Dr. U. Köthe *Heidelberg, 21. Dezember 2016*

Übungsblatt 9

Aufgabe 9.1 Rectangle Klasse [20 Punkte]

In dieser Aufgabe sollen Sie die Grundlagen der objektorientierte Programmierung üben, indem Sie eine Klasse `Rectangle` implementieren. Geben Sie Ihre Lösung in der Datei `rectangle.cpp` ab. Ein Grundgerüst dieser Datei, das Sie nur noch vervollständigen müssen, finden Sie auf Moodle. Diese Datei enthält außerdem den Code der Klasse `Point` aus der Vorlesung, auf dem diese Aufgabe aufbaut.

Die Klasse `Rectangle` repräsentiert achsparallele Rechtecke durch zwei gegenüberliegende Ecken, die in den privaten Member-Variablen `p0_` (für die linke untere Ecke) und `p1_` (für die rechte obere Ecke) gespeichert sind. Sind eine oder beide Koordinaten von `p0_` und `p1_` gleich, entartet das Rechteck in eine Linie bzw. einen Punkt. Befindet sich `p1_` hingegen links oder unterhalb von `p0_`, ist das Rechteck ungültig (analog zu `NaN` bei reellen Zahlen).

(a) Implementieren Sie die folgenden Konstruktoren:
- `Rectangle()`: der Standardkonstruktor initialisiert ein entartetes Rechteck beim Punkt (0,0).
- `Rectangle(Point const & p1)`: Der Konstruktor mit einem Argument initialisiert ein Rechteck, das sich von Punkt (0,0) bis zum gegebenen Punkt `p1` erstreckt (oder ungültig ist, falls `p1` eine negative Koordinate hat).
- `Rectangle(Point const & p0, Point const & p1)`: Der Konstruktor mit zwei Argumenten initialisiert ein Rechteck, das sich vom gegebenen Punkt `p0` zum gegebenen Punkt `p1` erstreckt oder ungültig ist.

(b) Implementieren Sie die Getter-Funktionen `x0()`, `y0()`, `x1()`, `y1()`, die die jeweiligen Koordinaten der Ecken des Rechtecks zurückgeben. Schreiben Sie zwölf Tests in der Funktion `testRectangle()`, indem Sie mit jedem der drei Konstruktoren ein Rechteck erzeugen und prüfen, dass die Getter die erwarteten Werte zurückgeben.

(c) Implementieren Sie eine freie Funktion `std::string to_string(Rectangle const & r)`, die das Rechteck in einen String der Form `"[x0:x1, y0:y1]"` umwandelt (diese Funktion ist zum Debuggen sehr nützlich). Die Schreibweise `x0:x1` soll dabei ausdrücken, dass sich das Rechteck von `x0` bis `x1` erstreckt. Fügen Sie in `testRectangle()` drei Tests für `to_string()` hinzu.

(d) Implementieren Sie die Member-Funktionen `width()` und `height()`, die Breite und Höhe des Rechtecks zurückgeben (das können negative Werte sein, wenn das Rechteck ungültig ist). Fügen Sie in `testRectangle()` vier Tests hinzu.

(e) Implementieren Sie eine Member-Funktion `is_valid()`, die genau dann `false` zurückgibt, wenn das Rechteck ungültig ist, und eine Member-Funktion `area()`, die die Fläche des Rechtecks zurückgibt. Ist das Rechteck entartet, soll `area()` den Wert

`0.0` zurückgeben, ist es ungültig, den Wert `-1.0`. Fügen Sie in `testRectangle()` sechs Tests für diese Funktionen hinzu.

(f) Implementieren Sie die Member-Funktionen `transpose()`, die ein neues Rechteck mit vertauschten x- und y-Koordinaten zurückgibt, und `translate(Point const & v)`, die ein um den gegebenen Vektor `v` verschobes Rechteck zurückgibt. Fügen Sie in `testRectangle()` vier Tests für diese Funktionen hinzu.

(g) Implementieren Sie die Member-Funktionen

```
bool contains(Point const & p)
```

die genau dann `true` zurückgibt, wenn der Punkt `p` innerhalb des Rechtecks oder auf dessen Rand liegt, und

```
bool contains(Rectangle const & other)
```

die genau dann `true` zurückgibt, wenn das Rechteck `other` komplett im Rechteck `(*this)` enthalten ist (`other` darf dabei auch auf dem Rand von `(*this)` liegen). Ist eines der beteiligten Rechtecke ungültig, soll stets `false` zurückgegeben werden. Fügen Sie in `testRectangle()` sechs Tests für diese Funktionen hinzu.

(h) Implementieren Sie die folgenden freien Funktionen (d.h. außerhalb der `Rectangle`-Klasse):

- `Rectangle rectangle_union(Rectangle const & r1, Rectangle const & r2)`
  die das kleinste Rechteck zurückgibt, das sowohl `r1` als auch `r2` enthält, und
- `Rectangle rectangle_intersection(Rectangle const & r1, Rectangle const & r2)`
  die den Durchschnitt der beiden Rechtecke zurückgibt, also das größte Rechteck, das sowohl in `r1` als auch `r2` enthalten ist. Überschneiden sich die beiden Rechtecke nicht, soll ein beliebiges ungültiges Rechteck zurückgegeben werden.

