

Homework 3

C Program Interface with Corrected Security Issues

Student: Patrick Walsh

School: University of Maryland Global Campus

Course: SDEV 425 6980

Date: 7/24/2021

Professor: Dr. Nicholas Duchon

1. For the first part of this exercise demonstrate your C developer environment is working properly.

You can do this by running any of the sample C code applications. Modify the C code in this example to make the desired functionality work properly. Demonstrate the code works properly through screen captures and describing what changes were made to fix the functionality issues.

The application does not run as expected. To fix the issues, I added a statement “fflush(stdout);” before the `cont = getchar()` in the while loop of the main method so that pending input and output is “flushed”. I added a second `getchar()` statement after the `cont = getchar()` so that text is not being repeated twice. See example below:

```

26
27      // Display the Menu
28      showMenu();
29
30      // Get the user selection
31      fflush(stdout); // NEW CODE: place before getchar() to "flush" pending input and output
32      cont = getchar();
33      getchar(); // NEW CODE: so that loop does not repeat twice
34
35      // Display the menu response
36      showResults(cont);
37
38  }
```

Similarly, I added another “fflush(stdout);” statement before the `confirm = getchar()` outside the while loop inside the main method:

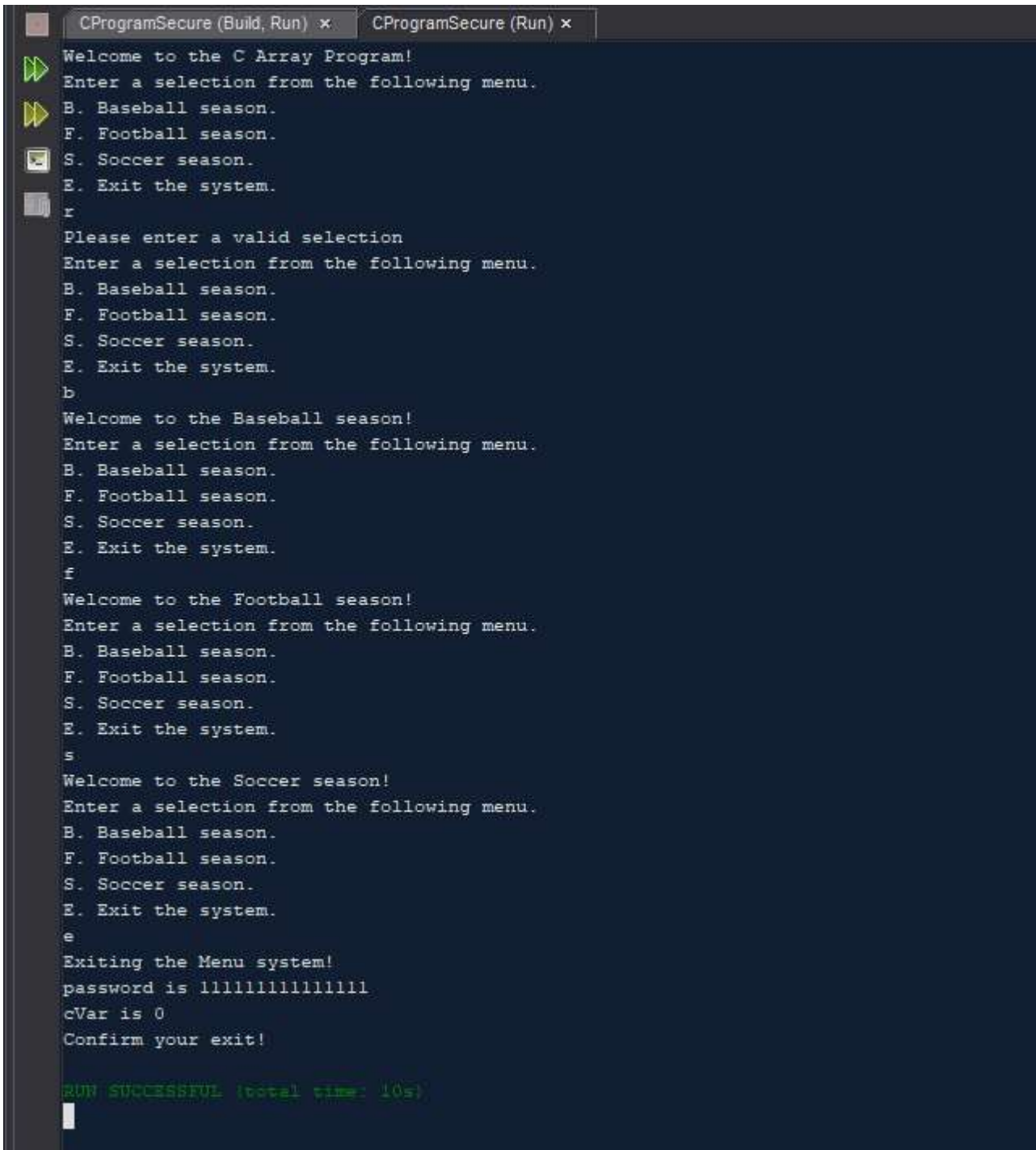
```

49      // Pause before exiting
50      char confirm;
51      printf("Confirm your exit!");
52      fflush(stdout); // NEW CODE: place before getchar() to "flush" pending input and output
53      confirm = getchar();
54
55      return 0;
56  }
```

