code was that the “case” statements were in perfect horizontal alignment with the “switch” statement they are a part of. I figured this should not be the case, and that they should be slightly indented. I confirmed this for myself when I went to the immediate next line under the “switch” statement, and saw that an indent was immediately made by the program. I suspected that the incorrectly indented “case” statements were the cause of most issues in the code. However, I wanted to back my assumptions

with evidence, so I went looking over the CERT rules and recommendations. One of the things that made the task very difficult for me was that I have almost never dealt with the C language prior to this class, focusing mostly on Java and Python. Thus, I was working somehow in the blind here.

After a brief glance, I did not see any CERT recommendations that had to do with indentation. However, I still decided to move the “case” statements into what I thought their proper position were, and then test the application, using the same test cases from earlier, to see how it would respond.

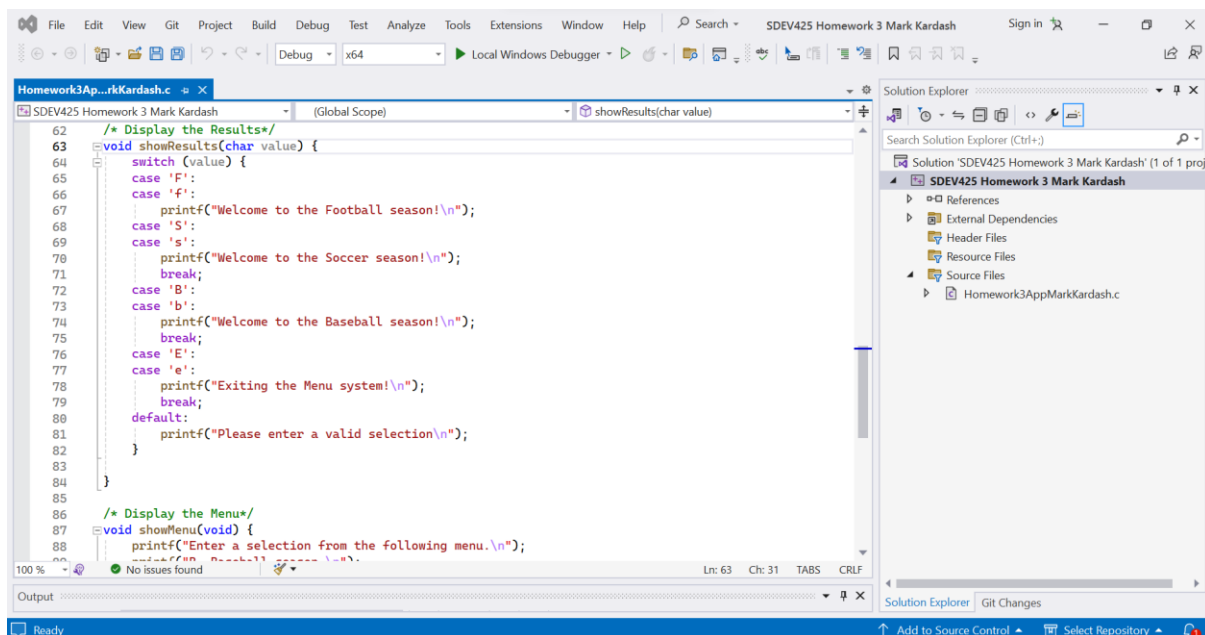


Figure 9: Original Position of Switch statement cases prior to any changes

As can be seen in the figure above, all the cases are directly underneath each other, and, in addition to this, directly underneath their parent “Switch” statement. With this indentation, the program probably does not treat them as part of the statement, causing the inputs to give incorrect outputs.

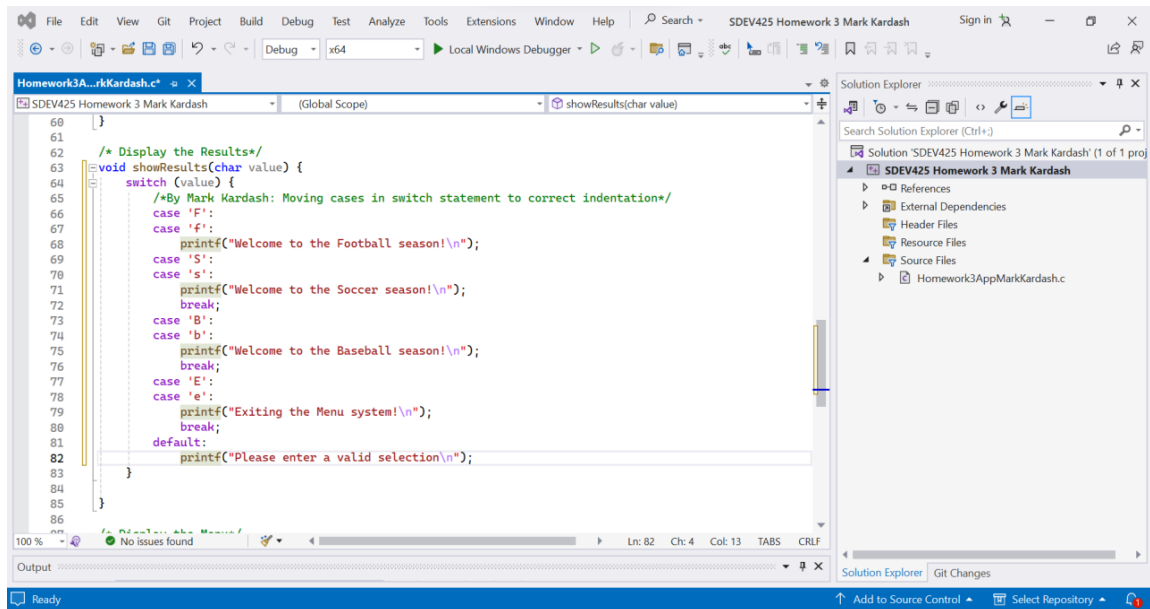


Figure 10: Cases of switch statement after indentation

Here, I moved all the cases in the “Switch” statement slightly to the right, so that they would not be aligned with the switch statement, but rather, slightly “embraced” by it. I did this in hopes of fixing the problem of incorrect outputs. I will now test my solution using the first out of the five previous test cases, to see whether anything has changed.

#### Repaired Code Test Case:

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
1.	B, f, S, e	<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p>	<p>While each letter does trigger the correct</p> <p>“Welcome to” message, this is</p>	Fail

		<p>“Welcome to the Soccer Season”</p> <p>“Exiting the Menu system”</p>	<p>immediately followed by the menu being printed two times, along with the “Please enter a valid selection” message, the exit message, and the password.</p>	
--	--	--	---	--

```

Microsoft Visual Studio Debug Console
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
B
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Welcome to the Football season!
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S

```

Figure 11: Test Output with First Changes Part 1

As can be seen, the test output is pretty much the same as it was before any changes, with the menu being printed after each correct option.

```

Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!
C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 6620) exited with code 0.
Press any key to close this window . . .

```

Figure 12: Test Output with First Changes Part 2

And just as before, only the letter “e” gives the desired result.

The constant re-printing of the menu caused me to surmise that there may be something wrong with the placement of the menu itself within the code. So I went back to reviewing it. Eventually, I realized there was some kind of a placement issue with all the program’s messages, since they were all printed together, no matter the variable I selected. It was then that an idea occurred to me. In a somewhat risky move, I decided to move the entire “Switch” statement out of the “showResults” method, and into the main method. This also involved replacing the “value” variable in the “Switch” statement with “cont”, which was the variable for user selection input.

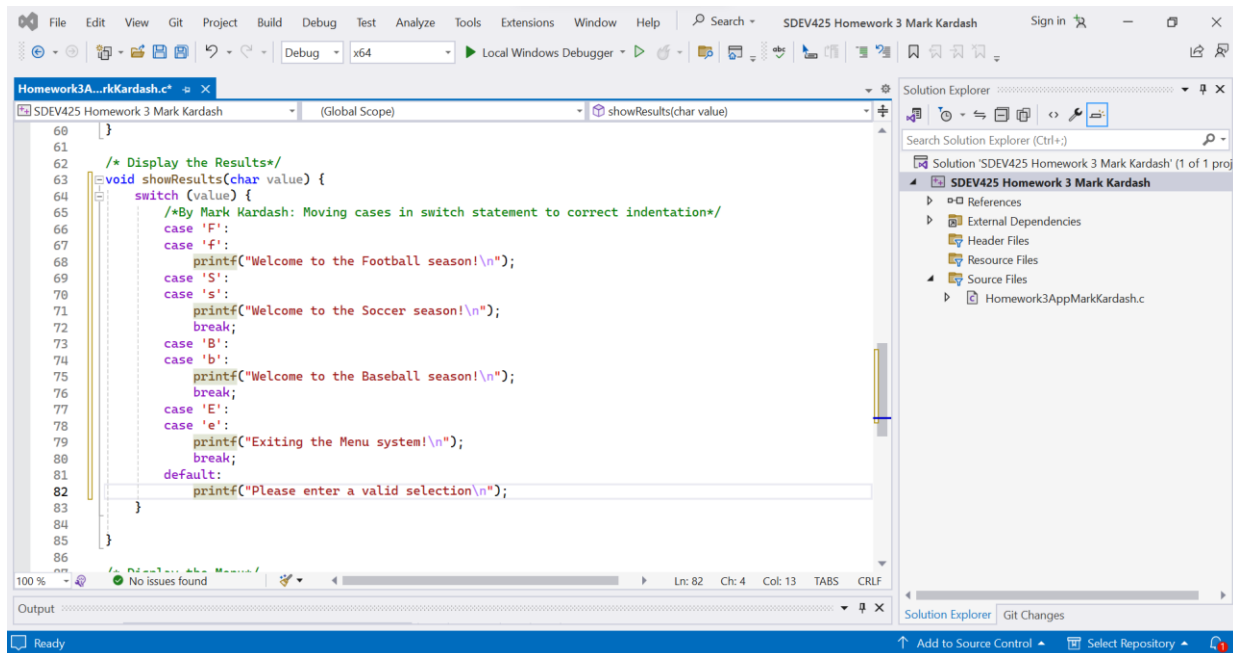


Figure 13 (Same as Figure 10): Original Placement of Switch Statement

Originally, the Switch statement was placed in a separate method called “showResults”. This made it quite unclear what exactly the “value” variable was that the cases depended on.

