Série 12 niveau 0

Exercice 1 (niveau 0): Définition de structures (en anglais)

In this exercise you will get familiar to the basics of “struct” in C programming language. To do that, you are expected to write a program which checks if two given circles intersect or not.

In 2D, a circle can be represented by its radius and the position of its center. There are different ways of representing a circle in a C program. A simple way is to use basic data types:

```c
float circle_radius;
float circle_center_x;
float circle_center_y;
```

Now, let’s think about a function for the circle intersection test. Its declaration would look like:

```c
int circlesIntersect(float circle1_radius,
                     float circle1_center_x,
                     float circle1_center_y,
                     float circle2_radius,
                     float circle2_center_x,
                     float circle2_center_y);
```

As you see in the declaration, the function takes six arguments for doing an operation on two circles. This makes both writing and understanding the code complicated. Moreover, instead of dealing with simple circles, we have to deal with three different data, and this also makes the design part harder. Therefore, to make the life of programmers easier, the C programming language lets us aggregate related data in labeled structures called “struct”. By using “struct” we can declare a circle as:

```c
struct Circle { float radius; float center_x; float center_y; }
```

And, in this case, we can define a variable, circle, in our program as:

```c
struct Circle circle;
```

Now let’s reconsider the function declaration for the intersection test. New declaration would look like:

```c
int circlesIntersect(struct Circle circle1, struct Circle circle2);
```

Initialization of “struct”s is similar to arrays:

```c
struct Circle circle1 = {3.0, 50.0, -30.0};
```

As you may notice, to create a variable using “struct” we add “struct” keyword in front of the definition. In order to avoid that we can use type definitions:

The first approach combines the struct definition with the type definition like this:
typedef struct
{
    float radius;
    float center_x;
    float center_y;
} CIRCLE;

The second approach decouples the struct definition from the type definition like this:

struct Circle
{
    float radius;
    float center_x;
    float center_y;
};
typedef struct Circle CIRCLE ;

The first approach looks more convenient and should be used whenever the struct model is to be made public over all modules.

However, in some cases, especially if we want to enforce the information hiding Principle then we have to use the second approach because the struct model definition can be hidden in the implementation (.c file) whereas the type definition is made public in the interface (.h file).

Once the type is defined, we can simply create a variable as:

CIRCLE circle1;

It is also possible to have other structures in a struct. Let’s create a “struct” to hold a 2D position:

typedef struct
{
    float x;
    float y;
} POSITION;

or like this:

struct Position
{   
    float x;
    float y;
};
typedef struct Position POSITION ;

Hence, we can declare a circle as:

typedef struct
{   
    float radius;
    POSITION center;
} CIRCLE;
or like this:

```c
struct Circle
{
float radius;
POSITION center;
};

typedef struct Circle CIRCLE;
```

We can access the elements of a “struct” using the dot operator ‘.’:

```c
circle1.radius = 2.5;
```

It is also possible to access the elements of the nested structures:

```c
circle1.center.x = 50.0;
```

Now, let’s complete our intersection function. Intersection of two circles in 2D can be done by comparing the distance between the centers of the circles with the sum of their radius lengths. You can find an example solution below (circle_intersection.c):

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

typedef struct //coordinates of a point in 2D.
{
    float x;
    float y;
}POSITION;

typedef struct // circle in 2D.
{
    float radius;
    POSITION center;
}CIRCLE;

// A function for circle-circle intersection test
int circlesIntersect(CIRCLE circle1, CIRCLE circle2)
{
    float horizontalDistance = circle1.center.x - circle2.center.x;
    float verticalDistance = circle1.center.y - circle2.center.y;
    float euclideanDistance = sqrt(horizontalDistance * horizontalDistance +
        verticalDistance * verticalDistance);

    float sumOfRadiuses = circle1.radius + circle2.radius;
    return euclideanDistance <= sumOfRadiuses;
}

void usageAlert(void)
{
    printf("Please enter 3 floating points:\n");
    printf("radius, center_x, center_y respectively\n");
    printf("for two circles.\n");
}

int main(void)
{
    CIRCLE circle1, circle2;
    printf("Enter radius, center_x and center_y for the first circle:\n");
    if(scanf("%f %f %f", &(circle1.radius), &(circle1.center.x), &(circle1.center.y)) != 3) {
        usageAlert();
        return 0;
    }
```
printf("Enter radius, center_x and center_y for the second circle:\n");

if(scanf("%f %f %f", &(circle2.radius), &(circle2.center.x), &(circle2.center.y)) != 3) {
    usageAlert();
    return 0;
}

if(circlesIntersect(circle1, circle2))
    printf("Circles intersect.\n");
else
    printf("Circles do not intersect.\n");

return 0;