CS 240 SI Worksheet
Valerie Zhao
Session #10
3/7/17

Pointers and Memory Allocation

1. Fill in the following table, given that:

```
int** ben = (int**) 0x100C
char** ash = (char**) 0x100C     //str
```

Data is stored in **little endian** (like in the x86), and each address is **4 bytes**.
**note that within each byte, same order; lsbyte in low addr, each addr maps to 1 byte*
(Both tables are parts of the heap.)

Address	**Content (in hex)**
0x1020	CA 03 26 8E
0x101C	00 00 31 44
0x1018	2C 38 95 AB
0x1014	00 00 31 50
0x1010	00 00 31 40
0x100C	00 00 31 48

Address	**Content (in hex)**
0x3154	28 19 0C D0
0x3150	8C 9B AD 0C
0x314C	74 9C DF 20
0x3148	BB 2C 08 92
0x3144	37 D7 99 0C
0x3140	04 29 3A B6

	Type	**Numeric Value**
`&ash[1]`	**char****	**0x 00 00 10 10**
`*ben`	**int***	**0x 00 00 31 48**
`*(ash[2])`	**char**	**0x 0C**
`*(ben-2)`	**int***	**unknown**
`(int**) ash - ben`	**ptrdiff_t**	**0**
`(**ash) + 3` **`(if **(ash + 3): need scale, because explicit deref)`**	**char**	**0x 95**
`sizeof(ben)`	**size_t**	**4**

2. Write a function that would, for each pointer in an array, allocate space for a 3-char word using **pointer arithmetic**:
   (Pretend that we don't want to keep the pointers to the malloc locations.)

```
void printWords(char** jean) {
      char** first = jean;
      while (*first != NULL) { // char* isn't null
            char* loc = (char*)malloc(sizeof(char)*3);
            first++;
      }
}
```

   Note that in actual programming this is really dangerous to do, since we're discarding the pointers to the space allocated by malloc: a memory leak will result because we have no way of freeing the allocated space afterwards.