

Homework Assignment 2

CSE 399 C++, Spring 2007

SOLUTIONS

Name:

Due: Monday, Jan 29th at noon.

Assumptions: For all of these problems you may assume the following

- `sizeof(int) = 4`; `sizeof(short) = 2`; `sizeof(char) = 1`; all pointers require 4 bytes
- The stack starts at address 100 and grows up.
- The heap starts at address 400 and grows up.
- ??? represents an unknown/uninitialized value.

Question 1 (10 points) : Given the following declarations:

```
int x = 0;
int * p = &x;
int ** p2 = &p;
void * v = &p3;
```

Examine each of the following expressions. If the expression is illegal, write ILLEGAL. If the expression is legal, write its type: 2 points each

- `&v` void **
- `*v` ILLEGAL
- `p2+1` int **
- `*p` int
- `&p2[0]` int **

Question 2 (32 points): Given the following code:

```
int x = 3;          /* x is at address 100 */
int y = 4;          /* y is at address 104 */
int * p = &x;       /* p is at address 108 */
int ** p2 = &p;     /* p2 is at address 112 */
/* Location 1 */
*p = 6;
/* Location 2 */
*p2 = &y;
**p2 = 11;
```

```

/* Location 3 */
*p = 12;
/* Location 4 */

```

Fill in the following table with the values of x, y, p, and p2 at the above indicated 4 locations: 2 points each box

	Loc 1	Loc 2	Loc 3	Loc 4
x	3	6	6	6
y	4	4	11	12
p	100	100	104	104
p2	108	108	108	108

Question 3 (18 points): A misguided C programmer writes the following function to read a line of input from the user (assume the `getchar()` function reads the next character from the keyboard and returns it).

```

#include <stdio.h>

char * readLine() {
    char buffer[1000];
    char c='\0';
    int index;
    for (index = 0; index < 1000 && c != '\n' ; index ++ ) {
        c = getchar();
        buff[index] = c;
    }
    return buffer;
}

```

Note: while having a fixed sized buffer is not good, it is acceptable for this problem (i.e. not what is wrong) as long as the buffer is not overflowed.

1. When the above code is compiled, the compiler produces the following warning:

```

stupid.c: In function 'readLine':
stupid.c:11: warning: function returns address of local variable

```

Briefly explain what this message means, why returning the address of a local variable is problematic, and what function (that we discussed in lecture) should be used instead to guarantee correct behavior.

Answer: (9 points)

This message appears because `buffer` is an array allocated on the stack. Since this memory is deallocated and re-used by other functions when `readLine` returns, the data pointed to by `buffer` (and thus the return value of the function) can be overwritten before it is used. The correct way to write this function involves the use of `malloc`.

2. When the previous problem is remedied, the function still does not work correctly. If the programmer attemptst to do

```
char * s = readLine();
printf("Input was %s", s);
```

then when the user enters the input `Hi`, the program prints

```
Input was Hi
X??@#4A8234?adsfjk
```

Briefly explain the remaining error in the code, and how to fix it.

Answer: (9 points)

The remaining problem is that the string is not null-terminated. I.e. The `readLine` function does not place a `'\0'` in the buffer after it finishes reading input. The junk characters printed are whatever happened to be in buffer until a null terminator character was found. The way to correct the code is to make sure that the loop terminates with at least 1 space left in buffer, and then manually insert `'\0'` after the last character read.

Question 4 (40 points): Consider the following code:

```
int * foo (int i, int * p1, int * p2) {
    /* i is at 124, p1 is at 128, and p2 is at 132 */
    /* Location 2 */
    while(i) {
        i--;
        p1[i] = p2[i];
    }
    /* Location 3*/
    return p2 + 2; /* be careful... */
}

int a[3]; /* a[0] is at 100 */
int b = 2; /* b is at 112 */
int * x; /* x is at 116*/
```

```

int * p = malloc (2 * sizeof (*p)); /* p is at 120 */
a[0] = 5; a[1] = 7 a[2] = 42;
/* Location 1 */
x = foo (b, p, a);
/* Location 4 */
x[0] = b;
x[1] = 300;
/* Location 5 */

```

Fill in the following table indicating the values of each variable/memory location at each marked program point above. Some boxes are filled in for you. 1 point per box

	Loc 1	Loc 2	Loc 3	Loc 4	Loc 5
100-103	5	5	5	5	5
104-107	7	7	7	7	7
108-111	42	42	42	42	2
112-115	2	2	2	2	300
116-119	???	???	???	108	108
120-123	400	400	400	400	400
124-127	???	2	0	???	???
128-131	???	400	400	???	???
132-135	???	100	100	???	???
400-403	???	???	5	5	5
404-407	???	???	7	7	7

The program above exhibits behavior which is likely not what the programmer intended, and certainly is not immediately apparent at first glance. Which line of code causes this strange behavior, and why does it happen?

Answer: (4 points)

The line `x[1] = 300` changes the value of `b`, which is probably a bug. Furthermore, even if the programmer did intend to change `b` on this line, assigning to `b` is the only correct way to do so, since other compilers may lay out the stack frame differently, and a person reading the code would understand it more easily