CSC 209 Review 8 Solution

September 11, 2020

1.
```c
int *my_malloc (int n) {
    int *res;

    res = malloc(n * sizeof(int));
    if (res == NULL) {
        perror("Allocation failed.");
    }

    return res;
}
```

Please see `question_1.c` for details.

2.
```c
char *duplicate(char *str) {
    char *res;

    res = malloc(strlen(str) + 1);
    if (res == NULL) {
        return res;
    }

    strcpy(res, str);

    return res;
}
```

Please see `question_2.c` for details.

3.
```c
int *create_array(int n, int initial_value) {
    int *p, *res;

    res = malloc(n * sizeof(int));

    if (res == NULL) {
        return res;
    }

```

```
        for (p = res; p < res + n; p++) {
            *p = initial_value;
        }

        return res;
    }
```

Please see `question_3.c` for details.

4.
```
        int main(void) {
            struct point {int x, y};
            struct rectangle {struct point upper_left, lower_right};
            struct rectangle *p;

            p = malloc(sizeof(struct rectangle));

            p.upper_left.x = 10;
            p.upper_left.y = 25;

            p.lower_right.x = 20;
            p.lower_right.y = 15;

            printf("%d %d\n", p.upper_left.x, p.upper_left.y);
            printf("%d %d\n", p.lower_right.x, p.lower_right.y);

            free(p);

            return 0;
        }


```

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`

6.
```c
struct node *delete_from_list(struct node *list, int n)
{
    struct node *cur = list, *temp;

    if (cur->value == n) {
        list = cur->next;
        return list;
    }

    for (cur = list; cur != NULL; cur = cur -> next) {

        if (cur->next != NULL && cur->next->value == n) {
        break;
        }
    }

    if (cur == NULL) {
        return list;
    }

    temp = cur->next;
    cur->next = cur->next->next;

    free(temp);
    return list;
}
```

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.

```c
struct node *temp;
p = first;
while (p != NULL) {
    temp = p;
    p = p -> next
    remove(temp);
}
```