CSC 209 Review 4 Solution

August 19, 2020

1. The answer is `a) *p` and `g) *&i`.

   Notes

   - **Address and Indirection Pointers**
     - If `x` is a variable, `&x` points to its memory address
     - * in `*p` is called **Indirection operator**
       - Allows variable to gain access to the object pointed by `p`
   - **Aliases**
     - Is the situation where the value in same memory location can be accessed using different variable names.

       Example 1:

       ```
       int i, p*;
       p = & i;
       printf("%d\n", *p); /* *p is an alias of i */
       ```

       Example 2:

       ```
       int i, p*;
       p = *&i /* *p is an alias of i */
       ```

2. The answers are `b) *p = &i;`, `f) p = q;`, and `i) *p = *q;`

   Correct Solution

   The answers are `e) p = *&q;`, `f) p = q;`, and `i) *p = *q;`

   `p = *&q;` is the same as `p = q`

Notes

- The `*` operator turns a *value* of type `pointer to T` into a *variable* of type `T`.
- The `&` operator turns a *variable* of type `T` into a *value* of type pointer to `T`.
- **Pointer Assignment**
  - The following is an example of correct pointer assignment

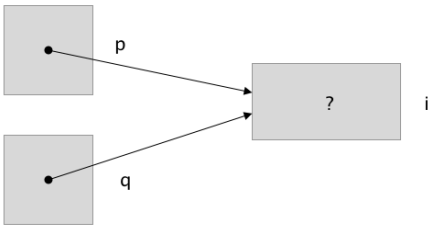
```
int i, j, *p. *q;
p = &i;
```

    * Means the memory address of `p` is pointing to memory address of `i`

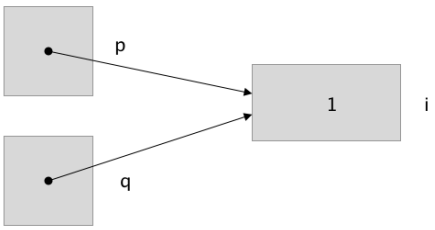
  - The following is another valid example of pointer assignment

```
int i, j, *p. *q;
p = &i;
q = p;
```

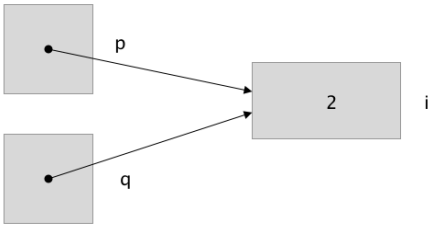
    * Means memory address of `q` is the memory address of `p` (which is the memory address of `i`)




```
*p = 1;
```



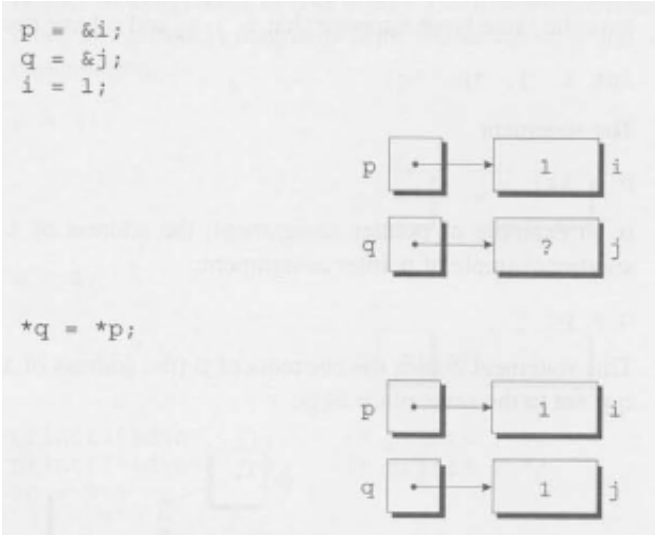

```
*p = 2;
```



– The following is not a pointer assignment

```
*q = *p
```

  * It copies the value that `p` points to



3.
```
1  void avg_sum(double a[], int n, double *avg, double *sum)
2  {
3      int i;
4
5      *sum = 0.0;
6      for (i = 0; i < n; i++)
7          *sum += a[i];
8      *avg = *sum / n;
9  }
```

