CSC 209 Review 5 Solution

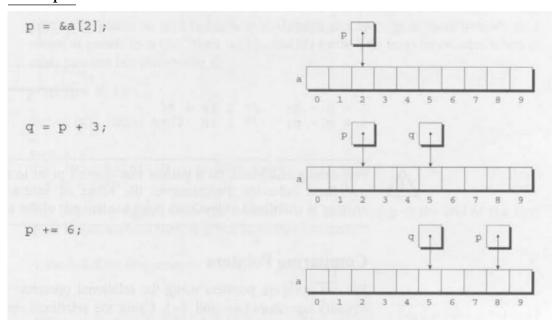## August 23, 2020

1 Exercises

1. a) 14

   b) 34

   c) 4

   d) true

   e) false

   Notes

   - **Pointer Arithematic**
     - Adding an integer to a pointer

     Example

– Subtracting an integer from a pointer

Example


```
p = &a[8];
```

```
q = p - 3;
```

```
p -= 6;
```

– Subtracting one pointer from another

Example


```
p = &a[5];
q = &a[1];
```

```
i = p - q;    /* i is 4 */
i = q - p;    /* i is -4 */
```

• **Comparing pointers**
  – Can compare pointers using relational operators (i.e. $<, <=, >, >=$) and the equality operators (i.e. $==, !=$)
  – Returns 1 if true and 0 if false

Example

```
p = &a[5];
q = &a[1];
```

```
p <= q is 0 and p >= q is 1
```

2. `low` and `high` are memory addresses.

   So, `low` + `high` is out of bound, and it could potentially point to an undesirable or wrong value.

   To fix this, we subtract the from high value to the low value:

   $$\texttt{middle} = \frac{\texttt{low} + \texttt{high}}{2} \tag{1}$$

3. I need to write the contents of an array `a` after the execution of statements outlined in problem sheet.

   After execution, the array would have contents of $[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$.

   <u>Notes</u>

   - **Combining the * and ++ Operators**
     - *p++ or *p++ → Value of expression is *p before increment; increment p later
     - (*p)++ → Value of expression is *p before increment; increment *p later
     - *++p or *(++p) → Increment p first; value of expression is *p after increment
     - ++*p or ++(*p) → Increment *p first; value of expression is *p after increment

   <u>Example</u>

   `a[i++] = j`

   Means assign the value `j` to `a[i]` before increment

   <u>Example 2</u>

   ```
   for (p = &a[0]; p < &a[N]; p++)
       sum += *p;
   ```

   Is the same as

   ```
   p = &a[0];
   while (p < &a[N])
       sum += *p++;
   ```

4. I need to re-write prototype `make_empty`, `is_empty` and `is_full` of the following code to use the pointer variable `top_ptr` instead of the integer variable `top`.

```
1   #include <stdbool.h>
2
3   #define STACK_SIZE 100
4
5   /*external variables*/
6   int contents[STACK_SIZE]
7   int top = 0;
8
9   void make_empty(void) {
10      top = 0;
11  }
12
13  bool is_empty(void) {
14      return top == 0;
15  }
16
17  bool is_full(void) {
18      return top == STACK_SIZE;
19  }
```

And after re-write using `top_ptr` instead of `top` have:

```
1   #include <stdbool.h>
2
3   #define STACK_SIZE 100
4
5   /*external variables*/
6   int contents[STACK_SIZE]
7   int *top_ptr = &contents[0];
8
9   void make_empty(void) {
10      top_ptr = &contents[0];
11  }
12
13  bool is_empty(void) {
14      return top_ptr == &contents[0];
15  }
16
17  bool is_full(void) {
18      return top_ptr == &contents[STACK_SIZE-1];
19  }
```

5. First, I need to identify which of the following expressions are illegal because of mismatched types.

    a) `p == a[0]`

    b) `p == &a[0]`

c) `*p == a[0]`

d) `p[0] == a[0]`

Here, only a) is illegal.

Second, I need to write which of the remaining expressions are true.

Here, the expressions that return true are b), c) and d).

Notes

- `*(a+i)` is equal to `a[i]`
- `*p` and `a[]` are the same given `p == a`
- **Using an Array Name as a Pointer**
  - The name of an array can be used as a pointer to the first element in the array.

