Week 4 solutions

2. See solutions on CodeJudge.
3. Do exercises 7-1 and 7-4 in "Beginning C"

   Exercise 7-1

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

int main(void) {

  int n = 0, N = 4;
  double *data = NULL, *tmp = NULL, val = 0.0, avg = 0.0;

  // Allocate storage for N doubles
  data = (double *)malloc(N*sizeof(*data));
  if (!data) {
    fprintf(stderr,"Memory allocation failed\n");
    return EXIT_FAILURE;
  }

  // Prompt user to enter data
  printf("Please enter your data:\n>");
  while (scanf("%lf", &val) == 1) {
    if (n == N) {
      // Request space for 4 additional doubles
      N = N + 4;
      tmp = (double *)realloc(data, N*sizeof(*data));
      if (!tmp) {
        // Reallocation failed
        fprintf(stderr,"Memory reallocation failed\n");
        free(data);
        return EXIT_FAILURE;
      }
      data = tmp;
    }
    data[n++] = val;
    printf(">");
  }

  // Compute and print average
  printf("You entered %d values.\n", n);
  for (size_t i = 0; i < n; i++) {
    avg += data[i];
  }
  avg /= n;
```

```
  printf("Average value: %g\n", avg);

  // Free allocated memory and exit
  free(data);
  return EXIT_SUCCESS;
}
```

Exercise 7-4

```
#include <stdio.h>
#include <stdlib.h>

#define REC_PER_DAY 6

int main(void) {

  int ndays = 0, N = 4;
  double *data = NULL, *tmp = NULL, *avg = NULL;
  char choice = 'n';

  // Allocate memory for N days
  data = (double *)malloc(N * REC_PER_DAY * sizeof(double));
  if (!data) {
    printf("Memory allocation failed.\n");
    exit(-1);
  }

  // Prompt user to input data
  do {
    if (ndays == N) {
      // Ask for more memory: N+4 days
      N += 4;
      tmp = realloc(data, N * REC_PER_DAY * sizeof(double));
      if (!tmp) {
        printf("Reallocation failed.\n");
        free(data);
        return EXIT_FAILURE;
      }
      data = tmp;
    }
    printf("Input data for day %d:\n", ndays + 1);
    for (size_t i = 0; i < REC_PER_DAY; i++) {
      printf("  Measurement #%zu: ", i + 1);
      scanf("%lf", data + ndays * REC_PER_DAY + i);
    }
    ndays++;
    printf("Continue? [y/n] ");
    scanf(" %c", &choice);
```

```
  } while (choice == 'y');

  // Allocate memory for average temperature
  avg = (double *)malloc(ndays * sizeof(double));
  if (!avg) {
    printf("Allocating memory for avg. temperature failed.\n");
    free(data);
    return EXIT_FAILURE;
  }

  // Compute and print averages
  for (int i = 0; i < ndays; i++) {
    avg[i] = 0.0;
    for (int j = 0; j < REC_PER_DAY; j++)
      avg[i] += data[i * REC_PER_DAY + j];
    avg[i] /= REC_PER_DAY;
    printf("Day %d average: %g\n", i + 1, avg[i]);
  }

  // Free memory and exit
  free(avg);
  free(data);
  return EXIT_SUCCESS;
}
```

4. Multi-file projects and dynamic allocation of two-dimensional arrays

```
/* array.h */
#ifndef ARRAY_H
#define ARRAY_H

#include <stdlib.h>
#include <stdint.h>
double ** malloc_array2d(size_t m, size_t n);
void free_array2d(double **ptr);

#endif
```

```
/* array.c */
#include "array.h"

/* Allocate two-dimensional array of doubles */
double **malloc_array2d(size_t m, size_t n) {
  // Check dimensions and check m*n for overflow
  if (m == 0 || n == 0 || m > SIZE_MAX / n)
    return NULL;
  // Allocate two-dimensional array
```