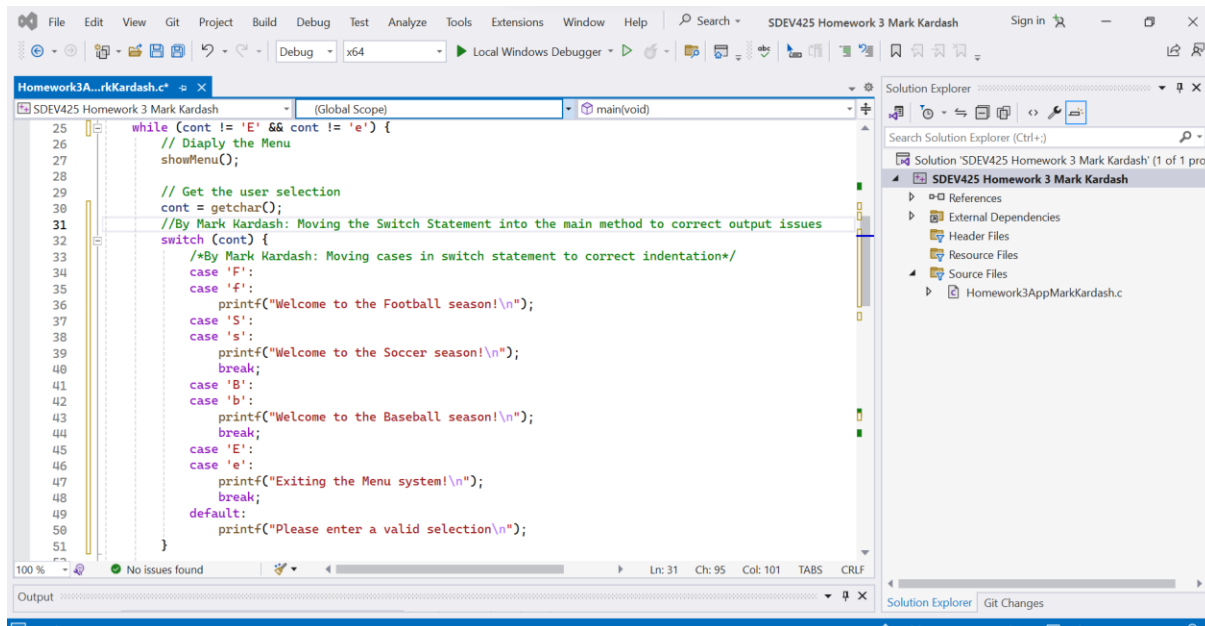


Figure 14: New Placement of Switch Statement

With the Switch statement inside the main method, and “cont” having been placed as a variable for it, I hope that the program will now base its actions on whichever value “cont” receives, generating the correct output.

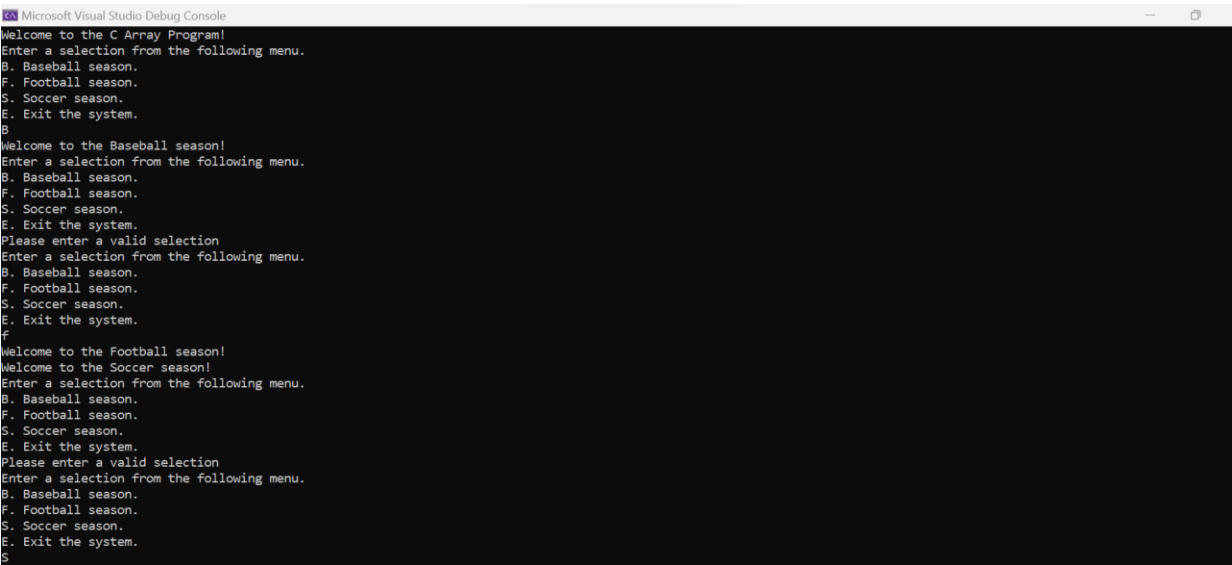
Now, as always, it is time to test the program once again. For simplicity, I will be using the same test case.

Repaired Code Test 2:

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
1.	B, f, S, e	<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p> <p>“Welcome to the Soccer Season”</p> <p>“Exiting the Menu system”</p>	<p>The program does not seem to have been affected in the slightest by the changes, printing the main menu twice, along with the default selection statement, and the exit message, with the exit</p>	Fail



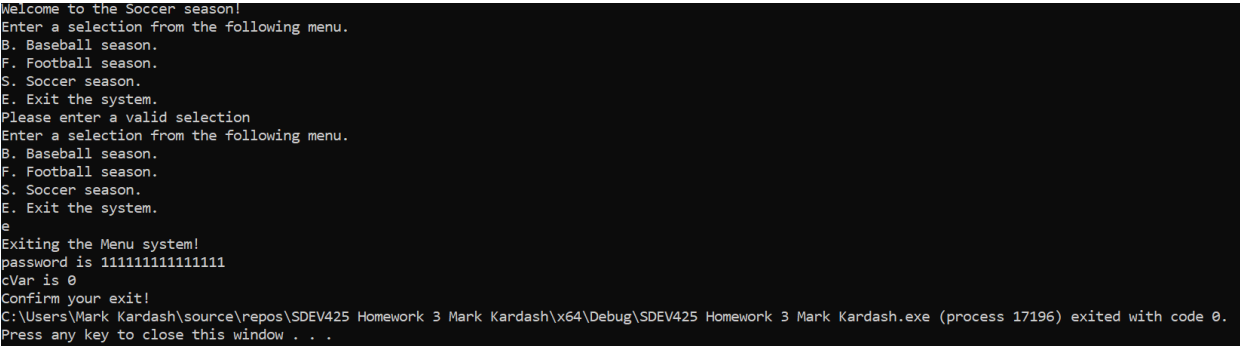
			option being the only valid one.	
--	--	--	-------------------------------------	--



```
Microsoft Visual Studio Debug Console
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
B
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Welcome to the Football season!
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
```

Figure 15: Repaired Code Test 2 Output Part 1

The program produces the exact same output as before, along with the “Please enter a valid selection” message.



```
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!
C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 17196) exited with code 0.
Press any key to close this window . . .
```

Figure 16: Repaired Code Test 2 Output Part 2

The letter “e” is the only one to produce the desired output.

Since the changes had no effect, I decided to move the “Switch” statement back to the “showResults” method.