Notes:

- **Pointer as Arguements:**
  – Construct protype using pointer variable as parameter so it can be passed by refernce

  Example

```
void decompose(double x, long *int_part, double *frac_part);

or

void decompose(double, long *, double *);
```

```
void decompose(double x, long *int_part, double *frac_part)
{
  *int_part = (long) x;
  *frac_part = x - *int_part;
}
```

– When using the prototype, pass variable to prototype by reference using & operator (points to variable's memory location)

```
decompose(3.14159, &i, &d);
```

4.
```
void swap(int *p, int *q) {
    int temp;

    temp = *p;
    *p = *q;
    *q = temp;
}
```

5.
```
void split_time(long total_sec, int *hr, int *min, int *sec) {
    int hours, mins, seconds, min_sec;

    hours = total_sec % 60;
    min_sec = total_sec - hours;
    mins = min_sec % 60;
    seconds = min_sec - mins;

    *hr = hours;
    *min = mins;
    *sec = seconds;
}
```

Correct Solution:

```
void split_time(long total_sec, int *hr, int *min, int *sec) {
    int hours, mins, seconds, min_sec;

    hours = total_sec % 3600;
    min_sec = total_sec - (hours * 3600);
    mins = min_sec % 60;
    seconds = min_sec - (mins * 60);

    *hr = hours;
    *min = mins;
    *sec = seconds;
}
```

6.
```
#include <stdbool.h> // bool
#include <limits.h>  // INT_MIN

bool is_largest(int current_max, int val);

void find_two_largest (int a[], int n, int *largest, int*
second_largest) {
```

```
        int current_max = INT_MIN;
        int current_second_max = INT_MIN;

        for (int i = 0; i < n; i++) {
            if (is_largest(current_max, a[i])) {
                current_second_max = current_max;
                current_max = a[i];
            }
        }

        *largest = current_max;
        *second_largest = current_second_max;
    }

    bool is_largest(int current_max, int val) {
        if (val > current_max) {
            return true;
        }

        return false;
    }
```

7. From calendar, we can see that

   - January, March, May, July, August, October and December have 31 days
   - February has 28 days (Assuming it has no leap year)
   - The rest (April, June, September and November) have 30 days

   Using this knowledge, we have

```
    void split_date (int day_of_year, int year, int *month, int *day) {
        int days_in_month;

        for (let i = 1; i < 13; i++) {
            *month = i

            if (i == 1 ||
                i == 3 ||
                i == 5 ||
                i == 7 ||
                i == 8 ||
                i == 10 ||
                i == 12
            ) {
                days_in_month = 31;
            } else if (
                i == 4 ||
                i == 6 ||
                i == 9 ||
                i == 11
            ) {
```

```
22                  days_in_month = 30;
23              } else {
24                  days_in_month = 28;
25              }
26
27              day_of_year -= days_in_month;
28
29              if (day_of_year < 0) {
30                  break;
31              }
32          }
33
34          *day = day_of_year + days_in_month;
35      }
```

8.

```
1       int *find_largest(int a[], int n) {
2           int curr_max = a[0];
3           int i_max = 0;
4
5           for (int i = 0; i < n; i++) {
6               if (a[i] > curr_max) {
7                   curr_max = a[i];
8                   i_max = i;
9               }
10          }
11
12          return &a[i_max];
13      }
```

Notes

- **Pointers as Return Values**
  - Must return one of parameter's value of type pointer to T as return value

    Example

```
1       int *max(int *a, int *b) {
2           if (*a > *b)
3               return a;
4           else
5               return b;
6       }
7
8       ...
9
10      int *p, i, j;
11      p = max(&i, &j);
```

    Example 2

```
int *find_middle(int a[], int n) {
    return &a[n/2];
}
```

– Never return a pointer to an `automatic` local variable

Example

```
int *f(void) {
    int i;
    ...
    return &i;
}}
```

* Because variable `i` doesn't exist once `f` returns.