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**
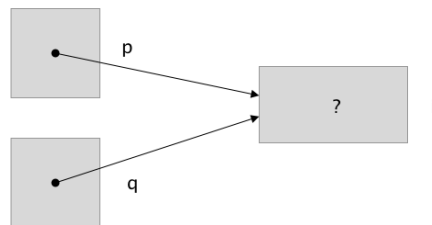   - The following is an example of correct pointer assignment

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

        * Means the memory address of `p` is pointing to memory address of `i`
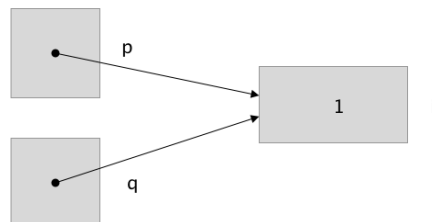
    - The following is another valid example of pointer assignment

      ```
      int i, j, *p. *q;
      p = &i;
      q = p;
      ```
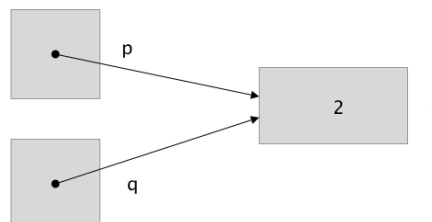
        * Means memory address of `q` is the memory address of `p` (which is the memory address of `i`)
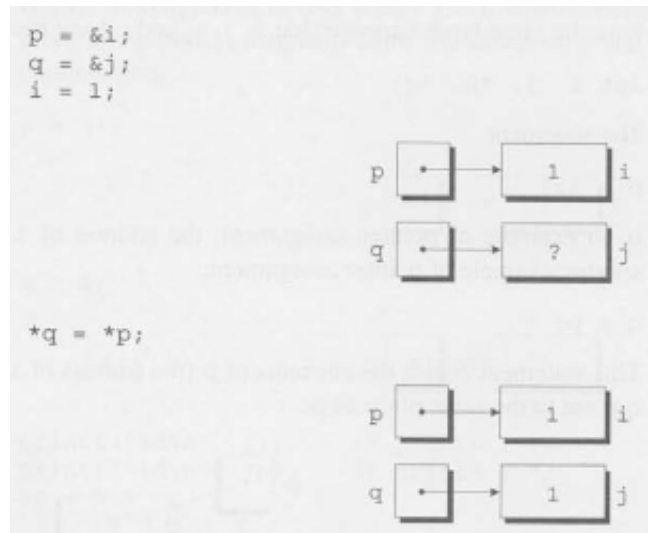


      ```
      *p = 1;
      ```



      ```
      *p = 2;
      ```

– The following is not a pointer assignment

`*q = *p`

* It <u>copies the value that</u> **p** points to

```
p = &i;
q = &j;
i = 1;
```



```
*q = *p;
```



```
3  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  }
```

<u>Notes:</u>

• **Pointer as Arguements:**

– Construct protype using pointer variable as parameter so it can be passed by refernce

<u>**Example**</u>

```
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;
}
```

3

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

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

4.
```
1    void swap(int *p, int *q) {
2        int temp;
3
4        temp = *p;
5        *p = *q;
6        *q = temp;
7    }
```

5.
```
1    void split_time(long total_sec, int *hr, int *min, int *sec) {
2        int hours, mins, seconds, min_sec;
3
4        hours = total_sec % 60;
5        min_sec = total_sec - hours;
6        mins = min_sec % 60;
7        seconds = min_sec - mins;
8
9        *hr = hours;
10       *min = mins;
11       *sec = seconds;
12   }
```

Correct Solution:

```
1    void split_time(long total_sec, int *hr, int *min, int *sec) {
2        int hours, mins, seconds, min_sec;
3
4        hours = total_sec % 3600;
5        min_sec = total_sec - (hours * 3600);
6        mins = min_sec % 60;
7        seconds = min_sec - (mins * 60);
8
9        *hr = hours;
10       *min = mins;
11       *sec = seconds;
12   }
```

6.
```
1    #include <stdbool.h> // bool
2    #include <limits.h>  // INT_MIN
3
4    bool is_largest(int current_max, int val);
5
6    void find_two_largest (int a[], int n, int *largest, int*
     second_largest) {
```

```
7            int current_max = INT_MIN;
8            int current_second_max = INT_MIN;
9
10           for (int i = 0; i < n; i++) {
11               if (is_largest(current_max, a[i])) {
12                   current_second_max = current_max;
13                   current_max = a[i];
14               }
15           }
16
17           *largest = current_max;
18           *second_largest = current_second_max;
19       }
20
21       bool is_largest(int current_max, int val) {
22           if (val > current_max) {
23               return true;
24           }
25
26           return false;
27       }
```

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

```
1        void split_date (int day_of_year, int year, int *month, int *day) {
2            int days_in_month;
3
4            for (let i = 1; i < 13; i++) {
5                *month = i
6
7                if (i == 1 ||
8                    i == 3 ||
9                    i == 5 ||
10                   i == 7 ||
11                   i == 8 ||
12                   i == 10 ||
13                   i == 12
14               ) {
15                   days_in_month = 31;
16               } else if (
17                   i == 4 ||
18                   i == 6 ||
19                   i == 9 ||
20                   i == 11
21               ) {
```

```
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  }
```

```
8₁  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  }
```

<u>Notes</u>

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

    <u>Example</u>

```
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);
```

    <u>Example 2</u>

```
1    int *find_middle(int a[], int n) {
2        return &a[n/2];
3    }
```

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

Example

```
1    int *f(void) {
2        int i;
3        ...
4        return &i;
5    }}
```

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

9. Please referr to file `question_9.c`.