CS 61C Fall 2018 — Memory Management Review

August 28, 2018

Notes

1 Let's Test Your Memory

1.1 Faulty Memory

Find what is wrong with each function and rewrite the function to return what it specifies in the comments.

```
int * arrayRange(int n){
    //Returns an array where arr[i] = i
    int arr[n];
    int i = 0;
    for (i = 0; i < n; i++){
        arr[i] = i;
    }
    return arr;
}
```

```
#include <string.h>
char * reverse(){
    //Returns the string "I LOVE CS61C!" reversed
    char *str = "I LOVE CS61C!";
    int mid = strlen(str)/2;
    int i = 0;
    char tmp;
    for(i=0; i < mid; i++ ){
        tmp = str[i];
        str[i] = str[strlen(str) - i - 1];
        str[strlen(str) - i - 1] = tmp;
    }
    return str;
}
```

1.2 Finding Waldo

```
int main() {
    char * waldo = "Here I am.";
    char * vik = (char *) malloc(sizeof(char) * 17);
    strcpy(vik ,"HI I'M not waldo");
    char al[] = "HI i'm al";

    char * house[4];

```

```
    *(house) = vik;
    *(house + 1) = waldo;
    *(house + 2) = "my name is Vin";
    *(house + 3) = al;

    return 0;
}
```

Find where in memory each expression points to.

(a) house[0]

(b) house

(c) &house

(d) house[1]

(e) house[2]

(f) house[3]

2 Assemble the Bear Stack

2.1 Consider the C code below. Assign the result of evaluating each C expression from numbers 1 to 4, based on the C memory model taught in class.

```
#include<stdlib.h>
#include<stdio.h>

typedef struct Bear {
    char* name;
    struct Bear* brothers;
} Bear;

int main(int argc, char const *argv[]) {
    Bear* weBareBears = (Bear*) malloc(sizeof(Bear) * 3);
    for (int i = 0; i < 3; i++) {
        switch(i) {
            case 0:
                weBareBears[i].name = "Grizz"; break;
            case 1:
                weBareBears[i].name = malloc(sizeof(char) * 4);
                weBareBears[i].name[0] = 'I';
                weBareBears[i].name[1] = 'c';
                weBareBears[i].name[2] = 'e';
                weBareBears[i].name[3] = '\0';
                break;
             case 2: weBareBears[i].name = "Panda"; break;
        }
        printf("%s", weBareBears[i].name);
```

```
25          weBareBears[i].brothers = weBareBears;
26      }
27  }
```

(a) `&weBareBears`

(b) `&weBareBears[0].name[0]`

(c) `&weBareBears[1].name[0]`

(d) `&weBareBears[2].name[0]`

3 Heaps of Fun

3.1 HashTables are very useful data structures, and we want to implement one in C. Fill in the `newTable`, `addEntry`, `deleteEntry`, `resizeTable` and `freeTable` functions. Assume a function `int hashCode(int key)` is defined for you. You may define any helper functions you would like. For allocation failures, assume all you must do is print an error statement.

```
1   #include <stdlib.h>
2   #include <stdio.h>
3
4   typedef struct Entry {
5       int key;
6       int value;
7       Entry* nextEntry;
8       } Entry
9
10  typedef struct Table {
11      size_t size;
12      Entry* buckets;
13  }
14
15  Table *newTable(size_t initialSize) {
16
17
18
19
20
21  }
22  void addEntry(int key, int value, Table *table) {
23      int index = hashCode(key) %  table->size;
24
25
26
27
28
29
```

```

}

void deleteEntry(int key, Table *table) {
    int index = hashCode(key) % table->size;









}



void freeTable(Table *table) {




```

```
76
77
78
79
80
81
82
83
84
85
86      }
87
88  void resizeTable(Table *table, size_t newSize) {
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104 }
```