    ### Example

    ```
    int a[10];

    *a = 7; /* stores 7 in a[0] */

    *(a+1) = 12; /* stores 7 in a[1] */
    ```

    ### Example 2

    To simplify the loop, we can replace `&a [0]` by `a` and `&a [N]` by `a + N`:

    ```
    for (p = a; p < a + N; p++)
        sum += *p;
    ```

6. I need to re-write the following to use pointer arithematic instead of array subscripting, and I need to make as few change as possible.

```
1    int sum_array(cost int a[], int n) {
2        int i, sum;
3
4        sum = 0;
5
6        for (i = 0; i < n; i++)
7            sum += a[i];
8        return sum;
9    }
```

After making changes to above code to use pointer arithematic, we have

```
1    int sum_array(cost int a[], int n) {
2        int i, sum;
3
4        sum = 0;
5
6        for (i = 0; i < n; i++)
7            sum += *(a+i);
8        return sum;
9    }
```

7. I need to write the following using pointer arithematic so it finds an element in `a` that matches to value `key`. I need to return true if there is a match.

   ```
   bool search(const int a[], int n, int key);
   ```

   And the solution is:

```
1    bool search(cost int a[], int n, int key) {
2
3        for (int i = 0; i < n; i++) {
4            if (*(a+i) == key) {
5                return true
6            }
7        }
8
9        return false;
10   }
```

8. Here, I need to re-write the following function to use pointer arithmetic instead of array subscripting.

```
1    void store_zeros(cost int a[], int n) {
2        int i;
3
4        for (i = 0; i < n; i++) {
5            a[i] = 0;
6        }
7    }
```

After re-writing above code, we have

```
1    void store_zeros(cost int a[], int n) {
2        int *p;
3
4        for (p = a; p < a + n; p++) {
5            *p = 0;
6        }
7    }
```

9. Here, I need to write the function

   ```
   double inner_product(const double *a, const double *b, int n)
   ```

   using pointer arithmetic such that it returns `a[0] * b[0] + a[1] * b[1] + a[2] * b[2] + ... + a[n-1] * b[n-1]`.

   The solution is provided below

   ```
1      double inner_product(const double *a, const double *b, int n) {
2          double sum = 0, p*;
3
4          p = a;
5
6          while (p < a + n) {
7              sum += *a * *b;
8
9              a++;
10             b++;
11             p++;
12         }
13
14         return sum;
15     }
   ```

   Correct Solution

   ```
1  double inner_product(const double *a, const double *b, int n) {
2      double sum = 0, *p;
3
4      p = a;
5
6      while (p++ < a + n) {
7          sum += *a++ * *b++;
8      }
9
10     return sum;
11 }
   ```

10. Here, I need to rewrite the function `find_middle` so that it uses pointer arithmetic - not subscripting - to visit any element.

    The solution to this exercise is provided below.

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

11. Here I need to modify the function `find_largest` function so that it uses pointer arithmetic - not subscripting - to visit array elements

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

12. I need to write the function

    void find_two_largest(const int *a, int n, int *largest, int *second_largest)

    using pointer arithmetic.

    The solution to this exercise is:

```
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      }
```

Correct Solution:

```
1    void find_two_largest(const int *a, int n, int *largest, int *
     second_largest) {
2
3         const int *p = a;
4         *largest = *second_largest = *a;
5
6         while (p++ < a + n) {
7             if (*p > *largest) {
8                 *second_largest = *largest;
9                 *largest = *p;
10            } else if (*p > *second_largest)
11                *second_largest = *p;
12        }
13   }
```

---

13. I need to rewrite the following program such that it uses a single pointer through the array one element at a time.

```
1    #define N 10
2
3    double ident[N][N];
4    int row, col;
5
6    for (row = 0; row < N; row++) {
7        for (col = 0; col < N; col++) {
8            if (row == col) {
9                ident[row][col] = 1.0;
10           } else {
11               ident[row][col] = 0.0;
12           }
13       }
14   }
```

The solution to this exercise is:

```
1    #include <stdio.h>
2    #include <stdbool.h>
3
4    #define N 10
5
6    bool is_row_eq_col(int i, int n);
7
8    int main() {
```

```
9
10          int i = 0;
11
12          double ident[N][N], *p;
13
14
15          for (p = &ident[0][0]; p < &ident[N-1][N-1]; p++) {
16              if (is_row_eq_col(i, N)) {
17                  *p = 1.0;
18              } else {
19                  *p = 0.0;
20              }
21              i++;
22          }
23
24
25          return 0;
26      }
27
28      bool is_row_eq_col(int i, int n) {
29          if (i % (n+1) != 0) {
30              return false;
31          }
32
33          return true;
34      }
```

Notes

- Learned that the memory address of each row in two dimensional array are placed right next to each other

14. I need to write a statement that uses the `search` function from Exercise 7 to search the entire `temperatures` array for thevalue 32.