Finally, I added a break statement after the `printf()` statement inside the `showResults()` function so that the football and soccer messages are not both displayed when football is selected:

```
69 void showResults(char value) {  
70     switch (value) {  
71         case 'F':  
72         case 'f':  
73             printf("Welcome to the Football season!\n");  
74             break; // NEW CODE break needed to avoid printing football and soccer  
75         case 'S':  
76         case 's':  
77             printf("Welcome to the Soccer season!\n");  
78     }
```

The program now runs as expected with no errors. See screen shots below:



```

CProgramSecure (Build, Run) x  CProgramSecure (Run) x
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.
r
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.
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 llllllllllllll
cVar is 0
Confirm your exit!

RUN SUCCESSFUL (total time: 10s)

```

2. Carefully, review the code and perform analysis as needed. Consider the following rules and recommendations and hints for items that you might want to review. Note, that some rules and recommendations listed below may not be found as issues in the code.

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

This coding standard ensures that there is sufficient space for string data by taking into account the null terminator (Razmyslov, 2021a). If the null terminating character is not taken into account, the program may try to write an extra byte of data beyond what is expected (Razmyslov, 2021a). To implement this standard, I modified the code below in the fillPassword() function to include n - 1 rather than n, shaving off the null terminator at the end:

```

66 void fillPassword(size_t n, char dest[]) {
67     // Should be n-1
68     // for (size_t j = 0; j < n; j++) { // OLD CODE
69     for (size_t j = 0; j < (n - 1); j++) { // NEW CODE STR31-C compliant
70         dest[j] = '1';
71     }

```

- **MSC24-C. Do not use deprecated or obsolescent functions.**

This coding standard ensures that C programs do not use deprecated or obsolescent functions which can leave programs vulnerable to security threats (Razmyslov, 2021b). Some functions are considered to be unchecked obsolescent functions, which should be avoided if an equivalent, non-obsolescent function exists (Razmyslov, 2021b). I implemented this standard by checking that no deprecated or obsolescent functions were being used in the program. I then checked for unchecked obsolescent functions and found that the program was using printf() and memcpy() rather than the non-obsolescent alternatives, printf_s() and memcpy_s(). However, I did not implement these functions because they are part of an Annex K replacement which was optional to implement and which is not universally supported by all C compilers (Razmyslov, 2021b). NetBeans did not appear to be compatible with either of these “_s” type functions.

- **FIO34-C. Distinguish between characters read from a file and EOF or WEOF.**

This standard ensures that an EOF is an actual EOF rather than a character due to end-of-file or errors (Svoboda & Britton, 2021). It accomplishes this through the feof() and ferror() functions along with an EOF check (Svoboda & Britton, 2021). I implemented this standard by modifying the program below. First, I changed the getchar() variable cont from a char to an int. I then changed the while loop to check against EOF characters as well as feof() and ferror(). To allow the program to still exit when the user chooses option 'e' or 'E', I added an IF statement inside the while loop. See the modified code below:

```

28 // Variables
29 char cont = 'y'; // OLD CODE
30 int cont; // NEW CODE FIO34-C compliant
31 int cVar = 0; // process variable
32
33 // Display menu and Get Selection
34 while (cont != 'E' && cont != 'e') { // OLD CODE
35 while ((cont != EOF) || (!feof(stdin) && !ferror(stdin))) { // NEW CODE FIO34-C compliant
36
37 // NEW CODE to check if user is trying to exit loop
38 if (cont == 'E' || cont == 'e'){
39     break;
40 }
41 // Display the Menu
42 showMenu();

