Week 3 solutions

1. Exercises 5-1, 5-2, and 5-4 in "Beginning C":

Exercise 5-1

```c
#include <stdio.h>

int main(void) {

  double arr[5], repr[5], rsum = 0.0;

  // Prompt user to enter numbers
  printf("Enter five nonzero numbers:\n");
  for (int i = 0; i < 5; i++) {
    printf("%2d> ", i + 1);
    scanf("%lf", arr + i);
  }

  // Compute and sum reciprocals
  for (int i = 0; i < 5; i++) {
    repr[i] = 1 / arr[i];
    rsum += repr[i];
  }
  printf("Sum of reciprocals: %.4e\n", rsum);

  return 0;
}
```

Exercise 5-2

```c
#include <stdio.h>

int main(void) {

  double data[100], val = 0.0, k = 1.0;
  int j;

  // Compute array with terms
  for (int i = 0; i < 100; i++) {
    j = 2 + 2 * i; // 2,4,6,...,200
    data[i] = 1.0 / (j * (j + 1) * (j + 2));
  }

  // Sum terms with alternating sign
  for (int i = 0; i < 100; i++) {
    val += data[i] * k;
    k *= -1;
```

```
  }

  // Multiply by 4, add 3, and print
  val = 4 * val + 3;
  printf("Value: %.16lf\n", val);

  return 0;
}
```

Exercise 5-4

```
#include <stdio.h>

int main(void) {

  // Allocate and fill array
  double data[11][5];
  for (int i = 0; i < 11; i++) {
    data[i][0] = 2.0 + 0.1 * i;           // x
    data[i][1] = 1 / data[i][0];          // 1/x
    data[i][2] = data[i][0] * data[i][0]; // x^2
    data[i][3] = data[i][0] * data[i][2]; // x^3
    data[i][4] = data[i][0] * data[i][3]; // x^4
  }

  // Print table head
  printf("%8s %8s %8s %8s %8s\n", "x", "1/x", "x^2", "x^3", "x^4");
  for (int i = 0; i < 11; i++) {
    // Print table row
    for (int j = 0; j < 5; j++)
      printf("%8.2f ", data[i][j]);
    printf("\n");
  }
  return 0;
}
```

2. See quiz answers here.

3. See quiz answers here.

4. The program computes the average of the 10 numbers. The while loop is equivalent to a for loop:

```
#include <stdio.h>

int main(void) {

  int arr[10] = {19, 74, 13, 67, 44, 80, 7, 36, 9, 77};
  double val = 0.0;
```

```c
  // Compute average
  for (int i = 0; i < 10; i++)
    val += arr[i];
  val /= 10;

  printf("Value: %.2f\n", val);

  return 0;
}
```

5. Copying arrays with `memcpy()`:

```c
#include <stdio.h>
#include <string.h>

#define N 10

int main(void) {

  double arr1[N], arr2[N];

  // Prompt user to enter N numbers
  printf("Input %d numbers:\n", N);
  for (int i = 0; i < N; i++) {
    printf("%2d>", i + 1);
    scanf("%lf", arr1 + i);
  }

  // Copy data from arr1 to arr2
  memcpy(arr2, arr1, N * sizeof(double));

  // Print values stored in arr2
  for (int i = 0; i < N; i++)
    printf("%2d: %.4f\n", i + 1, arr2[i]);

  return 0;
}
```

6. One-pass and two-pass variance (sequential summation).

```c
#include <stdio.h>
#define N 6
#ifndef fp_t
#define fp_t double
#endif

int main(void) {
```

```
  fp_t arr[N] = {34124.75, 34124.48, 34124.90,
                 34125.31, 34125.05, 34124.98};
  fp_t sum, sum_sq, mean, var_1pass, var_2pass;

  /* Sequential summation */

  /* Compute sum and sum of squares */
  sum = 0.0;
  sum_sq = 0.0;
  for (size_t i = 0; i < N; i++) {
    sum += arr[i];
    sum_sq += arr[i] * arr[i];
  }
  mean = sum / N;
  var_1pass = (sum_sq - N * mean * mean) / (N - 1);

  /* Compute two-pass variance */
  var_2pass = 0.0;
  for (size_t i = 0; i < N; i++)
    var_2pass += (arr[i] - mean) * (arr[i] - mean);
  var_2pass /= N - 1;

  printf("%20.16g %20.16g\n", var_1pass, var_2pass);

  return 0;
}
```

7. One-pass and two-pass variance (Kahan summation).

```
#include <stdio.h>
#define N 6
#ifndef fp_t
#define fp_t double
#endif

int main(void) {

  fp_t arr[N] = {34124.75, 34124.48, 34124.90,
                 34125.31, 34125.05, 34124.98};
  fp_t sum, sum_sq, mean, var_1pass, var_2pass;
  fp_t c1, c2, t, y;

  /* Compute sum and sum of squares */
  c1 = 0.0;
  c2 = 0.0;
  sum = 0.0;
```

```
  sum_sq = 0.0;
  for (size_t i = 0; i < N; i++) {
    y = arr[i] - c1;
    t = sum + y;
    c1 = (t - sum) - y;
    sum = t;

    y = arr[i] * arr[i] - c2;
    t = sum_sq + y;
    c2 = (t - sum_sq) - y;
    sum_sq = t;
  }
  mean = sum / N;
  var_1pass = (sum_sq - N * mean * mean) / (N - 1);

  /* Compute two-pass variance */
  c1 = 0.0;
  var_2pass = 0.0;
  for (size_t i = 0; i < N; i++) {
    t = arr[i] - mean;
    y = t * t - c1;
    t = var_2pass + y;
    c1 = (t - var_2pass) - y;
    var_2pass = t;
  }
  var_2pass /= N - 1;

  printf("%20.16g %20.16g\n", var_1pass, var_2pass);

  return 0;
}
```

The results are summarized in the table below:

Method	`float`	`double`
One-pass, seq.	0.0	0.07901687622070312
Two-pass, seq.	0.07877807319164276	0.07901666666598467
One-pass, Kahan	0.0	0.07901668548583984
Two-pass, Kahan	0.07877807319164276	0.07901666666598467

It is clear from the table that the two-pass method is much more reliable than the one-pass method. Notice that the Kahan summation cannot salvage the one-pass method in single precision since the issue is catastrophic cancellation in the computation after the loop. In double precision, the Kahan sum appears to have some effect on the one-pass method, but it does not address the real issue, namely catastrophic cancellation after the loop.

8. For the array provided in the textbook, estimate the (relative) condition number associated with each of the following sums:

- $S_1 = \sum_{i=1}^{n} a_i$
- $S_2 = \sum_{i=1}^{n} a_i^2$
- $S_2 - \frac{1}{n} S_1^2$

Since the array a is element-wise positive, the relative condition number associated with S_1 and S_2 are both equal to 1. The relative condition number associated with $S_2 - \frac{1}{n} S_1^2$ is approximately

$$\frac{|S_2| + \frac{1}{n}|S_1^2|}{|S_2 - \frac{1}{n} S_1^2|} \approx 3.5 \cdot 10^{10}.$$

The large condition number indicates that the relative error associated with the last summation could be very large.

9. The worst-case relative error is

$$\frac{\sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|} \cdot 1.06nu \approx 53{,}000nu.$$

10. The condition number of $f(x) = \cos(x)$ is

$$c(x) = \left| \frac{x f'(x)}{f(x)} \right| = \left| \frac{x \sin(x)}{\cos(x)} \right| = |x \tan(x)|.$$