```
  double **B = malloc(m * sizeof(*B));
  if (B == NULL)
    return NULL;
  B[0] = malloc(m * n * sizeof(double));
  if (B[0] == NULL) {
    free(B);
    return NULL;
  }
  for (size_t i = 1; i < m; i++)
    B[i] = B[0] + i * n;
  return B;
}

/* Free two-dimensional array of doubles */
void free_array2d(double **ptr) {
  free(ptr[0]);
  free(ptr);
  return;
}
```

```
/* test.c */
#include "array.h"
#include <math.h>
#include <stdio.h>

int main(int argc, char const *argv[]) {
  size_t m = 10, n = 5;
  double **E = malloc_array2d(m, n);

  for (size_t i = 0; i < 10; i++) {
    for (size_t j = 0; j < 5; j++) {
      E[i][j] = exp(-i * n - j);
    }
  }

  free_array2d(E);

  if (malloc_array2d(m, 0) != NULL)
    return -1;
  if (malloc_array2d(0, n) != NULL)
    return -1;
  if (malloc_array2d(0, 0) != NULL)
    return -1;

  return 0;
}
```

Makefile

```
CFLAGS=-Wall -std=c99
LDLIBS=-lm

test: test.o array.o
test.o: array.h
array.o: array.h

.PHONY: clean run
clean:
	-$(RM) test test.o array.o

run: test
	./test $(ARGS)
```

5. Do exercise 8-1 in "Beginning C"

Makefile

```
CFLAGS=-Wall -std=c99
LDLIBS=-lm

bc_81: avg.o read_input.o
avg.o: avg.h
read_input.o: read_input.h

.PHONY: clean run
clean:
	-$(RM) *.o bc_81

run: bc_81
	./bc_81
```

Source files

```
/* bc_81.c */
#include <stdlib.h>
#include <stdio.h>
#include "read_input.h"
#include "avg.h"

int main(int argc, char const *argv[]) {
  double * x;
  size_t n;
  read_input(&x,&n);  // This function handles memory allocation!
  if ((x==NULL)||(n==0)) return EXIT_FAILURE;
```

```
  printf("The average of the %lu numbers is: %.3f\n",n,avg(x,n));
  free(x);
  return 0;
}
```

```
/* avg.h */
#ifndef AVG_H
#define AVG_H

#include <math.h>
#include <stdlib.h>

double avg(double * x, size_t n);

#endif
```

```
/* avg.c */
#include "avg.h"

double avg(double * x, size_t n) {
    double sum = 0.0;
    if(x==NULL) return NAN;
    for (size_t i = 0; i < n; i++) {
        sum += x[i];
    }
    return sum/n;
}
```

```
/* read_input.h */
#ifndef READ_INPUT_H
#define READ_INPUT_H

#include <stdlib.h>
#include <stdio.h>

void read_input(double ** x, size_t * length);

#endif
```

```
/* read_input.c */
#include "read_input.h"

void read_input(double ** x, size_t * length) {

  size_t n=0, N=4;
  double val, *data=NULL, *tmp=NULL;
```

```
  *x = NULL;
  *length = 0;

  // Allocate storage for N doubles
  data = (double *)malloc(N*sizeof(*data));
  if (!data) {
    fprintf(stderr,"Memory allocation failed\n");
    return;
  }

  // Prompt user to enter data
  printf("Please enter your data "
         "(terminate with any non-numeric input):\n>");
  while (scanf("%lf", &val) == 1) {
    if (n == N) {
      // Request space for 4 additional doubles
      N = N + 4;
      tmp = (double *)realloc(data, N*sizeof(*data));
      if (!tmp) {
        // Reallocation failed
        fprintf(stderr,"Memory reallocation failed\n");
        free(data);
        return;
      }
      data = tmp;
    }
    data[n++] = val;
    printf(">");
  }

  *length = n;
  *x = data;
  return;
}
```