```

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

This standard ensures that unexpected behavior does not occur when a switch() case event is present (Svoboda & Razmyslov, 2021). The standard specifies that each case statement in the switch() should end with a break statement (Svoboda & Razmyslov, 2021). I implemented this standard by adding a break statement after the printf() for case 'f' inside the showResults() function. This break statement was not in the original code and

caused an unexpected action by printing the football and soccer messages when the user chose football. See example below:

```

83  /* Display the Results*/
84  void showResults(char value) {
85      switch (value){
86          case 'F':
87          case 'f':
88              printf("Welcome to the Football season!\n");
89              break; // NEW CODE break needed to avoid printing football and soccer
90          case 'S':
91          case 's':
92              printf("Welcome to the Soccer season!\n");
93              break;
94          case 'B':
95          case 'b':
96              printf("Welcome to the Baseball season!\n");
97              break;
98          case 'E':
99          case 'e':
100             printf("Exiting the Menu system!\n");
101             break;
102         default:
103             printf("Please enter a valid selection\n");
104     }

```

- **MSC33-C. Do not pass invalid data to the asctime() function.**

This standard ensures that the asctime() function, when used, is not overflowed with unexpected character strings (Seacord & Britton, 2021). The asctime() function is designed to output strings of no more than 25 characters plus a null terminating character, but care must be taken that larger payloads are not delivered, causing a buffer overflow (Seacord & Britton, 2021). I did not implement this standard because the program does not use the asctime() function.

- **DCL20-C. Explicitly specify void when a function accepts no arguments.**

This standard ensures that functions do not accept arbitrary data into their arguments, leading to potential security issues (He & Britton, 2021). The standard specifies that functions which do not take in any arguments should be specified with the void keyword, both in the function declaration and definition (He & Britton, 2021). I implemented this standard by adding the void keyword inside the declaration of the showMenu() function. See example below:

```
12 // Function prototypes
13 void fillPassword(size_t, char[]);
14 void showResults(char);
15 // should have void listed
16 //void showMenu(); // OLD CODE
17 void showMenu(void); // NEW CODE DCL-20C compliant
18
```

- **MEM30-C. Do not access freed memory.**

This standard ensures that freed memory is not accessed by the program after the memory has been deallocated, opening the program up to potential security issues (Seacord & Razmyslov, 2021). The standard specifies that memory should not be written to or read from once it has been freed (Seacord & Razmyslov, 2021). I did not implement this standard because the program does not have memory that it is trying to access after the memory is deallocated.

References

- He, F., & Britton, J. (2021, April 23). *DCL20-C. Explicitly specify void when a function accepts no arguments - SEI CERT C Coding Standard - Confluence*. Carnegie Mellon University.
<https://wiki.sei.cmu.edu/confluence/display/c/DCL20-C.+Explicitly+specify+void+when+a+function+accepts+no+arguments>
- Razmyslov, S. (2021a, June 24). *STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator - SEI CERT C Coding Standard - Confluence*. Carnegie Mellon University.
<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>
- Razmyslov, S. (2021b, June 24). *MSC24-C. Do not use deprecated or obsolescent functions - SEI CERT C Coding Standard - Confluence*. Carnegie Mellon University.
<https://wiki.sei.cmu.edu/confluence/display/c/MSC24-C.+Do+not+use+deprecated+or+obsolescent+functions>
- Seacord, R., & Britton, J. (2021, April 23). *MSC33-C. Do not pass invalid data to the asctime() function - SEI CERT C Coding Standard - Confluence*. Carnegie Mellon University.
[https://wiki.sei.cmu.edu/confluence/display/c/MSC33-C.+Do+not+pass+invalid+data+to+the+asctime\(\)+function](https://wiki.sei.cmu.edu/confluence/display/c/MSC33-C.+Do+not+pass+invalid+data+to+the+asctime()+function)
- Seacord, R., & Razmyslov, S. (2021, June 24). *MEM30-C. Do not access freed memory - SEI CERT C Coding Standard - Confluence*. Carnegie Mellon University.
<https://wiki.sei.cmu.edu/confluence/display/c/MEM30-C.+Do+not+access+freed+memory>
- Svoboda, D., & Britton, J. (2021, April 23). *FIO34-C. Distinguish between characters read from a file and EOF or WEOF - SEI CERT C Coding Standard - Confluence*. Carnegie Mellon University.

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

C.+Distinguish+between+characters+read+from+a+file+and+EOF+or+WEOF

Svoboda, D., & Razmyslov, S. (2021, June 24). *MSC17-C. Finish every set of statements associated with a case label with a break statement - SEI CERT C Coding Standard - Confluence*. Carnegie Mellon University.

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

C.+Finish+every+set+of+statements+associated+with+a+case+label+with+a+break+statement