After repeatedly trying and failing to fix the code, I decided to focus my attention away from the case statements themselves, and on the “If” loop that handles the menu. Following a great struggle, and a lot of fiddling with the code, I finally realized the problem did not lie quite there either. The answer was that the program was not equipped to handle the transition to a new line. Each time I enter a selection, the program is supposed to handle the inputted character, then transition to a new line, and print the expected output. However, I realized that the “\n” at the end of each statement was treated by the program as a separate character, rather than a command to change lines.

```
case 'F':
case 'f':
    printf("Welcome to the Football season!\n");
case 'S':
case 's':
    printf("Welcome to the Soccer season!\n");
    break;
case 'B':
case 'b':
    printf("Welcome to the Baseball season!\n");
    break;
case 'E':
case 'e':
    printf("Exiting the Menu system!\n");
    break;
```

Figure 17: Switch statement cases without changes

Because of this, when I, for example, entered “B” as a selection, the program saw it as me entering two characters, “B” and “\n”. This explains all the errors I was getting:

1. Since the program “thought” that it was receiving two separate characters, it printed the menu two times.
2. Since the second character, “\n”, was not among the valid ones, it triggered the default “Please enter a valid selection” message.
3. Since entering “e” or “E” exits the program immediately, the second input (“/n”) is discarded, making this the only selection that works correctly.

```

        printf("Welcome to the Soccer season!\n");
        break;
    case 'B':
    case 'b':
        printf("Welcome to the Baseball season!\n");
        break;
    case 'E':
    case 'e':
        printf("Exiting the Menu system!\n");
        break;
    default:
        printf("Please enter a valid selection\n");
    }
    //By Mark Kardash: Adding a "while" statement that will handle transitions to next line.
    while (getchar() != '\n');
}

```

Figure 18: New “While” statement to handle line transitions.

To make the program work correctly, I decided to place a “while” statement at the end of the Switch statement, just before the end of the “showResults” method. It will make sure that the menu and other statements will not get printed out if the character inputted is ‘\n’. Thus, the program will now print only the output that is expected based on the inputted letters. So without further ado, I began testing the program once again, this time using every single one of the previous test cases.

### Repaired Code Test 3:

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
1.	B, f, S, e	<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p> <p>“Welcome to the Soccer Season”</p> <p>“Exiting the Menu system” (Directory printed after 0 is entered)</p>	<p>Initial:</p> <p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p> <p><b>“Welcome to the Soccer Season!”</b></p> <p><b>“Welcome to the Soccer Season”</b></p> <p>“Exiting the Menu system” (Directory printed after 0 is entered)</p> <p>Final:</p>	<p>Fail (Initial)</p> <p>Pass (Final) (See explanation below)</p>

			<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p> <p>“Welcome to the Soccer Season”</p> <p>“Exiting the Menu system” (Directory printed after 0 is entered)</p>	
2.	<p>“b”, F, s, E (Ignore quotations on b. They are to signify that it is lowercase. Otherwise Microsoft Word</p>	<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p> <p>“Welcome to the Soccer Season”</p>	<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p> <p>“Welcome to the Soccer Season”</p>	Pass

	turns it into uppercase).	“Exiting the Menu system” Directory printed after 0 is entered.	“Exiting the Menu system” Directory printed after 0 is entered.	
3.	Something, Hi, Base	“Please enter a valid selection.” Menu reprinted	“Please enter a valid selection” message only appears when “Hi” is entered.  The program accepts the input if the first letter of a word is valid.	Fail

```

C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 M
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
B
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Welcome to the Football season!
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!

```

Figure 19: Repaired Code Test 3 Case 1 Output (Initial attempt)

Finally, the program works much better than it used to. However, I noticed one small error that still wasn't fixed. When entering the letter "F", the program would print welcome messages for both the football and soccer seasons. I decided to find the cause of this issue, repair it, and retest.

Within a minute I realized that the cause of this issue was a missing "break" statement between the "Football" and "Soccer" cases, something that was present between all other cases. The issue was quite an easy one to repair.

```

62  /* Display the Results*/
63  void showResults(char value) {
64      switch (value) {
65          /*By Mark Kardash: Moving cases in switch statement to correct indentation*/
66          case 'F':
67          case 'f':
68              printf("Welcome to the Football season!\n");
69          case 'S':
70          case 's':
71              printf("Welcome to the Soccer season!\n");
72              break;
73          case 'B':
74          case 'b':
75              printf("Welcome to the Baseball season!\n");
76              break;
77          case 'E':
78          case 'e':
79              printf("Exiting the Menu system!\n");
80              break;
81          default:
82              printf("Please enter a valid selection\n");
83      }
84      //By Mark Kardash: Adding a "while" statement that will handle transitions to next line.
85      while (getchar() != '\n');
86  }

```

Figure 20: Missing “break” statement between football and soccer seasons.

I may have accidentally removed the “break” statement when I was indenting the cases earlier.

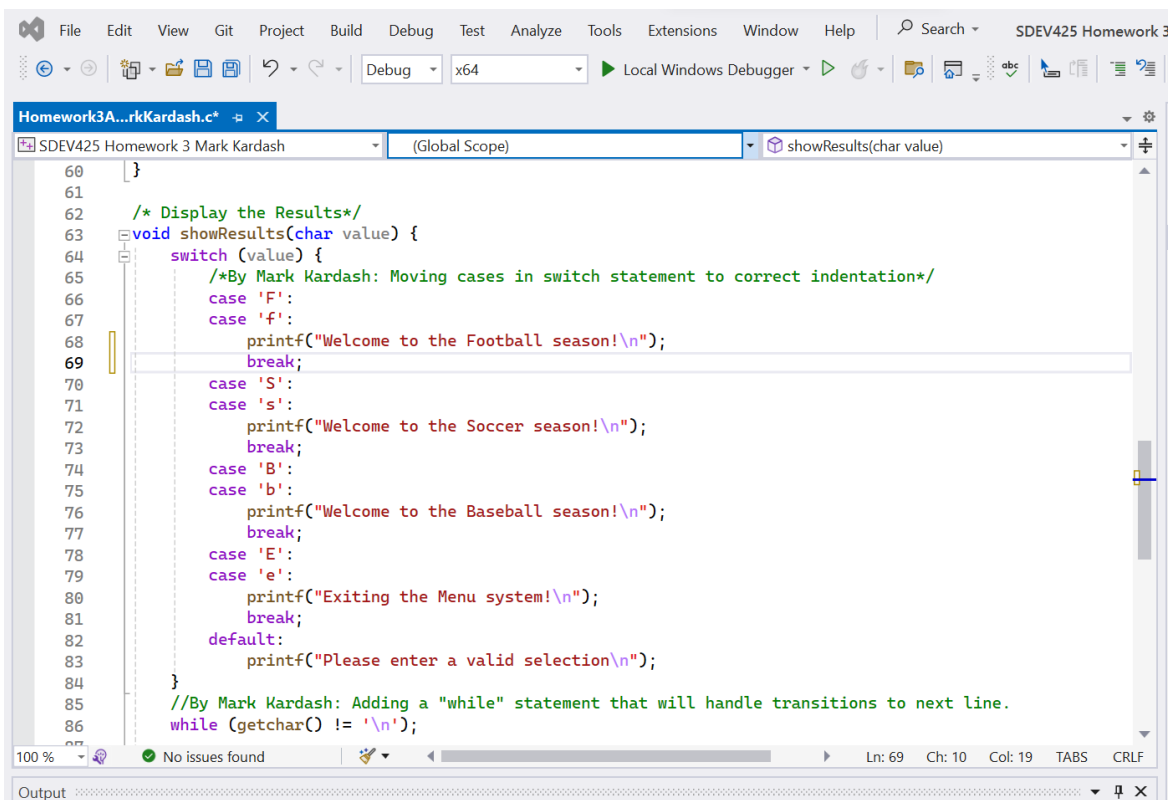
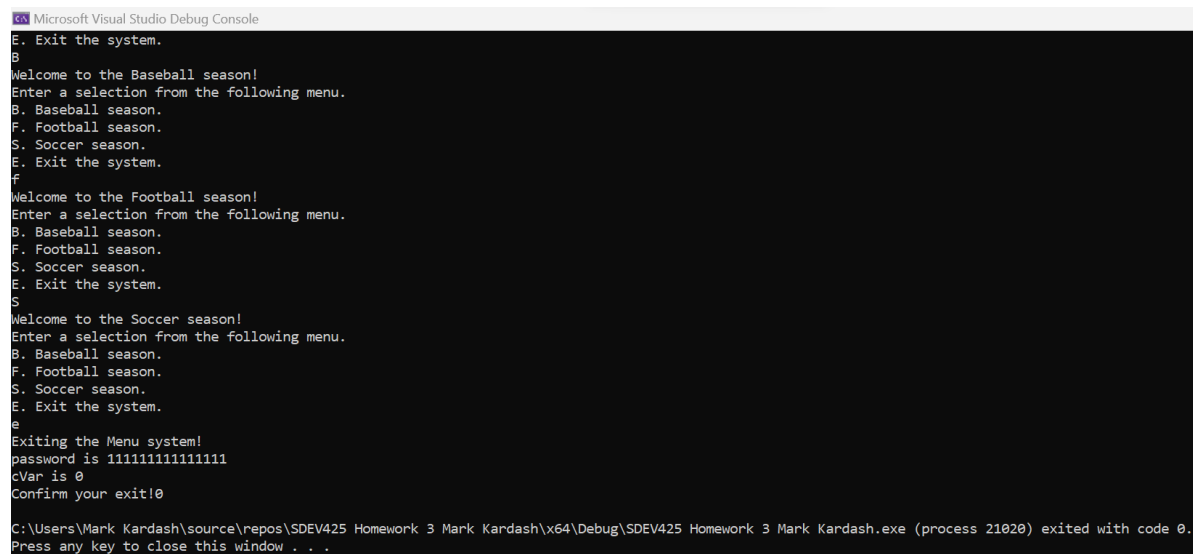


Figure 21: Corrected code with “break” statement between football and soccer seasons.



