Reading C Code

Program One
CS 3411 Spring 2022

Motivation

Identifying errors in code, especially around the use of pointers, remains challenging for automated tools. For this reason and others, it is useful to be able to read code. This project aims to help give you practice reading C code that includes relatively complex use of pointers.

Requirements

You must review the attached C code listing and answer the questions below. You should be able to answer the questions without typing in the code.

Assume that on the system for which this code is compiled an `int` is 4 bytes, a `char` is one byte and that the code is compiled for a 32 bit byte-addressable system.

Consider an execution where the values of `argv` and `argc` are given in the attachment. Assume that in this same execution the values of memory are also as given in the attachment. For simplicity, also assume that all calls to `malloc` are successful. Answer the following questions. Where you give a variable name, use *function:name*, where **function** is the function in which the variable is declared and **i** is the variable. An example is **main:i** which refers to the variable **i** declared in **main**.

1. Name all the variables that are not declared. Then give an appropriate declaration.
2. Name all the variables that are declared but not used.
3. Name all the variables that are not initialized (include the ones that were not declared). From this point forward, assume that all variables are initialized correctly.
4. Does the check at line 37 prevent an infinite loop? Explain. If not, also suggest a simple fix.
5. How much memory will be allocated for the `copy` variable in `toCaps`?
6. Give an expression (with program variables) for the correct parameter to `malloc` at line 41 so that there is space for the required array elements. From this point on, assume this statement is correct.
7. What value should the last element of the array pointed to by `copy` have? Does it have this value after execution of the for loop at line 44? Explain.
8. Give an expression (with program variables) for the correct parameter to `malloc` at line 46. From this point on, assume this statement is correct.
9. Does `strcpy` copy the null terminator? Does your expression for the `malloc` at line 46 include space for the null terminator?
10. Examine the for loop that begins on line 53 for efficiency. Identify extraneous operations (by line number) and explain they the are unnecessary.
11. What is the value of `argv[0]`? `&argv[0]`? `argv[2]`?
12. Where (in memory) is the null terminator for the `argv` array?
13. Give the character representation of the parameters to the program. Include the program name (argv[0]).
14. Suppose the value returned for `copy` after execution of line 41 is 0x10000.

(a) Can the value of `&copy[0]` be predicted at this point? If so, what is its value? If not, why?

(b) Can the value of `copy[0]` be predicted at this point? If so, what is its value? If not, why?

15. Suppose that the value of `copy[0]` after execution of line 46 is 0x5000. Can the value of `copy[1]` be predicted? If so, what is its value? If not, explain why.
16. Is the correct value returned? If so, explain why. If not, what value should be returned?
17. Is all the space allocated in the call to `toCaps` freed before the program exits? If not, give code that when added will ensure all the memory is freed.
18. Suppose that the call to `malloc` at line 46 fails for some value of `i` and that the `if` at line 47 returns true. Then there is a memory leak. Why? How can it be fixed?

Collaboration

No collaboration is allowed for this project.

Submission

Upload a PDF document in Canvas with your answers to the questions. Your submission should give each question number, the question, and the answer to that question. This is a large class. Each minute the grader spends on your submission amounts to about an hour and a half across the entire class. Be sure to follow submission requirements closely to save the grader time! Do all you can to format your submission so that it is easy to read.

The project is due on January 26 (Wednesday) at 11pm.

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <limits.h>
//-------------------------------------------------
//  toCaps - Makes a copy of an array of strings. Along the way, all
//           lowercase characters a-z are converted to uppercase.
//           Number of entries limited to INT_MAX (maximum value of
//           an integer).
//
//  Parameters:   char *input[]
//                     A null terminated array of pointers.  The array
//                     values are assumed to be string pointers.
//                     Number of non-zero elements must match size parameter.
//
//
//                int declaredSize
//                     Expected number of elements in the array.
//                     Must be non-negative.
//
//  Return:     Array of pointers; memory must be freed outside
//              this routine
//-------------------------------------------------
char **toCaps(char *input[],int size)
{
  int nStrs;
  char **copy;
  int i;

  //-- Size cannot be negative.
  if (size < 0) return((char **) 0);


  //-- Confirm the actual size by finding the first NULL.
  //   Ensure cannot enter infinite loop.
  while ((input[nStrs]!=(char *)0)&&(nStrs <= INT_MAX)) nStrs++;
  if (nStrs!=size) return((char **)0);

```

```
40:   //-- Allocate the pointer array
41:   copy=(char **)malloc(??);
42:   if ((char **)copy==(char **)0) return((char **)0);
43:   //-- Initialize the array
44:   for (i=0;i<nStrs;i++)
45:     {
46:       copy[i]=(char *)malloc(??);
47:       if (copy[i]==(char *)0) return((char **)0);
48:       if (strcpy(copy[i],input[i])!=copy[i])return((char **)0);
49:     }
50:
51:
52:   //-- Change lower case a-z to upper case
53:   for (i=0;i<nStrs;i++)
54:     {
55:       j=0;
56:       for (j=0;input[i][j]!=0;j++)
57:         {
58:           if ((input[i][j]>=97)&&(input[i][j]<=122)) copy[i][j]=input[i][j]-32;
59:           else copy[i][j]=input[i][j];
60:         }
61:
62:       copy[i][j]==0;
63:     }
64:   return &copy;
65:
66: }
67:
68:
69: int main(int argc, char *argv[])
70: {
71:   char **upper;
72:   int i;
73:   int j;
74:
75:   //--  Error checking
76:   upper=toCaps(argv,argc);
77:   if (upper==(char **)0)
78:     {
```

```
79:         printf("toCaps returned NULL\n");
80:         exit(1);
81:       }
82:     for (i=0;upper[i]!=(char *)0;i++)
83:     {
84:       printf("upper [%d] is <%s>\n",i,upper[i]);
85:     }
86:     for (i=0;upper[i]!=(char *)0;i++) free(upper[i]);
87:
88:
89: }
```

Values of selected variables and memory for assumed execution.

`argv` = 0x1004
`argc` = 3

Memory Location	Value
0x00001004	0x00004000
0x00001008	0x00004006
0x0000100C	0x00004010
0x00001010	0x00000000
0x00004000	0x2E
0x00004001	0x2F
0x00004002	0x70
0x00004003	0x67
0x00004004	0x6D
0x00004005	0x00
0x00004006	0x61
0x00004007	0x62
0x00004008	0x63
0x00004009	0x00
0x00004010	0x31
0x00004011	0x32
0x00004012	0x33
0x00004013	0x34
0x00004014	0x00