CSC 209 Review 8 Solution

August 28, 2020

1. I need to create a wrapper function `my_malloc` that does the following:

   - ask `my_malloc` it to allocate `n` bytes
   - call `malloc`
   - test `malloc` doesn't have a null pointer
   - return pointer from `malloc`

   The solution to this problem is:

```c
void *my_malloc(int n) {
    void *p;

    p = malloc(n);

    if (!p) {
        printf("ERROR: Malloc allocation failed");
    }

    return p;
}
```

   Notes

   - Learned that void function can return value
   - **Dynamic Storage Allocation**
     - Allows to allocate storage during program execution
     - Allows to create data structures and shink and grow array as needed
     - e.g. `malloc`, `calloc`, `realloc`
   - **Memory Allocation Functions**
     - `malloc` - Allocates a block of memory but doesn't initialize it
       - doesn't initialize the allocated memory

    - more efficient than `calloc`
    - accessing the content → segmentation fault (accessing value at invalid mem. location) or garbage values
  - `calloc` - Allocates a block of memory and clears it
    - allocates memory and initializes the memory block to zero
    - accessing the content of blocks would return 0
  - `realloc` - Resizes a previously allocated block of memory
- **Null Pointer**
  - is returned when it fails to allocate a block of memory large enough to satisfy the request

Example

```
p = malloc(10000);
if (p == NULL) {
  /* allocation failed; take appropriate action */
}
```

2. I need to write a function named `duplicate` that uses dynamic storage allocation to create a copy of a string.

   The requirements of the function are

   - `duplicate` allocates space for a string of the same length as `str`
   - `duplicate` copies the contents of str into the new string
   - `duplicate` returns a pointer to it
   - `duplicate` returns a null pointer if the memory allocation fails

   The solution to this problem is:

```c
#include <stdio.h>
#include <stdlib.h>  // malloc
#include <string.h>  // strlen

char *duplicate(const char *str);

int main(void) {
    char s[] = "hello world", *p;

    p = duplicate (s);
```

```
11
12          printf("Duplicate: %s\n", p);
13
14          free(p);
15          return 0;
16      }
17
18
19      char *duplicate(const char *str) {
20          char *p, *q;
21          const char *r;
22
23          int n = strlen(str);
24
25          p = (char *)malloc(n + 1);
26
27          if (!p) {
28              return p;
29          }
30
31          r = str;
32          q = p;
33          while (r < str + n) {
34              *q = *r;
35              q++;
36              r++;
37          }
38
39          *q = '\0';
40
41          return p;
42      }
```

Correct Solution:

```
1       #include <stdio.h>
2       #include <stdlib.h>  // malloc
3       #include <string.h>  // strlen
4
5       char *duplicate(const char *str);
6
7       int main(void) {
8           char s[] = "hello world", *p, *q;
9           ;
10          p = duplicate (s);
11
12          printf("Duplicate: %s\n", p);
13
14          free(p);
15          return 0;
16      }
17
```

```
18
19    char *duplicate(const char *str) {
20        char *p, *q;
21        const char *r;
22
23        int n = strlen(str);
24
25        p = (char *)malloc(n + 1);
26
27        if (!p) {
28            p = ((void*)0);
29            return p;
30        }
31
32        r = str;
33        q = p;
34        while (r < str + n) {
35            *q = *r;
36            q++;
37            r++;
38        }
39
40        *q = '\0';
41
42        return p;
43    }
```

Note

- Null pointer has value `((void*)0)`
- `const` tag in parameter prevetns the function from modifying what it's pointer variable is pointing to.
  - value is modifiable
  - changes the parameter to pass by value

3.
```
1    int *create_array(int n, int initial_value) {
2
3    }
```