

## Module 5 solutions

1. See quiz answers [here](#).
2. Do exercises 2, 3, and 10 (Ch. 16, p. 408) in the textbook.

```
/* King, Exercise 2, Chapter 16 */
#include <stdlib.h>
#include <stdio.h>

int main(void) {

    // part (a) + (b)
    struct {
        double real, imaginary;
    } c1 = {0.0, 1.0}, c2 = {1.0, 0.0}, c3;

    // part (c)
    c1 = c2;
    printf("c1.real=%f c1.imaginary=%f\n", c1.real, c1.imaginary);

    // part (d)
    c3.real = c1.real + c2.real;
    c3.imaginary = c1.imaginary + c2.imaginary;
    printf("c3.real=%f c3.imaginary=%f\n", c3.real, c3.imaginary);

    return EXIT_SUCCESS;
}
```

```
/* King, Exercise 3, Chapter 16 */
#include <stdlib.h>
#include <stdio.h>

// part (a)
struct complex {
    double real, imaginary;
};

// part (c) + (d)
struct complex make_complex(double real, double imaginary);
struct complex add_complex(struct complex a, struct complex b);

int main(void) {

    // part (b)
    struct complex c1, c2, c3;
```

```

    // part (c)
    c1 = make_complex(1.0,-1.0);
    c2 = make_complex(4.0,2.0);
    printf("c1.real=%f c1.imaginary=%f\n",c1.real,c1.imaginary);
    printf("c2.real=%f c2.imaginary=%f\n",c2.real,c2.imaginary);

    // part (d)
    c3 = add_complex(c1,c2);
    printf("c3.real=%f c3.imaginary=%f\n",c3.real,c3.imaginary);

    return EXIT_SUCCESS;
}

// part (c)
struct complex make_complex(double real, double imaginary) {
    struct complex c = {.real = real, .imaginary = imaginary};
    return c;
}

//part (d)
struct complex add_complex(struct complex a, struct complex b) {
    struct complex c;
    c.real = a.real + b.real;
    c.imaginary = a.imaginary + b.imaginary;
    return c;
}

```

```

/* King, Exercise 10, Chapter 16 */
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

struct point { int x, y; };
struct rectangle { struct point upper_left, lower_right; };

double area(struct rectangle r);
struct point center(struct rectangle r);
struct rectangle move(struct rectangle r, int x, int y);
bool inside(struct rectangle r, struct point p);

int main(void)
{
    struct point lr = {4, 5}, ul = {8, 1};
    struct point po = {12, 3}, pi = {5, 3};
    struct rectangle r = {lr, ul};
    printf("Rectangle r: \n"
           "  upper_left:  x=%d,y=%d\n"

```

```

        " lower_right: x=%d,y=%d\n",
        r.upper_left.x, r.upper_left.y,
        r.lower_right.x, r.lower_right.y);
printf(" area: %g\n", area(r));
struct point cnt = center(r);
printf(" center: x=%d,y=%d\n", cnt.x, cnt.y);

struct rectangle rt = move(r, 4, -2);
printf("Translated rectangle: \n"
      " upper_left:  x=%d,y=%d\n"
      " lower_right: x=%d,y=%d\n",
      rt.upper_left.x, rt.upper_left.y,
      rt.lower_right.x, rt.lower_right.y);

if (inside(r, pi))
    printf("point pi is inside r\n");
if (!inside(r, po))
    printf("point po is outside r\n");
return 0;
}

double area(struct rectangle r)
{ // part (a)
    return (r.upper_left.y - r.lower_right.y) *
          (r.lower_right.x - r.upper_left.x);
}

struct point center(struct rectangle r)
{ // part (b)
    struct point p;
    p.x = (r.upper_left.x + r.lower_right.x) / 2; // integer div.
    p.y = (r.upper_left.y + r.lower_right.y) / 2; // integer div.
    return p;
}

struct rectangle move(struct rectangle r, int x, int y)
{ // part (c)
    struct rectangle rt = r;
    rt.upper_left.x += x;
    rt.upper_left.y += y;
    rt.lower_right.x += x;
    rt.lower_right.y += y;
    return rt;
}

bool inside(struct rectangle r, struct point p)
{ // part (d)

```

```
    if (p.x > r.lower_right.x || p.x < r.upper_left.x)
        return false;
    if (p.y < r.lower_right.y || p.y > r.upper_left.y)
        return false;
    return true;
}
```

3. Do exercise 2 (Ch. 17, p. 453) in the textbook.

```
/* King, Exercise 3, Chapter 17 */
#include <stdlib.h>
#include <stdio.h>

int *create_array(int n, int initial_value);

int main(void)
{
    int *a = create_array(-1,0); // n is negative -> NULL
    if (!a) return EXIT_FAILURE;

    int *b = create_array(5,2);
    if (!b) return EXIT_FAILURE;

    for (int k=0; k<5; k++) printf("b[%d] = %d\n", k, b[k]);

    free(b);
    return EXIT_SUCCESS;
}

int *create_array(int n, int initial_value) {
    if (n<=0) return NULL;
    int *a = malloc(n*sizeof(*a));
    if (!a) return NULL;
    for (int k=0; k<n; k++) a[k] = initial_value;
    return a;
}
```

4. Autolab solutions will be available on [DTU Learn](#) after the deadline.