Sind `r1` oder `r2` ungültig, soll das resultierende Rechteck ebenfalls ungültig sein. Fügen Sie in `testRectangle()` acht Tests für diese Funktionen hinzu. Tipp: Mit `std::min` und `std::max` schafft man beide Funktionen in wenigen Zeilen und ohne if/else.

Aufgabe 9.2 Geschenke unterm Weihnachtsbaum [20 Punkte]

Wir wollen jetzt die `Rectangle`-Klasse benutzen, um einen Algorithmus zum optimalen Platzieren von Geschenken unterm Weihnachtsbaum zu implementieren. Kopieren Sie Ihre Lösung `rectangle.cpp` von Aufgabe 9.1 in eine neue Datei `gifts.cpp`, die als Ausgangspunkt für diese Aufgabe dient. Geben Sie die vervollständigte Datei `gifts.cpp` ab.

In dieser Aufgabe gelten folgende Regeln: der Tisch, der Weihnachtsbaumständer und die Geschenke sind in der auf Moodle verfügbaren Datei `gifts.txt` als Rechtecke gegeben. Die Datei sieht etwa so aus (mit Größen in Zentimetern):

```
Rectangle              table(Point(120.0, 100.0));
Rectangle              tree_stand(Point(30.0, 30.0));
std::vector<Rectangle> gifts = { Rectangle(Point(20.0, 10.0)),
                                 Rectangle(Point(10.0, 11.0)),
                                 ... };
```

Sie sollen einen Teil der Geschenke auswählen und achsparallel so auf den Tisch legen, dass sie sich nicht überlappen und auch nicht an der Tischkante überstehen. Außerdem muss der

Weihnachtsbaum (genauer: der Weihnachtsbaumständer) auf dem Tisch ohne Überlapp Platz finden. Offensichtlich reicht die Tischfläche nicht aus, um sämtliche Geschenke unter diesen Randbedingungen auf dem Tisch zu platzieren. Die Aufgabe besteht deshalb darin, die Auswahl und Platzierung mit dem folgenden Algorithmus so geschickt vorzunehmen, dass auf dem Tisch möglichst wenig Freiraum bleibt (die übrig gebliebenen Geschenke kommen zurück in den Sack :-). Der Algorithmus ist im Artikel "A Thousand Ways to Pack the Bin" von J. Jylänki unter dem Namen GUILLOTINE-BSSF-SAS-RM beschrieben. Er liefert Lösungen, die der optimalen Lösung nahekommen – die Bestimmung einer absolut optimalen Lösung ist hier nicht verlangt, weil dieses Problem zur Klasse der NP-harten Probleme gehört und daher sehr schwierig ist.

(a) Schritt 1: Fügen Sie den Inhalt der Datei `gifts.txt` in `gifts.cpp` ein. Erzeugen Sie in `main()` drei Variablen vom Typ `std::vector<Rectangle>` für die noch zu platzierenden Objekte (`to_be_placed`), die bereits platzierten Objekte (`already_placed`) und den verbleibenden Freiraum (`free_rectangles`). Initialisieren Sie diese Arrays (z.B. ist der freie Raum anfangs der ganze Tisch).

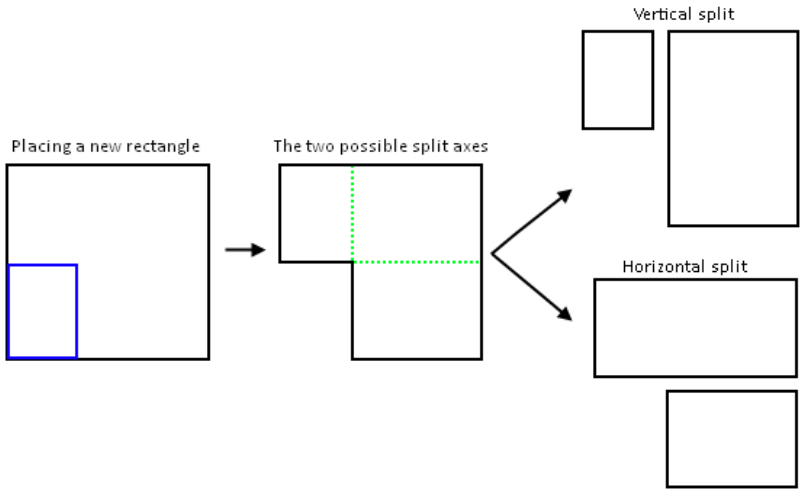
(b) Schritt 2: Implementieren Sie eine Funktion
`double bssf_score(Rectangle const& free, Rectangle const& obj)`
die den BSSF-Score dafür berechnet, wie günstig es wäre, das Objekt `obj` in den freien Raum `free` zu legen. Man bestimmt dazu die Restbreite $w_{\text{free}} - w_{\text{obj}}$ und die Resthöhe $h_{\text{free}} - h_{\text{obj}}$ und gibt den kleineren der beiden Werte zurück. Wenn das Objekt allerdings nicht in den freien Raum hineinpasst, muss "unendlich" zurückgegeben werden (repräsentieren Sie "unendlich" durch eine sehr große Zahl, z.B. mit der Definition `double big_number = 1e300;`).