The “break” statement was then added in, and the program is now to be retested.



```

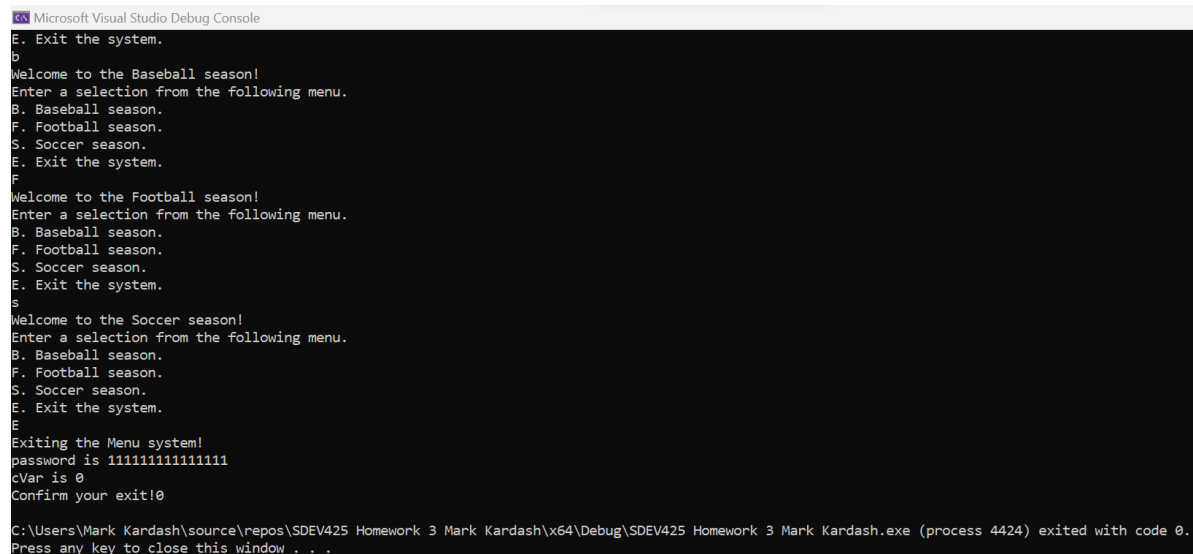
Microsoft Visual Studio Debug Console
E. Exit the system.
B
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Welcome to the Football season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!0

C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 21020) exited with code 0.
Press any key to close this window . . .

```

Figure 22: Repaired Code Test 3 Case 1 Output (Final Attempt)

Finally, the program began to work normally, as it passed the first test case with flying colors.



```

Microsoft Visual Studio Debug Console
E. Exit the system.
b
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
F
Welcome to the Football season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
s
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
E
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!0

C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 4424) exited with code 0.
Press any key to close this window . . .

```

Figure 23: Repaired Code Test 3 Case 2 Output

The second test case had perfect results as well.

```

C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 M
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Something
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Hi
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Base
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.

```

Figure 24: Repaired Code Test 3 Case 3 Input

Things proved slightly more complicated with the third case, as the program would accept even incorrect inputs, so long as the first letter of the input was valid. The “Please enter a valid selection” message was only printed when the input was “Hi”, a word which did not have a valid character as its first letter. Unfortunately, however, I was not able to figure out how exactly I could fix this problem, and decided to simply begin implementing the security recommendations.

#### Implementing CERT Security Recommendations on the Corrected Code:

### DCL20-C: Explicitly specify void when a function accepts no arguments

The first issue or recommendation that I saw within my program was DCL20-C: Explicitly specify void when a function accepts no arguments. According to Carnegie Melon University (CMU), if a function declarator has empty parentheses, it becomes obsolescent, as by being empty, it specifies that no parameters for the function have been provided (DCL20-C, n.d.). This is why, in there are no arguments in a function, it should immediately be specified as void.

The only time this error is ever spotted in the code is at the very beginning, on line 8. Specifically, in the “showMenu()” function, which is the only one not specified as “void”. In its present condition, the function can have a very ambiguous interface, and cause the outflow of sensitive information (DCL20-C, n.d.). For example, a caller may enter a random number, or even letter into the function, which can result in an error. In this case, this is even more urgent, since the definition of the function does have a “void” parameter, but not the call.

```
90      /* Display the Menu*/
91      void showMenu(void) {
92          printf("Enter a selection from the following menu.\n");
93          printf("B. Baseball season.\n");
94          printf("F. Football season.\n");
95          printf("S. Soccer season.\n");
96          printf("E. Exit the system.\n");
97      }
```

Figure 25: “showMenu” function definition specified as “void”.

```

7      // should have void listed
8      void showMenu();
9
10     // Define a variable to hold a password
11     // and the copy
12     char password[15];
13     char cpassword[15];
14

```

Figure 26: Original call of “showMenu” without “void” specification

The fix of this issue was quite simple, as we only needed to specify the function call as “void”.

```

7      // should have void listed
8      void showMenu(void); // By Mark Kardash: Specifying the function as "void" (Regulation DCL20-C).
9

```

Figure 27: Fixed Code with DCL20-C addressed.

I will be testing the program after each code correction, using the first of the test cases I previously used.

#### Enhanced Code Test 1: DCL20-C

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
1.	B, f, S, e	<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p>	<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p>	Pass

		<p>“Welcome to the Soccer Season”</p> <p>“Exiting the Menu system”</p> <p>(Directory printed after 0 is entered)</p>	<p>“Welcome to the Soccer Season”</p> <p>“Exiting the Menu system”</p> <p>(Directory printed after 0 is entered)</p>	
--	--	--	--	--

```

Microsoft Visual Studio Debug Console
E. Exit the system.
B
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Welcome to the Football season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!0

C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 17236) exited with code 0.
Press any key to close this window . . .

```

Figure 28: Enhanced Code Test 1 Output

The change was very small, and the system did not suffer any damage, passing the test perfectly.

MSC17-C: Finish every set of statements associated with a case label with a break statement

Originally, I had actually already fixed this issue earlier without even realizing it. CMU explains that a “switch” statement consists of several “Case” statements, arranged one under another, with an optional “default” statement. According to the university, a “break” statement should come after each pair of cases (MSC17-C, n.d.). The original code provided to me did not have a “break” statement after the cases responsible for the letters “F” and “f”. According to CMU, if such a statement is omitted, “if omitted, control flow falls through to the next case in the switch statement block.” (MSC17-C, n.d.). This was very clearly seen when the program printed out the “Welcome” statements for both the football and soccer seasons in response to an input of “F”. Detailed screenshots of the issue, and its fix, as well as the subsequent test run, can be found on pages 17-18 of this document. I had not realized this was one of the intentional errors, and assumed that I accidentally deleted the statement while indenting the cases.

#### MSC24-C: Do not use deprecated or obsolescent functions

After reviewing this rule, I noticed that the code uses several functions, particularly “printf()” and “memcpy()”, that are considered deprecated. The CMU warns of potential security compromise should these functions not be replaced (MSC24-C, n.d.). Therefore, I went about replacing them, as shown in the code below.

```

38      | // Display variable values
39      | printf("password is %s\n", password);
40      | printf("cVar is %d\n", cVar);
41      |

```

Figure 29: “printf” statements to print password from original code.

```

62  /* Display the Results*/
63  void showResults(char value) {
64      switch (value) {
65          /*By Mark Kardash: Moving cases in switch statement to correct indentation*/
66          case 'F':
67          case 'f':
68              printf("Welcome to the Football season!\n");
69              break;
70          case 'S':
71          case 's':
72              printf("Welcome to the Soccer season!\n");
73              break;
74          case 'B':
75          case 'b':
76              printf("Welcome to the Baseball season!\n");
77              break;
78          case 'E':
79          case 'e':
80              printf("Exiting the Menu system!\n");
81              break;
82          default:
83              printf("Please enter a valid selection\n");
84      }
85      //By Mark Kardash: Adding a "while" statement that will handle transitions to next line.
86      while (getchar() != '\n');
87  }
88

```

Figure 30: “printf()” statements to print case messages from original code.

```

90  /* Display the Menu*/
91  void showMenu(void) {
92      printf("Enter a selection from the following menu.\n");
93      printf("B. Baseball season.\n");
94      printf("F. Football season.\n");
95      printf("S. Soccer season.\n");
96      printf("E. Exit the system.\n");
97  }

```

Figure 31: “printf()” statements to print the menu from original code.

```

41
42  // Copy password
43  memcpy(cpassword, password, sizeof(password));
44
45  // Pause before exiting

```

Figure 32: “memcpy()” statement, from original code, to copy password

As a fix for this issue, I decided to replace the statements with more secure ones.

```

18     printf_s("Welcome to the C Array Program!\n");
19     // Variables
20     char cont = 'y'; // To continue with loop
21     int cVar = 0; // process variable
22
23     // Display menu and Get Selection
24     while (cont != 'E' && cont != 'e') {
25         // Display the Menu
26         showMenu();
27
28         // Get the user selection
29         cont = getchar();
30
31         // Display the menu response
32         showResults(cont);
33     }
34     // Call the Copy routine
35     fillPassword(sizeof(password), password);
36
37     // Display variable values
38     printf_s("password is %s\n", password);
39     printf_s("cVar is %d\n", cVar);
40
41     // Copy password
42     memcpy_s(cpassword, password, sizeof(password));
43
44     // Pause before exiting
45     char confirm;