    The solution to this problem is:

```
1      int row = 0;
2
3      for (int row = 0; row < 7; row++) {
4          if (search(temperatures[row], 24, 32)) {
5              return true
6          }
7      }
8
9      return false;
```

15. Create a loop that prints all temperature readings in row `i` of the following array using a pointer to visit each element of the row.

```
inte temperatures[7][24]
```

The solution to this problem is:

```
1      int *p;
2
3      for (p = &a[i]; p < &a[i] + 24; p++) {
4          printf("%d", *p);
5      }
```

Correct Solution:

```
1       int *p;
2
3       for (p = &temperatures[i]; p < &temperatures[i] + 24; p++) {
4           printf("%d", *p);
5       }
```

16. I need to write a loop that prints the highest temperature in the `temperatures` array for each day of the week using `find_largest` function.

    The solution to this problem is:

```
1      int *p, *val;
2
3      for (p = &temperatures[i]; p < &temperatures[i] + 24; p++) {
4          val = find_largest(p, 24);
5          printf("%d\n", *val);
6      }
```

17. I need to rewrite the following function to use pointer arithmetic and single loop, instead of array subscripting.

```
1      int sum_two_dimensional_array(const int a[][LEN], int n) {
2          int i, j, sum = 0;
3
4          for (i = 0; i < n; i++) {
5              for (j = 0; j < LEN; j++) {
6                  sum += a[i][j];
7              }
8          }
9
10         return sum;
11     }
```

The solution to this problem is:

```
1        int sum_two_dimensional_array(const int a[][LEN], int n) {
2            int *p, sum = 0;
3
4            for (p = &a[0][0]; p < &a[n-1][LEN-1]; p++) {
5                sum += *p;
6            }
7
8            return sum;
9        }
10
```

18. I need to rewrite the `evaluation_position` function described in exercise 13 of chapter 9 using pointer arithmatic and single loop.

    The solution to this problem is:

```
1      int evaluate_position(char board[8][8]) {
2
3          int white = 0, black = 0;
4          char *p;
5
6          for (p = &board[0][0]; p < &board[7][7]; p++) {
7              switch(*p) {
8                  case 'Q':
9                      white += 9;
10                     break;
11                 case 'q':
12                     black += 9;
13                     break;
14                 case 'R':
15                     white += 5;
16                     break;
17                 case 'r':
18                     black += 5;
19                     break;
20                 case 'B':
21                     white += 3;
22                     break;
23                 case 'b':
24                     black += 3;
25                     break;
26                 case 'N':
27                     white += 3;
28                     break;
29                 case 'n':
30                     black += 3;
31                     break;
32                 case 'P':
33                     white++;
34                     break;
35                 case 'p':
36                     black++;
37                     break;
```

```
38                default:
39                    break;
40            }
41        }
42
43        return white - black;
44    }
```

References

1) Githib (William Gherman), c-solutions (Chapter 9 Exercise 13), link

2   Programming Projects

1. a) I need to write a program that reads a message, then prints the reversal of the message.

   The additional requirement for this program are:

   - Read the message one character at a time using getchar
   - Read characters should be stored in an array
   - Read should be stopped when the array is full or the character read is '\n'

   The solution to this problem is:

```
1     #include <stdio.h>
2
3     #define SIZE 10
4
5     int main(void) {
6
7         char array[SIZE], c, temp;
8         int i = 0, n;
9
10        printf("Enter a message: ");
11
12        // fill array
13        do {
14            c = getchar();
15
16            if (c == '\n') {
17                break;
18            }
19
20            array[i] = c;
21            i++;
22        } while (i < SIZE);
23
24        n = i;
25
```

```
26          // reverse characters in array
27
28          for (int j = 0; j < (n - j); j++) {
29              temp = array[j];
30              array[j] = array[n - (1 + j)];
31              array[n - (1 + j)] = temp;
32          }
33
34          printf("Reversal is: ");
35
36          for (int j = 0; j < n; j++) {
37              printf("%c", array[j]);
38          }
39
40          return 0;
41      }
```

