TECHNISCHE UNIVERSITÄT
CHEMNITZ

Professur Betriebssysteme

Bonusaufgabe 6

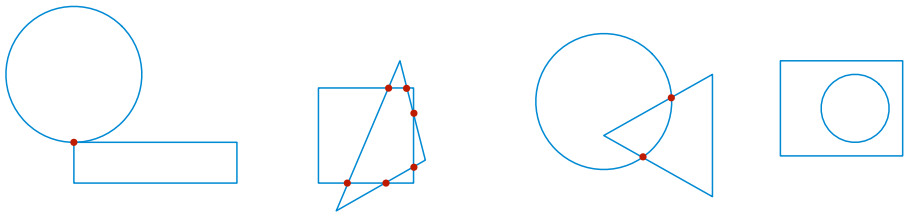
Algorithmen und Programmierung

Matthias Werner

Allgemeine Regeln

- Bonusaufgaben sind fakultativ, außer für Studierende des Bachelors Informatik und Kommunikationswissenschaften (B_CC), hier sind sie Vorleistung.
- Die Lösung soll ausschließlich über OpenSubmit abgegeben werden: `https://osg.informatik.tu-chemnitz.de/submit/`
- Bis zum Abgabeende (Deadline) können beliebig neue Lösungen abgegeben werden, die die jeweils älteren Versionen ersetzen.
- Eine Lösung wird nur bewertet, wenn der Eingangstest von OpenSubmit positiv ausfällt. Das Ergebnis ist im Status Ihrer Abgabe im OpenSubmit dargestellt; Sie werden außerdem bei einem negativen Ergebnis per E-Mail informiert. Bei negativen Eingangstest können Sie (bis zum Abgabeende) jederzeit eine neue Lösung einreichen.
- Ihr Programm muss auf der Testmaschine übersetzbar und lauffähig sein. Die exakten Daten der Testmaschine und des Compilers erfahren Sie über den Link „Test Machines“ im OpenSubmit-Dashboard.
- Ihre Abgabe besteht aus einem ZIP- oder TAR-Archiv, welches ausschließlich ihre Quelldatei enthält. Insbesondere sollte das Archiv kein Unterverzeichnis enthalten.
- Zur Übersetzung werden folgende Optionen genutzt:
  `-std=c99 -Wall -Wextra -Wpedantic -Werror -c`

In einem Computerspiel gibt es sich bewegende geometrische Formen (Software-Sprites), nämlich Kreise, Dreiecke und Rechtecke. Ihre Aufgabe besteht darin eine C-Funktion `cutpoints()` zu schreiben, die alle Schnitt- oder Berührungspunkte von jeweils zwei gegebenen Formen ermittelt.



Alle Formen werden mit einer gemeinsamen Datenstruktur beschrieben, die in der Headerdatei `cp.h` definiert ist.

```c
/* cp.h – Headerdatei */

typedef enum{s_circle, s_rectangle, s_triangle} form_t;

// Kartesische Koordinaten
typedef struct {
  double x;
  double y;
} point_t;


typedef struct shape{
  form_t      type;         // welche Art von geometrischer Figur?
  double      param[6];     // Parameter der Figur
} shape_t;

point_t *cutpoints(shape_t shape_a, shape_t shape_b, int *num);
```

Die Interpretation der Parameter richtet sich nach Art der Form:

- Kreis (`type=s_circle`):
  - `param[0]`: x-Koordinate des Kreismittelpunktes
  - `param[1]`: y-Koordinate des Kreismittelpunktes
  - `param[2]`: Radius des Kreises
- Rechteck (`type=s_rectangle`):
  - `param[0]`: x-Koordinate der linken, oberen Ecke des Rechtecks
  - `param[1]`: y-Koordinate der linken, oberen Ecke des Rechtecks
  - `param[2]`: Breite (x-Richtung) des Rechtecks
  - `param[3]`: Höhe (y-Richtung) des Rechtecks
- Dreieck (`type=s_triangle`):
  - `param[0]`: x-Koordinate der ersten Ecke des Dreiecks
  - `param[1]`: y-Koordinate der ersten Ecke des Dreiecks
  - `param[2]`: x-Koordinate der zweiten Ecke des Dreiecks
  - `param[3]`: y-Koordinate der zweiten Ecke des Dreiecks
  - `param[4]`: x-Koordinate der dritten Ecke des Dreiecks
  - `param[5]`: y-Koordinate der dritten Ecke des Dreiecks

Die Seiten der Rechtecke sind immer parallel zu einer Koordinatenachse. Nicht genutzte Parameter (beim Kreis und beim Rechteck) können beliebige Werte haben.

Problem 1 (9 Punkte)

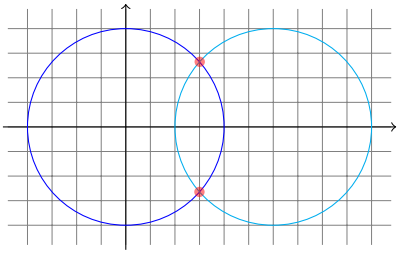
Schreiben Sie eine C-Funktion

`point_t *cutpoints(shape_t shape_a, shape_t shape_b, int *num)`

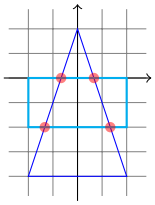
die für die Formen `shape_a` und `shape_b` ein Zeiger auf ein Array von Punkten, die beiden Formen gemeinsam haben (also Schnitt- und Berührungspunkte) zurückgibt, oder `NULL`, wenn es keine derartigen Punkte gibt. In `*num` soll die Anzahl der gefundenen Punkte geschrieben werden.