```

Figure 33: Replacing statements- Part 1

Here, the “printf()” statements used to print the password have been replaced by “printf\_s()” statements, while the “memcpy()” statement was replaced by the safer “memcpy\_s()”, as instructed by CMU (MSC24-C, n.d.).

```

    // Pause before exiting
    char confirm;
    printf_s("Confirm your exit!");
    confirm = getchar();
    return 0;
}

// Make a String of '1's
void fillPassword(size_t n, char dest[]) {
    // Should be n-1
    for (size_t j = 0; j < n; j++) {
        dest[j] = '1';
    }
    // Add null terminator for string
    dest[n] = '\0';
}

```

Figure 34: Replacing statements – Part 2

Replacing the statement for “confirm your exit”.



```

61  /* Display the Results*/
62  void showResults(char value) {
63      switch (value) {
64          /*By Mark Kardash: Moving cases in switch statement to correct indentation*/
65          case 'F':
66          case 'f':
67              printf_s("Welcome to the Football season!\n");
68              break;
69          case 'S':
70          case 's':
71              printf_s("Welcome to the Soccer season!\n");
72              break;
73          case 'B':
74          case 'b':
75              printf_s("Welcome to the Baseball season!\n");
76              break;
77          case 'E':
78          case 'e':
79              printf_s("Exiting the Menu system!\n");
80              break;
81          default:
82              printf_s("Please enter a valid selection\n");
83      }
84      //By Mark Kardash: Adding a "while" statement that will handle transitions to next line.
85      while (getchar() != '\n');
86  }
87

```

Figure 35: Replacing statements- Part 3

Here, I replaced all the “printf()” statements in the cases of the “switch” statement with “printf\_s()”.

```

/* Display the Menu*/
void showMenu(void) {
    printf_s("Enter a selection from the following menu.\n");
    printf_s("B. Baseball season.\n");
    printf_s("F. Football season.\n");
    printf_s("S. Soccer season.\n");
    printf_s("E. Exit the system.\n");
}

```

Figure 36: Replacing statements – Part 4

I then replaced all the “printf()” statements in the menu with “printf\_s()”.

I did notice, however, that replacing memcpy() with memcpy\_s() caused the very end of the function to be highlighted in red (As can be seen in Figure 33). Hovering over it gave me a “too few arguments for function call” error. I later figured out that this was caused by the argument “sizeof(cpassword)” missing from the function. Therefore, I made sure to add it in.

```

37 // Display variable values
38 printf_s("password is %s\n", password);
39 printf_s("cVar is %d\n", cVar);
40
41 // Copy password
42 memcpy_s(cpassword, sizeof(cpassword), password, sizeof(password));
43
44 // Pause before exiting

```

Figure 37: Adding “sizeof(cpassword)” to “memcpy\_s” function

By giving my application the ability to calculate the size of the copied password, I resolved the error. With the changes applied, I will now test the program.

Enhanced Code Test 2: MSC24-C

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
2.	B, f, S, e	“Welcome to the Baseball Season”  “Welcome to the Football Season”  “Welcome to the Soccer Season”	“Welcome to the Baseball Season”  “Welcome to the Football Season”  “Welcome to the Soccer Season”	Pass

		“Exiting the Menu system”  (Directory printed after 0 is entered)	“Exiting the Menu system”  (Directory printed after 0 is entered)  .	
--	--	--	---	--

```

Microsoft Visual Studio Debug Console
E. Exit the system.
B
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Welcome to the Football season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!0

C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 18588) exited with code 0.
Press any key to close this window . . .

```

Figure 38: Enhanced Code Test 2 Output

Once again, the app has passed the test with flying colors.

STR31-C: Guarantee that storage for strings has sufficient space for character data and the null terminator

According to CMU, this issue occurs when a program's buffer is not large enough to hold sufficient data. Should this occur, the program becomes at risk of a buffer overflow (STR31-C, n.d.), in which data can “spill” beyond the buffer's boundaries, creating a gateway for attackers. The University recommends making sure that the destination of the data is of sufficient size to hold the data and null-termination character, as well as using truncation to limit copies, as a fix to this issue (STR31-C, n.d.). In the case of this particular application, the problem is found in the “fillPassword()” function call, on lines 52-59. The function was large enough to fit the password data, but not the null terminator, which meant that its calculation of the password data was always off by 1.

```

55     // Make a String of '1's
56     void fillPassword(size_t n, char dest[]) {
57         // Should be n-1
58         for (size_t j = 0; j < n; j++) {
59             dest[j] = '1';
60         }
61         // Add null terminator for string
62         dest[n] = '\0';
63     }

```

Figure 39: Null terminator not accounted for in “fillPassword()” function

My very simple fix for this issue was to put a “-1” directly after the “n” in “j<n”, which would free space for the null terminator.

```

55     // Make a String of '1's
56     void fillPassword(size_t n, char dest[]) {
57         // Should be n-1
58         //By Mark Kardash: Adding "-1" to "j < n-1" to account for null terminator
59         for (size_t j = 0; j < n-1; j++) {
60             dest[j] = '1';
61         }
62         // Add null terminator for string
63         dest[n] = '\0';
64     }

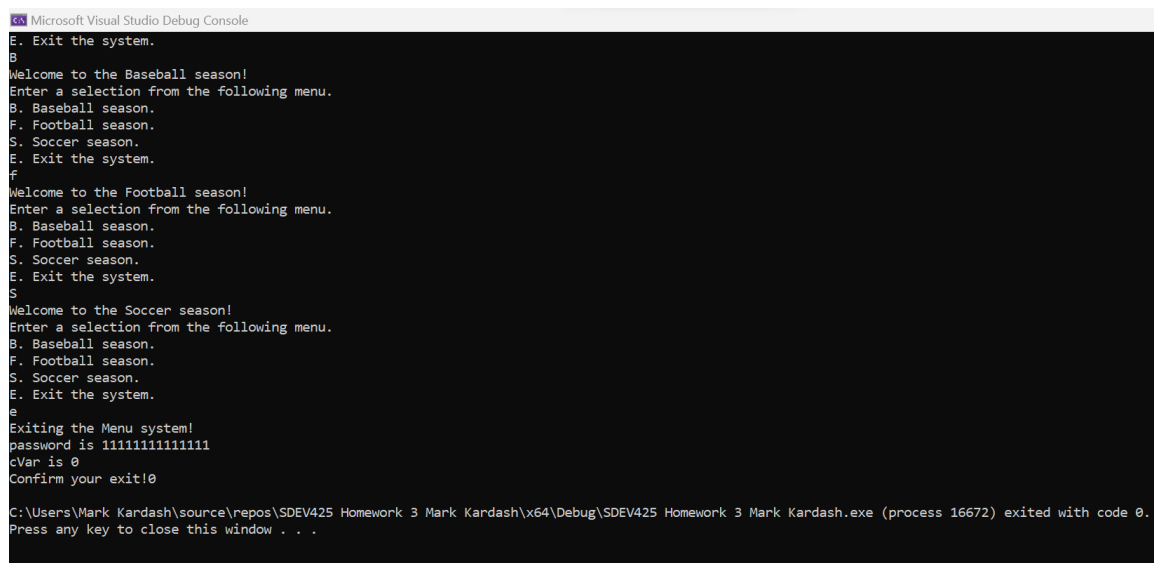
```

Figure 40: Accounting for null terminator

As always, after every change, I tested the program.

Enhanced Code Test 3: STR31-C

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
3.	B, f, S, e	“Welcome to the Baseball Season”  “Welcome to the Football Season”  “Welcome to the Soccer Season”  “Exiting the Menu system”  (Directory printed after 0 is entered)	“Welcome to the Baseball Season”  “Welcome to the Football Season”  “Welcome to the Soccer Season”  “Exiting the Menu system”  (Directory printed after 0 is entered)  .	Pass



```

Microsoft Visual Studio Debug Console
E. Exit the system.
B
Welcome to the Baseball season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Welcome to the Football season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
Welcome to the Soccer season!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Exiting the Menu system!
password is 1111111111111111
cVar is 0
Confirm your exit!0

C:\Users\Mark Kardash\source\repos\SDEV425 Homework 3 Mark Kardash\x64\Debug\SDEV425 Homework 3 Mark Kardash.exe (process 16672) exited with code 0.
Press any key to close this window . . .

```

Figure 41: Enhanced Code Test 3 Output

The program is still working perfectly.

### FIO34-C: Distinguish between characters read from a file and EOF or WEOF

This issue was a difficult one for me to grasp. CMU explains that it has to do with byte functions, such as “fgetc()”, “getc()”, and “getchar()”. Each of them read stream characters, and return them as integers. In the case that the stream is at the end of a file, or if an error in reading happens, EOF (“End of File” character) is returned after the appropriate indicators (“error” and “end-of-the-line”) are set (FIO34-C, n.d.). Should the functions succeed, the returned character is cast into an “unsigned char”. Normally, the returned character should not match the value of any unsigned character. However, in environments where the variables “int” and “char” have the same width, a character with a bit-pattern identical to the EOF character can be read and returned (FIO34-C, n.d.). Attackers can achieve this by inserting into data streams or files a value identical to EOF. CMU suggests that, in dealing with this problem, developers use the functions

“ferror()” and “feof()” to verify EOF-related matters and errors, as they are unaffected by integer and character sizes (FIO34-C, n.d.).