b)
```
1       #include <stdio.h>
2
3       #define SIZE 10
4
5       int main(void) {
6
7           char array[SIZE], c, temp, *p;
8           int i = 0, n;
9
10          p = &array[0];
11
12          printf("Enter a message: ");
13
14          // fill array
15          do {
16              c = getchar();
17
18              if (c == '\n') {
19                  break;
20              }
21
22              *p++ = c;
23              i++;
24          } while (p < array + SIZE);
25
26          n = i;
27
28          // reverse characters in array
29          char *q = &array[0], *r = &array[n-1];
30
31          while (q < r) {
32              temp = *q;
33              *q++ = *r;
34              *r-- = temp;
35          }
36
37          printf("Reversal is: ");
```

```
38
39          for (p = &array[0]; p < array + n; p++) {
40              printf("%c", *p);
41          }
42
43          return 0;
44      }
```

2. a) I need write a program that reads a message and checks if it's a palindrome.

The solution to this problem is:

```
1       #include <stdio.h>
2       #include <stdbool.h>
3       #include <ctype.h>
4
5       #define SIZE 100
6
7       bool is_palindrome(char array[], int n);
8
9       int main(void) {
10          char array[SIZE], c;
11          int n = 0;
12
13          // Read characters
14          // Put characters into array
15          printf("Enter a message: ");
16
17          do {
18              c = getchar();
19
20              if (c == '\n') {
21                  break;
22              }
23
24              if (!isalpha(c)) {
25                  continue;
26              }
27
28              array[n] = tolower(c);
29              n++;
30          } while (n < SIZE);
31
32          // Check if characters in array is palindrome
33          if (is_palindrome(array, n)) {
34              printf("Palindrome");
35          } else {
36              printf("Not a Palindrome");
37          }
38
39          return 0;
40      }
41
```

```
42      bool is_palindrome(char array[], int n) {
43          char temp;
44
45          for (int i = 0; i < (n - i); i++) {
46              if (array[i] != array[n - (1 + i)]) {
47                  return false;
48              }
49          }
50
51          return true;
52      }
```

b) I need to rewrite above program to use pointers instead of integers to keep track of position in the array.

The solution to this problem is:

```
1       #include <stdio.h>
2       #include <stdbool.h>
3       #include <ctype.h>
4
5       #define SIZE 100
6
7       bool is_palindrome(char array[], int n);
8
9       int main(void) {
10          char c, array[SIZE], *p = &array[0];
11          int n = 0;
12
13          // Read characters
14          // Put characters into array
15          printf("Enter a message: ");
16
17          do {
18              c = getchar();
19
20              if (c == '\n') {
21                  break;
22              }
23
24              if (!isalpha(c)) {
25                  continue;
26              }
27
28              *p++ = tolower(c);
29              n++;
30          } while (n < SIZE);
31
32          // Check if characters in array is palindrome
33          if (is_palindrome(array, n)) {
34              printf("Palindrome");
35          } else {
36              printf("Not a Palindrome");
37          }
```

```
38
39              return 0;
40          }
41
42      bool is_palindrome(char array[], int n) {
43              char *p = &array[0],
44                    *q = &array[n-1];
45
46              while (p < q) {
47                      if (*p++ != *q--) {
48                              return false;
49                      }
50              }
51
52              return true;
53          }
```

3. I need to simplify programming project 1(b) by using the fact that array name can be used as a pointer.

   The solution to this problem is:

```
1       #include <stdio.h>
2
3       #define SIZE 100
4
5       int main(void) {
6
7           char array[SIZE], c, *p = &array[0];
8
9           printf("Enter a message: ");
10
11          // fill array
12          do {
13              c = getchar();
14
15              if (c == '\n') {
16                  break;
17              }
18
19              *p++ = c;
20          } while (p < array + SIZE);
21
22          p--;
23          printf("Reversal is: ");
24
25          while (p >= array) {
26              printf("%c", *p);
27              p--;
28          }
29
30          return 0;
31      }
```

4. I need to simplify programming project 2(b) by taking advantage of the fact that array name can be used as a pointer.

   The solution to this problem is:

```c
#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>

#define SIZE 100

bool is_palindrome(char *p, char *q);

int main(void) {
    char c, array[SIZE],
            *p = array,
            *q = array;

    // Read characters
    // Put characters into array
    printf("Enter a message: ");

    do {
        c = getchar();

        if (c == '\n') {
            break;
        }

        if (!isalpha(c)) {
            continue;
        }

        *q++ = tolower(c);
    } while (q < array + SIZE);
    // Set pointer to last element in the array
    q--;
    // Check if characters in array is palindrome
    if (is_palindrome(p,q)) {
        printf("Palindrome");
    } else {
        printf("Not a Palindrome");
    }

    return 0;
}

bool is_palindrome(char *p, char *q) {
    while (p < q) {
        if (*p++ != *q--) {
            return false;
        }
    }
```

```
49
50            return true;
51        }
```

