Week 5 solutions

1. Do exercise 11-1 in “Beginning C”

```c
/* length.h */
#ifndef LENGTH_H
#define LENGTH_H

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

// Structure representing length in yards, feet, and inches
struct Length {
  double yards;
  double feet;
  double inches;
};

// Function prototypes
struct Length * add(struct Length * l1, struct Length * l2);
void show(struct Length * l);

#endif
```

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

// Function that adds to Length structures and
// returns a pointer to a new Length structure (the sum)
struct Length * add(struct Length * l1, struct Length * l2) {

  // Check input and allocate new "Length" structure
  struct Length * psum;
  if ( l1 == NULL || l2 == NULL ) return NULL;
  psum = malloc(sizeof(struct Length));
  if ( psum == NULL ) return NULL;

  // Add length l1 and l2, and return pointer to sum
  psum->yards = l1->yards + l2->yards;
  psum->feet = l1->feet + l2->feet;
  psum->inches = l1->inches + l2->inches;
  // Convert multiples of 12 inches to feet
  if (psum->inches >= 12) {
    psum->feet += floor(psum->inches/12);
    psum->inches -= 12*floor(psum->inches/12);
  }
  // Convert multiples of 3 feet to yards
  if (psum->feet >= 3) {
    psum->yards += floor(psum->feet/3);
    psum->feet -= 3*floor(psum->feet/3);
  }
  return psum;
}

// Function that prints length
void show(struct Length * L) {
  if ( L != NULL )
    printf("The length is %g yd %g' %g\"\n",
              L->yards,L->feet,L->inches);
  return;
}
```

```c
/* myprog.c */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "length.h"

// Our program starts here
int main(void) {

  struct Length *plen, *ptmp, *ptotal;

  // Allocate memory for two "Length" structures
  plen = calloc(1,sizeof(struct Length));
  ptotal = calloc(1,sizeof(struct Length));
  if ( ptotal == NULL || plen == NULL ) return EXIT_FAILURE;

  // Prompt user to enter length in yards, feet, and inches
  printf("Enter length in yards, feet, and inches [X yd Y' Z\"]: ");
  while (scanf("%lf yd %lf' %lf\"",
          &(plen->yards),
          &(plen->feet),
          &(plen->inches)) == 3) {
    // Add Length structs pointed to by ptotal and plen
    ptmp = add(ptotal, plen);

    // Check for errors
    if ( ptmp == NULL ) {
      free(ptotal);
      free(plen);
      return EXIT_FAILURE;
    }
    // Make ptotal point to new total
    free(ptotal);
    ptotal = ptmp;  // ptmp is new total

    printf("Enter another length [X yd Y' Z\"]: ");
  }

  // Display total length
  show(ptotal);

  // Free memory and return
  free(ptotal); free(plen);
  return 0;
}
```

2. Write a short program that (i) prompts the user to enter three points in $\mathbb{R}^2$ that define a triangle, and (ii) computes and prints the area of the triangle. Your program should use structures and functions.

```
/* triangle.h */
#ifndef TRIANGLE_H
#define TRIANGLE_H

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

// Structure representing a point (x,y)
struct point {
  double x;
  double y;
};

// Structure representing a triangle (three points)
struct triangle {
  struct point vertices[3];
};

double area(struct triangle * T);
int point_in_triangle(struct point *P, struct triangle *T);
int congruent(struct triangle *T1, struct triangle *T2);

#endif
```

```
/* triangle.c */
```

```c
#include "triangle.h"

// Computes the area of a triangle
double area(struct triangle * T) {

  double a,b,c,s;
  struct point *v1, *v2, *v3;

  v1 = &T->vertices[0];  // Pointer to first vertex
  v2 = &T->vertices[1];  // Pointer to second vertex
  v3 = &T->vertices[2];  // Pointer to third vertex

  // Calculate area of triangle using Heron's formula
  a = hypot(v1->x - v2->x, v1->y - v2->y);
  b = hypot(v2->x - v3->x, v2->y - v3->y);
  c = hypot(v3->x - v1->x, v3->y - v1->y);
  s = (a+b+c)/2.0;
  return sqrt(s*(s-a)*(s-b)*(s-c));
}


// Checks if a point is inside a triangle
#define x(i) T->vertices[i-1].x
#define y(i) T->vertices[i-1].y
int point_in_triangle(struct point *P, struct triangle *T) {

  double a1,a2,a3;
  double x = P->x;
  double y = P->y;