The risk of FIO34-C also exists within the code examined here. In the “while” loop responsible for handling the “con!=E” and “cont!=e” statements, there are no secure statements to handle EOF issues and errors, leaving the code vulnerable to attacks, such as injection.

```

24      // Display menu and Get Selection
25      while (cont != 'E' && cont != 'e') {
26          // Diaply the Menu
27          showMenu();
28
29          // Get the user selection
30          cont = getchar();
31
32          // Display the menu response
33          showResults(cont);
34      }
35      // Call the Copy routine
36      fillPassword(sizeof(password), password);

```

Figure 42: No EOF-handling functions included in code.

Therefore, to fix the code, I added an “ferror()” and an “feof()” function to handle such matters. I also noticed that the code used a “getchar()” method to read character input. However, this method read each character one by one, rather than reading them in a single line, which could negatively affect the memory for data, contributing to the unsafe buffer practices that are also related to STR31-C. I therefore decided to change the “getchar()” function to “getline()”, which can adjust the size of the buffer memory, depending on the size of the data. I placed all of this in a loop to control the buffer.

```

//By Mark Kardash: Setting variables and character size to proceed with EOF loop
int ch;
size_t n = 5;
char* buffer;
buffer = (char*)malloc(n * sizeof(char));

```

Figure 43: Setting variables for handling EOF.

Before creating the loop to handle EOF issues, I must set up all required variables that will be used in it. The variable “ch” is the character that the program will accept. “n” measures the size of the buffer, while the buffer has its memory allocation (“malloc”) designed and adjusted within it.

```

31 // Display menu and Get Selection
32 while (cont != 'E' && cont != 'e') {
33     // Display the Menu
34     showMenu();
35
36     //By Mark Kardash: "if" loop to handle EOF matters and errors
37     if (getline(&buffer, &n, stdin) == -1) {
38         printf_s("Error: Characters unable to be read.");
39     }
40
41     else {
42         char* p = strchr(buffer, '\n');
43         if (p) {
44             *p = '\0';
45         }
46         else {
47             while ((ch = getchar()) != '\n' && ch != EOF);
48             if (ch == EOF && !feof(stdin) && !ferror(stdin)) {
49                 printf_s("Error: End-of-File");
50             }
51         }
52     }
53
54     // Get the user selection
55     cont = getchar();
56

```

Figure 44: “if” loop to handle EOF issues.

Now, I can finally designed the “if” loop that will handle any problems or issues related to EOF.

In this loop, the “getline()” statement takes in the “buffer”, the size “n”, and the input, to either



print out an error message it is unable to read the characters, read the data if everything is correct, or throw an “End-of-File” error message if EOF is reached.

Having applied the changes, I will now test the application.

Enhanced Code Test 4: FIO34-C

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:

4.	B, f, S, e	<p>“Welcome to the Baseball Season”</p> <p>“Welcome to the Football Season”</p> <p>“Welcome to the Soccer Season”</p> <p>“Exiting the Menu system”</p> <p>(Directory printed after 0 is entered)</p>	<p>Initial:</p> <p>“Unresolved External Symbol” error</p> <p>Second:</p> <p>“Unresolved External Symbol” error.</p> <p>Final:</p> <p>Switch statement only prints default option.</p> <p>Even letter “e” does not give results.</p>	<p>Fail (Initial)</p> <p>Fail (Final)</p>
----	------------	--	---	---

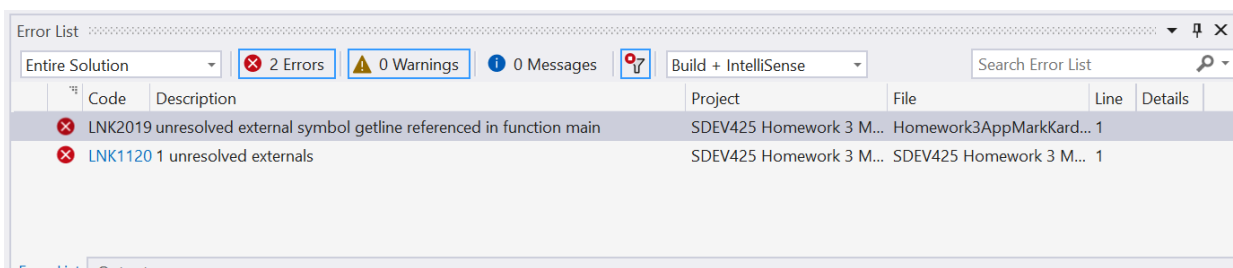


Figure 45: Enhanced Code Test 4 Error

Running the code initially gave me a “unresolved external symbol” error. This happens when a function has been declared, but does not have a definition. I realized that I was probably getting it because I had not included a loop for the case that memory could not be allocated.

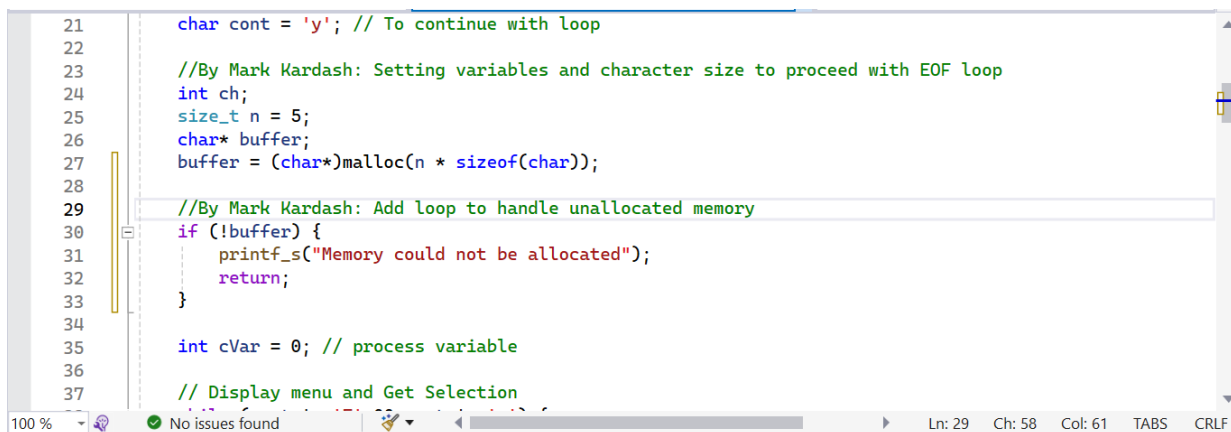
```

23      //By Mark Kardash: Setting variables and character size to proceed with EOF loop
24      int ch;
25      size_t n = 5;
26      char* buffer;
27      buffer = (char*)malloc(n * sizeof(char));
28
29      int cVar = 0; // process variable
--

```

Figure 46: No loop with error message for memory that could not be allocated.

I now knew I had to add a loop for the case that memory could not be allocated.



```

21      char cont = 'y'; // To continue with loop
22
23      //By Mark Kardash: Setting variables and character size to proceed with EOF loop
24      int ch;
25      size_t n = 5;
26      char* buffer;
27      buffer = (char*)malloc(n * sizeof(char));
28
29      //By Mark Kardash: Add loop to handle unallocated memory
30      if (!buffer) {
31          printf_s("Memory could not be allocated");
32          return;
33      }
34
35      int cVar = 0; // process variable
36
37      // Display menu and Get Selection

```

The screenshot shows a code editor window with the following code. The code includes a comment at the top: "char cont = 'y'; // To continue with loop". Below this, there is a comment: "//By Mark Kardash: Setting variables and character size to proceed with EOF loop". The code then declares variables: "int ch;", "size\_t n = 5;", "char\* buffer;", and "buffer = (char\*)malloc(n \* sizeof(char));". Below this, there is a comment: "//By Mark Kardash: Add loop to handle unallocated memory". The code then has an "if" loop: "if (!buffer) {" followed by "printf\_s(\"Memory could not be allocated\");", "return;", and "}". Below the "if" loop, there is a comment: "int cVar = 0; // process variable". At the bottom, there is a comment: "// Display menu and Get Selection". The editor status bar at the bottom shows "100 %", "No issues found", and "Ln: 29 Ch: 58 Col: 61 TABS CRLF".

Figure 47: “if” loop to handle unallocated memory.

I also eventually realized I should move my “cont” statement to inside the loop, change the “getchar()” within it to a “buffer[0]” to define buffer size.