5. I need to modify programming project 14 from chapter 8 so that it uses a pointer instead of an integer.

   The solution to this problem is:

```
1        #include <stdio.h>
2        #include <string.h>
3
4        #define MAX_VALUE 100
5
6        int main(void) {
7
8            char c, terminating_char, words[MAX_VALUE],
9                 *p = &words[0], *q, *r;
10
11           printf("Enter a sentence: ");
12           do {
13               c = getchar();
14
15               if (c == '\n') {
16                   break;
17               }
18
19               if (c == '.' || c == '!' || c == '?') {
20                   terminating_char = c;
21                   break;
22               }
23
24               *p++ = c;
25           } while (p < words + MAX_VALUE);
26
27           p--;
28           printf("Reversal of sentence: ");
29           while (p > words) {
30               if (*p == ' ') {
31                   p--;
32                   continue;
33               }
34
35               q = p;
36               // find the start of a word
37               while(q >= words) {
38                   if (*q == ' ') {
39                       break;
40                   }
41                   q--;
42               }
43
```

```
44              // print word
45              r = q+1;
46
47              while (r <= p) {
48                  printf("%c", *r);
49                  r++;
50              }
51
52              // add space
53              if (q > words) {
54                  printf(" ");
55              }
56
57              p = q;
58          }
59
60          printf("%c\n", terminating_char);
61
62          return 0;
63      }
```

References

1) Githib (William Gherman), c-solutions (Chapter 8 Exercise 14), link

6. Here, I need to modify `qsort.c` program of section 9.6 so `low`, `high` and `middle` are pointers to array elements than integers.

The solution to this problem is:

```
1    /* qsort.c (Chapter 9, page 207) */
2    /* Sorts an array of integers using Quicksort algorithm */
3
4    #include <stdio.h>
5
6    #define N 10
7
8    void quicksort(int *low, int *high);
9    int *split(int *low, int *high);
10
11   int main(void)
12   {
13     int a[N], i,*low = &a[0], *high = &a[N-1];
14
15     printf("Enter %d numbers to be sorted: ", N);
16     for (i = 0; i < N; i++)
17       scanf("%d", &a[i]);
18
19     quicksort(low, high);
20
21     printf("In sorted order: ");
```

```
22        for (i = 0; i < N; i++)
23          printf("%d ", a[i]);
24        printf("\n");
25
26        return 0;
27      }
28
29      void quicksort(int *low, int *high)
30      {
31        int *middle;
32
33        if (low >= high) return;
34        middle = split(low, high);
35        quicksort(low, middle - 1);
36        quicksort(middle + 1, high);
37      }
38
39      int *split(int *low, int *high)
40      {
41        int part_element = *low;
42
43        for (;;) {
44          while (low < high && part_element <= *high)
45            high--;
46          if (low >= high) break;
47          *low++ = *high;
48
49          while (low < high && *low <= part_element)
50            low++;
51          if (low >= high) break;
52          *high-- = *low;
53        }
54
55        *high = part_element;
56        return high;
57      }
```

7. I need to modify `maxmin.c` program of section 11.4, so that the `max_min` function uses a pointer instead of an integer to keept track of the current postion in the array.

   The solution to this problem is:

```
1      /* maxmin.c (Chapter 11, page 250) */
2      /* Finds the largest and smallest elements in an array */
3
4      #include <stdio.h>
5
6      #define N 10
7
8      void max_min(int a[], int n, int *max, int *min);
9
10     int main(void)
```

```
11      {
12      int b[N], i, big, small;
13
14      printf("Enter %d numbers: ", N);
15      for (i = 0; i < N; i++)
16          scanf("%d", &b[i]);
17
18      max_min(b, N, &big, &small);
19
20      printf("Largest: %d\n", big);
21      printf("Smallest: %d\n", small);
22
23      return 0;
24      }
25
26      void max_min(int a[], int n, int *max, int *min)
27      {
28      int i;
29
30      *max = *min = a[0];
31      for (i = 1; i < n; i++) {
32          if (a[i] > *max)
33          *max = a[i];
34          else if (a[i] < *min)
35          *min = a[i];
36      }
37      }
```