  // Barycentric coordinates for x,y
  a1 = ((y(2)-y(3))*(x-x(3)) + (x(3)-x(2))*(y-y(3))) /
         ((y(2)-y(3))*(x(1)-x(3)) + (x(3)-x(2))*(y(1)-y(3)));
  a2 = ((y(2)-y(1))*(x-x(3)) + (x(1)-x(3))*(y-y(3))) /
         ((y(2)-y(3))*(x(1)-x(3)) + (x(3)-x(2))*(y(1)-y(3)));
  a3 = 1.0 - a1 - a2;

  if (a1 >= 0 && a2 >= 0 && a3 >= 0)
    return 1;  // (x,y) is inside triangle
  else
    return 0;  // (x,y) is outside triangle
}
#undef x
#undef y
```

```c
#define ALMOST_EQUAL(x,y) (fabs(x-y)<=(fabs(x)+fabs(y))*DBL_EPSILON)
#define sidelen(T,i) hypot(T->vertices[i%3].x-T->vertices[(i+1)%3].x, \
    T->vertices[i%3].y-T->vertices[(i+1)%3].y)

int congruent(struct triangle *T1, struct triangle *T2) {

  double a1,b1,c1,a2,b2,c2,s;

  // Compute side lengths in triangle 1
  a1 = sidelen(T1,0);
  b1 = sidelen(T1,1);
  c1 = sidelen(T1,2);

  // Sort a1,b1,c1
  if (a1 < b1) { s = a1; a1 = b1; b1 = s; }
  if (a1 < c1) { s = a1; a1 = c1; c1 = s; }
  if (b1 < c1) { s = b1; b1 = c1; c1 = s; }

  // Compute side lengths in triangle 2
  a2 = sidelen(T2,0);
  b2 = sidelen(T2,1);
  c2 = sidelen(T2,2);

  // Sort a2,b2,c2
  if (a2 < b2) { s = a2; a2 = b2; b2 = s; }
  if (a2 < c2) { s = a2; a2 = c2; c2 = s; }
  if (b2 < c2) { s = b2; b2 = c2; c2 = s; }

  // Compare side lengths
  if (ALMOST_EQUAL(a1,a2) &&
      ALMOST_EQUAL(b1,b2) &&
      ALMOST_EQUAL(c1,c2)) {
    return 1; // Triangles are congruent (within numerical precision)
  }
  else {
    return 0; // Triangles are not congruent
  }
};
```

```c
/* testprog.c */
#include <stdio.h>
#include "triangle.h"

// Our program starts here
int main(void) {

  struct point P = {.x=1.0,.y=1.0};
  struct triangle T[2];

  // Prompt user to enter two triangles
  for (int k=0;k<2;k++) {

    printf("Triangle T%d:\n",k+1);
    for (int i=0;i<3;i++) {
      printf("Enter triangle vertex %d [x,y]: ", i+1);
      scanf("%lf,%lf", &(T[k].vertices[i].x),&(T[k].vertices[i].y));
    }

    // Compute and print area of triangle
    double a = area(&T[k]);
    printf("Area of triangle T%d: %g\n",k+1,a);

    // Check if (1,1) is contained in triangle
    if (point_in_triangle(&P, &T[k]))
      printf("(1,1) is inside the triangle T%d\n",k+1);
    else
      printf("(1,1) is not inside the triangle T%d\n",k+1);
  }
  if (congruent(&T[0],&T[1]))
    printf("The triangles T1 and T2 are congruent.\n");
  else
    printf("The triangles T1 and T2 are not congruent.\n");

  return 0;
}
```

3. See solution example for exercise 3.

4. See solution example for exercise 3.

Optional exercises

1. Suppose that an `int` takes up 4 bytes, a `short` takes up 2 bytes, and a `char` takes up 1 byte. Then the structure requires at least 7 bytes, but `sizeof(my_struct)` is 8. Why the discrepancy? The answer can be found in “Beginning C” on page 441:

   It's very important to use `sizeof` *when you need the number of bytes occupied by a structure. It does not necessarily correspond to the sum of the bytes occupied by each of its individual members, so you may get it wrong if you try to work it out yourself. Variables other than type char are often stored beginning at an address that's a multiple of two for 2-byte variables, a multiple of four for 4-byte variables, and so on. This is called boundary alignment and it has nothing to do with C in particular, but it can be a hardware requirement.*