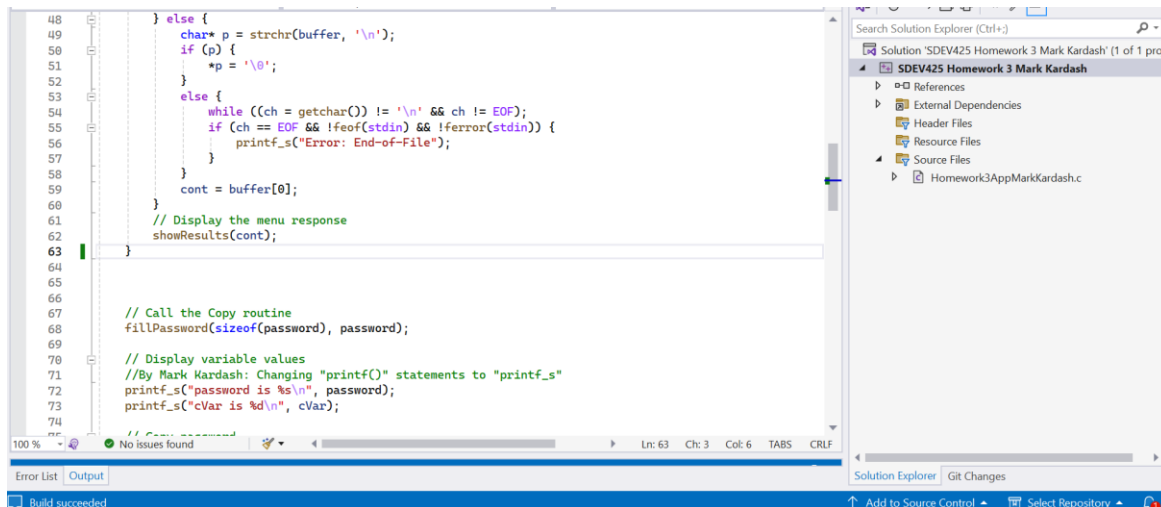


Figure 48: Moving “cont” into “if” loop, and changing “getchar()” to “buffer[0]”.

Now, I retest the app.

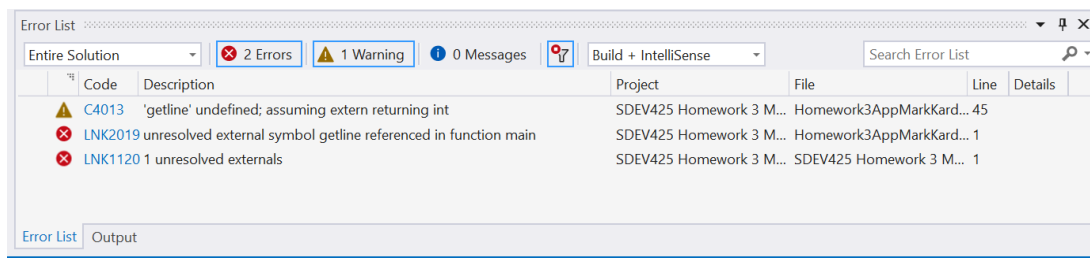


Figure 49: “getline()” “undefined” error.

Running the program again told me that the “getline()” function I had used for the EOF “if” loop was undefined. So, I looked back at my code to see what exactly I had done wrong.

After much fiddling, I figured out that the “<string.h>” library I was using did not contain the statement “getline()”, and that it was instead contained in “<string>”. However, importing “<string>” caused some of my other variables to become undefined. As a result, I decided to change the “getline()” function back to the previously used “getchar()”.

```

39
40 // Display menu and Get Selection
41 while (cont != 'E' && cont != 'e') {
42     // Diaply the Menu
43     showMenu();
44
45     //By Mark Kardash: "if" loop to handle EOF matters and errors
46     if (getchar(&buffer, &n, stdin) == -1) {
47         printf_s("Error: Characters unable to be read.");
48     }
49
50     else {
51         char* p = strchr(buffer, '\n');
52         if (p) {
53             *p = '\0';
54         }
55         else {

```

Figure 50: “getline()” function replaced with “getchar()” again, to comply with library rules.

For the program to work, I also had to remove the “while (getchar() != ‘\n’;” statement at the end of the “switch” statement cases, that was supposed to handle line transitions, as it was causing problems with the displaying of the menu. With the changes made, it was not even really needed anymore.

Finally, after a huge struggle to get things right, I retested (See all results above – Test 4 Table).

```

C:\Users\Mark Kardash\source\repos\SDEV425 Homework
E. Exit the system.
B
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.

```

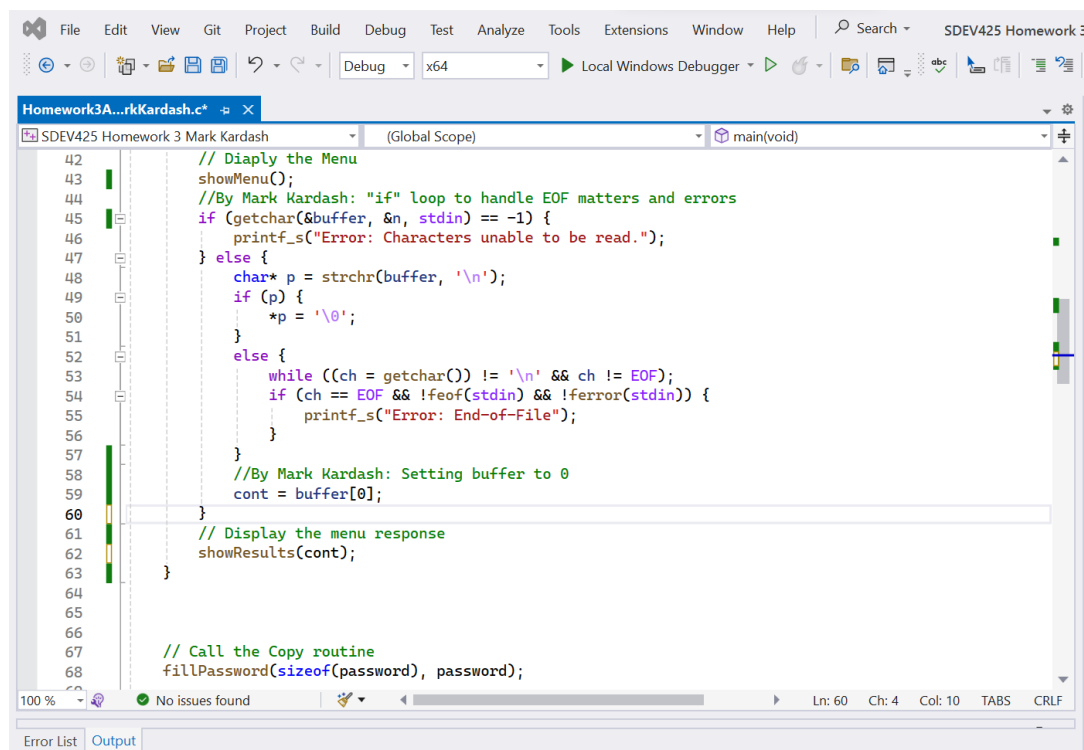
Figure 51: Only default option printed as output

For some bizarre reason, the program was now only printing the default option. Not even the letter “e” gave the desired output anymore. So, again, I went investigating. However, I failed to identify the true cause of the problem, though I suspect it has something to do with me setting the buffer to 0. Still, I decided to simply move on to describing and implementing the last rule I found to be necessary.

#### MEM30-C: Do not access freed memory

This was the last security enhancement I had to deal with, and probably the easiest one to implement. This recommendation warns users against accessing a dangling pointer, which is a

pointer to deallocated memory. Accessing deallocated memory through such pointers can make an application vulnerable in ways that can be exploited by attackers (MEM30-C, n.d.). Initially, my code was completely unprotected from this risk, as any function within it could read deallocated memory.



```

42 // Diaply the Menu
43 showMenu();
44 //By Mark Kardash: "if" loop to handle EOF matters and errors
45 if (getchar(&buffer, &n, stdin) == -1) {
46     printf_s("Error: Characters unable to be read.");
47 } else {
48     char* p = strchr(buffer, '\n');
49     if (p) {
50         *p = '\0';
51     }
52     else {
53         while ((ch = getchar()) != '\n' && ch != EOF);
54         if (ch == EOF && !feof(stdin) && !ferror(stdin)) {
55             printf_s("Error: End-of-File");
56         }
57     }
58     //By Mark Kardash: Setting buffer to 0
59     cont = buffer[0];
60 }
61 // Display the menu response
62 showResults(cont);
63 }
64
65
66
67 // Call the Copy routine
68 fillPassword(sizeof(password), password);

```

Figure 52: Original unprotected code.

Even though my code was no longer working properly, I decided to still implement my solution. For this, I simply added a “free(buffer);” statement at the very end of the main method, at the end of the EOF loop. This way, I can ensure that none of the code’s functions have access to deallocated memory.

```

42 // Display the Menu
43 showMenu();
44 //By Mark Kardash: "if" loop to handle EOF matters and errors
45 if (getchar(&buffer, &n, stdin) == -1) {
46     printf_s("Error: Characters unable to be read.");
47 } else {
48     char* p = strchr(buffer, '\n');
49     if (p) {
50         *p = '\0';
51     }
52     else {
53         while ((ch = getchar()) != '\n' && ch != EOF);
54         if (ch == EOF && !feof(stdin) && !ferror(stdin)) {
55             printf_s("Error: End-of-File");
56         }
57     }
58     //By Mark Kardash: Setting buffer to 0
59     cont = buffer[0];
60 }
61 // Display the menu response
62 showResults(cont);
63 }
64 //By Mark Kardash: Adding "free" statement for buffer, to handle deallocated memory.
65 free(buffer);
66
67
68
69 // Call the main function

```

Figure 53: “Freeing” the buffer.

Having implemented the change, I, as always, test.

