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;
    }

```

```
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)->a` is the same as `(*(&x)).a`.

   Since `(*(&x)) = x`, we can write `(&x)->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.
```c
struct node {
    int value;
    struct node *next;
};

struct node *find_last(struct node *list, int n)
{
    struct node *res = NULL, *p;

    for (p = list; p != NULL; p = p->next) {
        if (p->value == n) {
            res = p;
        }
    }

    return res;
}
```

Please see file `question_12.c` for details.

13.
```c
struct node {
    int value;
    struct node *next;
};

struct node *insert_into_ordered_list(struct node *list, struct node
*new_node)
{
    struct node *cur = list, *prev = NULL;

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

    while (cur != NULL && cur->value <= new_node ->value) {
        prev = cur;
        cur = cur->next;
    }

    prev->next = new_node;
    new_node->next = cur;
    return list;
}
```

14.
```c
struct node *delete_from_list(struct node **list, int n)
{
    struct node *cur, *prev;

    for (cur = *list, prev = NULL;
            cur != NULL && cur->value != n;
```

```
 7                          prev = cur , cur = cur ->next)
 8
 9                              ;
10
11              if (cur == NULL) {
12                  return (*list);
13              }
14
15              if (prev == NULL) {
16                  *list = (*list)->next;
17              } else {
18                  prev->next = cur->next;
19              }
20
21              free(cur);
22              return (*list);
23          }
```

Please see `question_14.c` for details.

15. It returns the value of `n` that is equal to `i * i + i - 12`.

Here, the value of `n` is 3.

Please see `question_15.c` for details.

```
16  int sum(int (*f)(int), int start , int end)
 2  {
 3      int res = 0;
 4      for (int i = start; i <= end; i++) {
 5          res += (*f)(i);
 6      }
 7
 8      return res;
 9  }
```

Please see `question_16.c` for details.

```
17  qsort(&a[50], 50, sizeof(a[0]), compare_parts);
```

```
18  int compare_parts(const void *p, const void *q)
 2  {
 3      return ((struct part *) q)->number - ((struct part *) p)->number
    ;
 4  }
```

```
19  struct {
 2      char *cd_name;
 3      void (*cmd_pointer)(void);
 4  } file_cmd[] = {
```

```
 5              {"new", new_cmd},
 6              {"open", open_cmd},
 7              {"close", close_cmd},
 8              {"close all", close_all_cmd},
 9              {"save", save_cmd},
10              {"save as", save_as_cmd},
11              {"save all", save_all_cmd},
12              {"print", print_cmd},
13              {"exit", exit_cmd}
14      } ;
15
16
17      void run_cmd(const char *cmd)
18      {
19          int cmd_cnt = sizeof(file_cmd)/sizeof(file_cmd[0]);
20          char cmd_cpy[21], *p;
21
22          strcpy(cmd_cpy, cmd);
23          if (strlen(cmd) == 21) {
24              cmd_cpy[20] = '\0';
25          }
26
27          for (p = cmd_cpy; *p != '\0'; p++)
28          {
29              *p = tolower(*p);
30          }
31
32          for (int i = 0; i < cmd_cnt; i++) {
33              if (strcmp((file_cmd[i]).cmd_name, cmd_cpy) == 0) {
34                  return (file_cmd[i]).cmd_pointer();
35              }
36          }
37      }
38
39      void new_cmd(void)
40      {
41          printf("new_cmd\n");
42      }
43      void open_cmd(void)
44      {
45          printf("open_cmd\n");
46      }
47      void close_cmd(void)
48      {
49          printf("close_cmd\n");
50      }
51      void close_all_cmd(void)
52      {
53          printf("close_all_cmd\n");
54      }
55      void save_cmd(void)
56      {
57          printf("save_cmd\n");
58      }
```

6

```
59    void save_as_cmd(void)
60    {
61        printf("save_as_cmd\n");
62    }
63    void save_all_cmd(void)
64    {
65        printf("save_all_cmd\n");
66    }
67    void print_cmd(void)
68    {
69        printf("print_cmd\n");
70    }
71    void exit_cmd(void)
72    {
73        printf("exit_cmd\n");
74    }
```

Please see `question_19.c` for details.

20. Please see `question_20.c` for details.

21. Please see `question_21.c` for details.

Notes

- qsort
    - is a part of <stdlib> library

22. Please see `question_22.c` for details

23. Please see `question_23.c` for details