

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

```
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)
2   {
3       struct node *cur = list, *temp;
4
5       if (cur->value == n) {
6           list = cur->next;
7           return list;
8       }
9
10      for (cur = list; cur != NULL; cur = cur->next) {
11
12          if (cur->next != NULL && cur->next->value == n) {
13              break;
14          }
15      }
16
17      if (cur == NULL) {
18          return list;
19      }
20
21      temp = cur->next;
22      cur->next = cur->next->next;
23
24      free(temp);
25      return list;
26  }

```

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 $x.a$.

```

101 void print_part(struct part *p)
2   {
3       printf("Part number: %d\n", p->number);
4       printf("Part name: %s\n", p->name);
5       printf("Quantity on hand: %d\n", p->on_hand);
6   }

```

11. Please see `question_11.c` for details.

```
12_1 struct node {
2_     int value;
3_
4_ }
5_
6_ void print_part(struct part *p)
7_ {
8_     printf("Part number: %d\n", p->number);
9_     printf("Part name: %s\n", p->name);
10_    printf("Quantity on hand: %d\n", p->on_hand);
11_ }
```

```
13_1 struct node {
2_     int value;
3_     struct node *next;
4_ };
5_
6_ struct node *find_last(struct node *list, int n)
7_ {
8_     struct node *res = NULL, *p;
9_
10_    for (p = list; p != NULL; p = p->next) {
11_        if (p->value == n) {
12_            res = p;
13_        }
14_    }
15_
16_    return res;
17_ }
```

Please see file `question_12.c` for details.

```
14_1 struct node {
2_     int value;
3_     struct node *next;
4_ };
5_
6_ struct node *insert_into_ordered_list(struct node *list, struct node
7_ *new_node)
8_ {
9_     struct node *cur = list, *prev = NULL;
10_
11_    if (list == NULL) {
12_        list = new_node;
13_        return list;
14_    }
15_
16_    while (cur != NULL && cur->value <= new_node->value) {
17_        prev = cur;
18_        cur = cur->next;
19_    }
```

```
19     prev->next = new_node;
20     new_node->next = cur;
21     return list;
22 }
23
```

```
15 struct node *delete_from_list(struct node *list, int n)
16 {
17     struct node *cur, *prev;
18
19     for (cur = list, prev = NULL;
20         cur != NULL && cur->value != n;
21         prev = cur, cur = cur->next)
22
23         ;
24
25     if (curr == NULL) {
26         return list;
27     }
28
29     if (prev == NULL) {
30         list = list->next;
31     } else {
32         prev->next = cur->next;
33     }
34
35     free(cur);
36     return list;
37 }
```