### Enhanced Code Test 5: MEM30-C

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
5.	B, f, S, e	“Welcome to the Baseball Season”  “Welcome to the Football Season”  “Welcome to the Soccer Season”	Switch statement only prints default option.  Even letter “e” does not give results.	Fail



		“Exiting the Menu system”  (Directory printed after 0 is entered)		
--	--	--	--	--

```

C:\Users\Mark Kardash\source\repos\SDEV425 Homework
E. Exit the system.
B
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.

```

Figure 54: Enhanced Code Test 5 Output

As I expected, the program unfortunately still prints out only the default output option, and I am uncertain as to why.

Final General Application Test:

Having now completed the analysis and implementation of various security enhancements, I thought it would be appropriate to run one final, general test of the entire application, on all test cases that I had run previously. This way, I can have a final, general assessment of the program I ended up with.

Test Case #	Inputs:	Expected Outputs:	Actual Outputs:	Pass/Fail:
1.	B, f, S, e	“Welcome to the Baseball Season”  “Welcome to the Football Season”  “Welcome to the Soccer Season”  “Exiting the Menu system”  (Directory printed after 0 is entered)	Switch statement only prints default option as output. Even letter “e” does not give desired result.	Fail
2.	“b”, F, s, E  (Ignore quotations on b.  They are to signify that it is	“Welcome to the Baseball Season”  “Welcome to the Football Season”	Switch statement only prints default option as output. Even letter “e” does	Fail

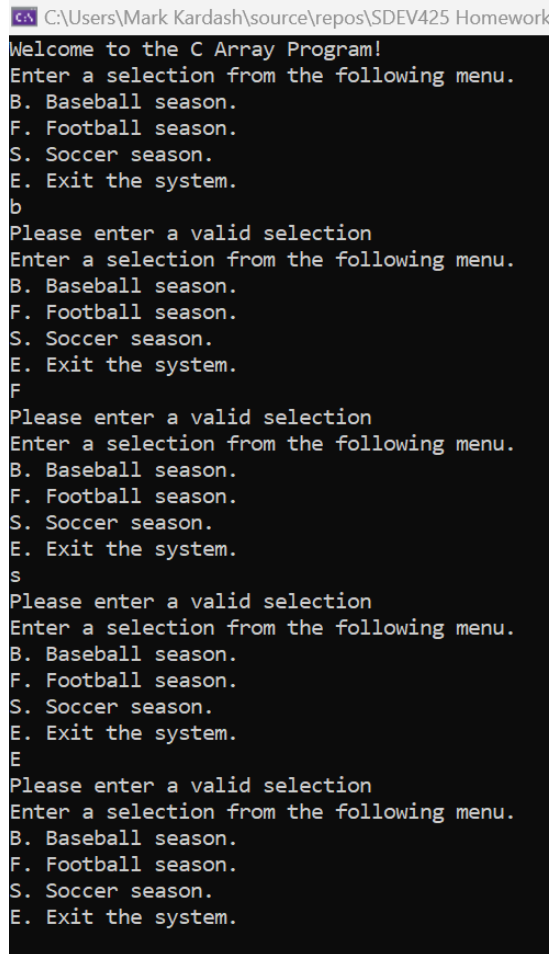
	lowercase.  Otherwise  Microsoft Word  turns it into  uppercase).	“Welcome to the  Soccer Season”  “Exiting the  Menu system”  Directory  printed after 0 is  entered.	not give desired  result.	
4.	Something, Hi,  Base	“Please enter a  valid selection.”  Menu reprinted		

C:\Users\Mark Kardash\source\repos\SDEV425 Homework

```
E. Exit the system.
B
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
f
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
S
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
e
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
```

Figure 55: General Test Case 1 Output

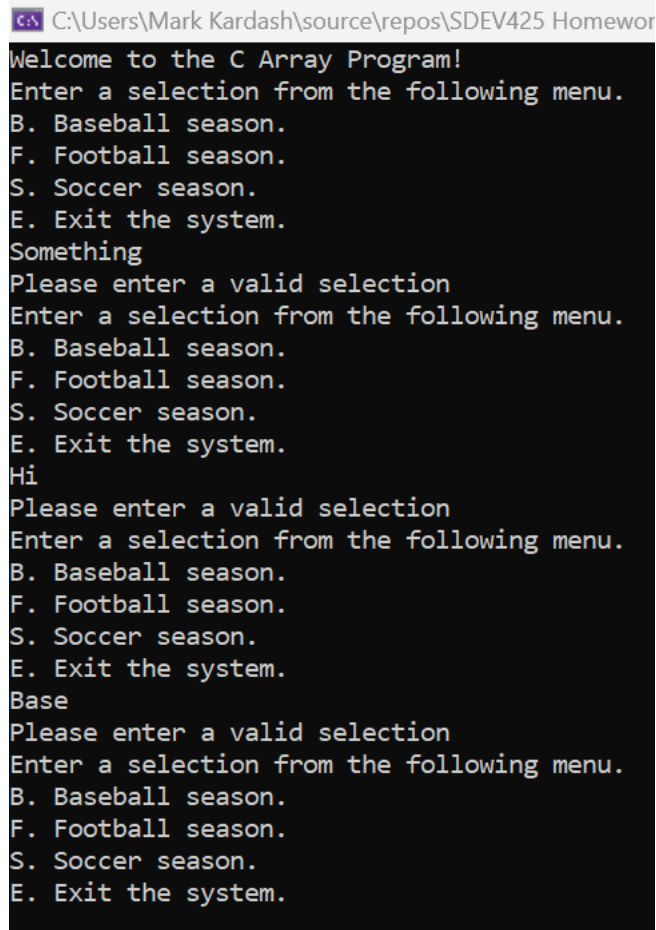
As was already seen before, the program now only outputs the default option for any of the inputs in Test Case 1, with not even the letter “e” giving the correct output like it used to.



```
C:\Users\Mark Kardash\source\repos\SDEV425 Homework
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
b
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
F
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
s
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
E
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
```

Figure 56: General Test Case 2 Output

The Exact same thing is true for the second test case.



```

C:\Users\Mark Kardash\source\repos\SDEV425 Homework
Welcome to the C Array Program!
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Something
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Hi
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.
Base
Please enter a valid selection
Enter a selection from the following menu.
B. Baseball season.
F. Football season.
S. Soccer season.
E. Exit the system.

```

Figure 57: General Test Case 3 Output

And it is true for the third as well.

### Conclusion:

This assignment was a true disaster. My own procrastination resulted in a significant delay with starting it, while a mixture of family circumstances and computer problems aggravated the delay to levels I had never reached before. The assignment itself was a complicated one, although I did quite well after many days of struggle. I was getting through it quite successfully in terms of functionality, and things went sour for me only when I tried to improve EOF functionality. This is because, like I had said before, I had great trouble understanding the “EOF” recommendation,

and evidently created the “while” loop wrong. Overall, this was one of the most complex and extensive assignments I have ever undertaken, although I feel that there may have only been one small mistake hiding in plain sight somewhere, which, if corrected, would make the program work perfectly.

## References

Carnegie Melon University. (n.d.). *DCL20-C. Explicitly specify void when a function accepts no arguments - Sei cert C coding standard - Confluence.*

<https://wiki.sei.cmu.edu/confluence/display/c/DCL20->

[C.+Explicitly+specify+void+when+a+function+accepts+no+arguments](https://wiki.sei.cmu.edu/confluence/display/c/DCL20-C.+Explicitly+specify+void+when+a+function+accepts+no+arguments)

Carnegie Melon University. (n.d.). *FIO34-C. Distinguish between characters read from a file and eof or weof - Sei cert C coding standard - Confluence.*

<https://wiki.sei.cmu.edu/confluence/display/c/FIO34->

[C.+Distinguish+between+characters+read+from+a+file+and+EOF+or+WEOF](https://wiki.sei.cmu.edu/confluence/display/c/FIO34-C.+Distinguish+between+characters+read+from+a+file+and+EOF+or+WEOF)

Carnegie Melon University. (n.d.). *MSC17-C. Finish every set of statements associated with a case label with a break statement - Sei cert C coding standard - Confluence.*

<https://wiki.sei.cmu.edu/confluence/display/c/MSC17->

[C.+Finish+every+set+of+statements+associated+with+a+case+label+with+a+break+statement](https://wiki.sei.cmu.edu/confluence/display/c/MSC17-C.+Finish+every+set+of+statements+associated+with+a+case+label+with+a+break+statement)

Carnegie Melon University. (n.d.). *MSC24-C. Do not use deprecated or obsolescent functions - Sei cert C coding standard - Confluence.* <https://wiki.sei.cmu.edu/confluence/display/c/MSC24->

[C.+Do+not+use+deprecated+or+obsolescent+functions](https://wiki.sei.cmu.edu/confluence/display/c/MSC24-C.+Do+not+use+deprecated+or+obsolescent+functions)

Carnegie Melon University. (n.d.). *STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator - Sei cert C coding standard - Confluence.*

<https://wiki.sei.cmu.edu/confluence/display/c/STR31->

[C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator](https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator)

Carnegie Melon University. (n.d.). *MEM30-C. Do not access freed memory - Sei cert C coding standard* - Confluence. <https://wiki.sei.cmu.edu/confluence/display/c/MEM30-C.+Do+not+access+freed+memory>