Beispiele

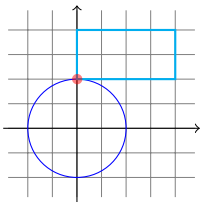
- `shape_a={s_circle, {0.0, 0.0, 2.0}}, shape_b={s_circle, {3.0, 0.0, 2.0}}` ⇝ `{{1.5,1.322},{1.5,-1.322}}`, `*num=2`



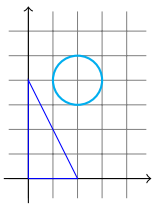
- `shape_a={s_triangle, {-1.0, -2.0, 0.0, 1.0, 1.0, -2.0}}, shape_b={s_rectangle, {-1.0, 0.0, 2.0, 1,0}}` ⇝ `{{-0.333, 0.0},{-0.666, -1.0},{0.333, 0.0},{0.666, -1.0}}`, `*num=4`



- `shape_a={s_circle, {0.0, 0.0, 1.0}}, shape_b={s_rectangle, {0.0, 2.0, 2.0, 1.0}}` ⇝ `{{0.0 ,1.0}}`, `*num=1`



- `shape_a={s_triangle, {0.0, 0.0, 1.0, 0.0, 0.0, 2.0}}, shape_b={s_circle, {1.0, 2.0, 0.5}}` ⇝ `NULL`, `*num=0`



- Die Verwendung ihrer Funktionen könnte z.B. durch folgenden Code erfolgen:

```
#include "cp.h"
#include <stdio.h>

int main(){
  shape_t circle={s_circle, {0.0, 0.0, 1.0}};
  shape_t rectangle={s_rectangle, {1.0, 1.0, 3.0, 2.0}};
  shape_t triangle={s_triangle, {1.0, 1.0, 3.0, 2.0, 2.0, -1.0}};
  int num1, num2, num3;
```

```
  point_t *points1, *points2, *points3;

  points1 = cutpoints(circle, rectangle,  &num1);
  points2 = cutpoints(circle, triangle,  &num2);
  points3 = cutpoints(triangle, rectangle,  &num3);

  printf("The circle and the recangle have %d points in common.", num1);
  printf("The circle and the triangle have %d points in common", num2);
  printf("The triangle and the rectangle have %d points in common, namely:", num3);
  for (int i=0; i<num3; i++){
    printf(" (%lf,%lf)\n",points3[i].x, points3[i].y);
  }

  if (points1 != NULL) free(points1);
  if (points2 != NULL) free(points2);
  if (points3 != NULL) free(points3);
  return 0;
}
```

Beachten Sie bei der Lösung folgende **Bedingungen**:

- Die Funktion (und evtl. nötige weitere Funktionen) soll in **einer** Quelldatei enthalten sein.
- Die Datei, die Sie (archiviert, siehe allgemeine Bedingungen) einreichen, **muss** den Namen `cutpoint.c` haben.
- Ihr Code wird als Modul übersetzt und darf **keine** `main()`-Funktion, **keine** globalen Variablen besitzen, und **keine** Ausgabe enthalten. Falls Sie für Debugging-Zwecke `main()` und/oder Ausgaben benutzen, löschen Sie entsprechende Codeteile vor der Abgabe oder kommentieren Sie diese aus!
- Die von `cutpoints()` zurückgegebenen Liste muss auch nach einem weiteren Aufruf von `cutpoints()` noch gültig sein.
- Sie dürfen alle Funktionen der Standardbibliothek und der Mathematikbibliothek nutzen[1], aber **keine weiteren** Bibliotheken.
- Sie dürfen **nicht** die Signaturen der zu definierenden Funktion **ändern**.
- Ihr Code sollte von beliebigen Hauptfunktionen aus nutzbar sein.

Hinweise:

- Die Übersetzungsoptionen behandeln alle Warnungen als Fehler. Stellen Sie sicher, dass ihr Code keine Warnungen generiert.
- Sie dürfen davon ausgehen, dass die Formen nicht entartet sind (z.B. Kreisradius=0.0)
- Sie dürfen davon ausgehen, dass sich schneidende Seiten von Rechtecken und Dreiecken nie auf der gleichen Gerade liegen (d.h. unendlich viele gemeinsame Punkte haben)
- Sie können die Headerdatei, die Rohfassung des Codes sowie für Ihre Tests den Quelltext der `main()`-Funktion auf der Aufgabenseite herunterladen:
  `https://osg.informatik.tu-chemnitz.de/lehre/aup/Templates/cp.h`
  `https://osg.informatik.tu-chemnitz.de/lehre/aup/Templates/cutpoint.c`
  `https://osg.informatik.tu-chemnitz.de/lehre/aup/Templates/cp-main.c`
- Sie können bei dieser Aufgabe bis zu acht Punkte erhalten. Entdeckte Betrugsversuche führen zu 0 Punkten.

[1] Es gilt die Standardbibliothek in der Konfiguration der Testmaschine von OpenSubmit.