

CSC 209 Review 8 Solution

September 11, 2020

```
1  int *my_malloc (int n) {
2      int *res;
3
4      res = malloc(n * sizeof(int));
5      if (res == NULL) {
6          perror("Allocation failed.");
7      }
8
9      return res;
10 }
```

Please see question_1.c for details.

```
2  char *duplicate(char *str) {
3      char *res;
4
5      res = malloc(strlen(str) + 1);
6      if (res == NULL) {
7          return res;
8      }
9
10     strcpy(res, str);
11
12     return res;
13 }
```

Please see question_2.c for details.

```
3  int *create_array(int n, int initial_value) {
4      int *p, *res;
5
6      res = malloc(n * sizeof(int));
7
8      if (res == NULL) {
9          return res;
10     }
11 }
```

```
10     for (p = res; p < res + n; p++) {
11         *p = initial_value;
12     }
13
14     return res;
15 }
```

Please see `question_3.c` for details.

```
41 int main(void) {
2     struct point {int x, y};
3     struct rectangle {struct point upper_left, lower_right};
4     struct rectangle *p;
5
6     p = malloc(sizeof(struct rectangle));
7
8     p->upper_left.x = 10;
9     p->upper_left.y = 25;
10
11     p->lower_right.x = 20;
12     p->lower_right.y = 15;
13
14     printf("%d %d\n", p->upper_left.x, p->upper_left.y);
15     printf("%d %d\n", p->lower_right.x, p->lower_right.y);
16
17     free(p);
18
19     return 0;
20 }
21
22
```

Please see `question_4.c` for details.

5. b), c) and d) are legal.

Correct Solution

b), c) are legal.

Notes

- **The -> Operator**

- doesn't carry over to accessing nested members. Only works when struct is a pointer

Example

`p->upper_left.x`

```

61 struct node *delete_from_list(struct node *list, int n)
62 {
63     struct node *cur = list, *temp;
64
65     if (cur->value == n) {
66         list = cur->next;
67         return list;
68     }
69
70     for (cur = list; cur != NULL; cur = cur->next) {
71
72         if (cur->next != NULL && cur->next->value == n) {
73             break;
74         }
75     }
76
77     if (cur == NULL) {
78         return list;
79     }
80
81     temp = cur->next;
82     cur->next = cur->next->next;
83
84     free(temp);
85     return list;
86 }

```

7. It's incorrect because it's deleting the node before moving to next.

To fix this bug, p must move to the next node before removing the current.

```

1 struct node *temp;
2 p = first;
3 while (p != NULL) {
4     temp = p;
5     p = p->next
6     remove(temp);
7 }

```

8. Please see file `question_8/stack.h`, `question_8/stack.c`, `question_8/calc.c` for details.

9. True.

By definition, $(\&x) \rightarrow a$ is the same as $(*(\&x)).a$.

Since $(*(\&x)) = x$, we can write $(\&x) \rightarrow a$ is the same as $x.a$.