(c) Schritt 3: Bestimmen Sie für alle Paare `(free, obj)`, die aus den Elementen der Arrays `free_rectangles` und `to_be_placed` gebildet werden können, den BSSF-Score und merken Sie sich die Indizes der Kombination mit dem kleinsten (=besten) Score.

(d) Schritt 4: Wiederholen Sie diese Berechnung, aber transponieren Sie jedes Objekt vor dem Aufruf von `bssf_score()` (es kann günstiger sein, ein Geschenk hochkant statt quer hinzulegen). Ergibt sich hierbei eine Kombination mit kleinerem Score als in (c), merken Sie sich diese Indizes sowie die Tatsache, dass das Objekt transponiert hingelegt werden soll.

(e) Schritt 5: Ist der beste Score jetzt "unendlich", kann kein Objekt platziert werden, und es geht mit Teilaufgabe (h) weiter. Sonst seien `(best_free, best_obj)` die Indizes mit dem besten Score. Kopieren Sie das Objekt `to_be_placed[best_obj]`, transponieren Sie es, falls nötig, mit `rect.transpose()` und verschieben Sie es mit `rect.translate()` so, dass seine linke untere Ecke mit der linken unteren Ecke von `free_rectangles[best_free]` übereinstimmt. Um das Platzieren abzuschließen, aktualisieren Sie die Arrays wie folgt:

- Fügen Sie das verschobene und evtl. transponierte Objekt in das Array `already_placed` ein.
- Entfernen Sie das Objekt `best_obj` aus dem Array `to_be_placed`.
- Entfernen Sie den Freiraum `best_free` aus dem Array `free_rectangles`.
- Meist füllt das Objekt den Freiraum nicht vollständig aus – es bleibt ein L-förmiges Gebiet übrig, das entweder mit einem horizontalen oder einem vertikalen Schnitt in zwei neue freie Rechtecke zerlegt werden kann:



(Abbildung aus J. Jylänki). Die Wahl der Zerlegung bestimmt die SAS-Regel: Gilt für die ursprüngliche Breite und Höhe des Freiraums $w_{\text{free}} < h_{\text{free}}$, wird der horizontale Schnitt ausgeführt, andernfalls der vertikale. Erzeugen Sie die beiden neuen freien Rechtecke entsprechend und fügen Sie sie in das Array `free_rectangles` ein.

(f) Schritt 6 (optional): Die SAS-Regel führt häufig zu einer starken Fragmentierung des Freiraums. Die RM-Regel dient dazu, kleine freie Rechtecke wieder in größere zu vereinigen: Testen Sie für alle Paare von Rechtecken aus `free_rectangles`, ob ihre Vereinigung mittels `rectangle_union()` ein Rechteck liefert, das exakt die beiden ursprünglichen Rechtecke enthält (mit anderen Worten, diese Rechtecke hatten eine gemeinsame Kante). Ist dies der Fall, entfernen Sie die beiden Rechtecke aus dem Array `free_rectangles` und fügen stattdessen das vereinigte Rechteck ein.

(g) Schritt 7: Wiederholen Sie die Schritte ab (c), bis keine weiteren Geschenke mehr platziert werden können. Stellen Sie dabei sicher, dass der Weihnachtsbaum(ständer) auf jeden Fall platziert wird.

(h) Schritt 8: Testen Sie mittels `assert()`, ob bei Ihrer Lösung (d.h. bei den Objekten in `already_placed`) die Randbedingungen erfüllt sind:

- Prüfen Sie, ob der Weihnachtsbaumständer in `already_placed` enthalten ist.
- Prüfen Sie mit der Funktion `table.contains()`, dass kein Objekt an der Tischkante übersteht.
- Prüfen Sie mit der Funktion `rectangle_intersection()` für alle Paare von Objekten in `already_placed`, dass es keine Überlappungen gibt.

Geben Sie Ihre Lösung (d.h. die Elemente des Arrays `already_placed`) mit Hilfe der Funktion `to_string(rectangle)` auf die Konsole aus. Berechnen Sie außerdem die Fläche des verbleibenden Freiraums und geben Sie diese ebenfalls aus.

(i) Bonusaufgabe: Variieren Sie die Regeln für das Platzieren oder experimentieren Sie mit anderen Regeln, um den verbleibenden Freiraum weiter zu reduzieren. Der Artikel von J. Jylänki enthält dazu viele Vorschläge, aber Ihrer Kreativität sind keine Grenzen gesetzt.

(j) Bonusaufgabe: Zeichnen Sie Ihre Lösung auf Papier oder mit einem Programm.

Bitte laden Sie Ihre Lösung spätestens bis 11. Januar 2017, 9:00 Uhr in Moodle hoch.