

Homework Assignment 2

CSE 399 C++, Spring 2008

SOLUTIONS

Name:

Due: Wednesday, Jan 30th 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:

```
char c = 'A';
char * p = &c;
char ** p2 = &p;
void * v = &p2;
```

Examine each of the following expressions. If the expression is illegal, write ILLEGAL. If the expression is legal, write its type (i.e. `int **` or `unsigned long` etc):
2 points each

- `&p2` `char ***`
- `p2[2]` `char *`
- `p + 4` `char *`
- `&p2[4]` `char **`
- `v[4]` `ILLEGAL`

Question 2 (5 points): There is an old C programmer's joke which goes as follows:

Two strings walk into a bar. The first one says

```
Hi I'd like a beer.A2%asd$ASDlk2;3423Ammm.234ASDfmmlASDFLJ:#@$
```

The second says

```
You'll have to pardon my friend, he's not null terminated.
```

Explain the joke:

Answer: (5 points)

In C, there is no special data type for strings— they are just a sequence of characters terminated by the special null character ('\0'). If text is placed in a string with no null terminator, then anything that uses the string will keep reading memory until it happens across a random '\0'. The joke is then based on this fact with the first string having random garbage after its intended contents.

Question 3 (32 points): Given the following code:

```
int x = 42;          /* x is at address 100 */
int y = 13;         /* y is at address 104 */
int * p;           /* p is at address 108 */
int ** p2;        /* p2 is at address 112 */
/* Location 1 */
p = &y;
p2 = &p;
/* Location 2 */
*p2 = &x;
**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	42	42	11	12
y	13	13	13	13
p	??	104	100	100
p2	??	108	108	108

Question 4 (13 points): Consider the following code:

```
int * foo(int x) {  
    int a[2];  
    a[0] = x; a[1] = x+1;  
    return a;  
}
```

...

```
int * p1 = foo(2);  
int * p2 = foo(3);
```

At the conclusion of this code snippet, what are the values of each of the following (2 pts each):

- p1[0] = 3
- p1[1] = 4
- p2[0] = 3
- p2[1] = 4

Explain why (5 pts):

Answer: (5 points)

Since `a` is allocated on the stack, the memory does not remain allocated when the call returns and may be reused by something else. In this case, since `foo(3)` is called immediately after `foo(2)` at the same call depth (i.e. they use the same stack space), they will both return the same pointer— i.e. `p1` is equal to `p2`. Since `p1` and `p2` point to the same memory, `p1[0]` and `p2[0]` are the same (likewise for `p1[1]` and `p2[1]`).

Question 5 (35 points): Consider the following code:

```
void * copy(int * dst, int * src, int count) {
    /* dst is at 124, src is at 128, count is at 132 */
    /* Location 2*/
    while (count) {
        count --;
        dst[count] = src[count];
    }
    /* Location 3 */
    return src + 1; /* 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] = 9; a[1] = 22; a[2] = 112;
x = &a[1];
/* Location 1 */
x[0] = 33;
x[1] = 99;
x = copy (p, x, b);
/* Location 4 */
x[-1] = 4;
x[0] = 5;
/* 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	9	9	9	9	9
104-107	22	33	33	33	4
108-111	112	99	99	99	5
112-115	2	2	2	2	2
116-119	104	104	104	108	108
120-123	400	400	400	400	400
124-127	???	400	400	???	???
128-131	???	104	104	???	???
132-135	???	2	0	???	???
400-403	???	???	33	33	33
404-407	???	???	99	99	99

Question 6 (5 pts): Consider the following code:

```
int a[2];
int b;
int i;
b = 5;

for (i = 0; i <= 2; i++) {
    a[i] = i;
}
printf("b is %d\n", b);
```

When the program is run, the value of `b` mysteriously changes from 5 to 2, even though no assignment is made to `b`. Explain why this occurs:

Answer: (5 points)

The for loop modifies 3 elements of the array `a`: 0, 1, and 2, however, `a` only has space allocated for 2 elements (0 and 1). `b` happens to be right past the end of `a`, so the attempt to change `a[2]` results in changing the value of `b`. Another way to think of this is that if `a[0]` is at address 100 and `b` is at address 108, then `a[2]` is also at address 108.