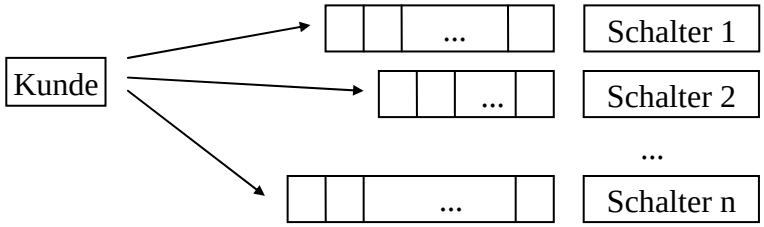
Ereignisgesteuerte Simulation

In einer Bank gibt es n Bankschalter, bei denen Kunden bedient werden können. Bei größerem Kundenandrang ergeben sich an jedem Schalter Warteschlangen.



Warteschlangen

Die Bank ist bestrebt, einerseits möglichst wenige Schalter zu öffnen und andererseits die Wartezeit der Kunden nicht allzu hoch werden zu lassen. Dazu soll in einer Simulation die durchschnittliche Wartezeit der Kunden bei n geöffneten Schalter ermittelt werden.

In einer statistischen Untersuchung werden die Ankunftsrate der Kunden und der Aufwand ihrer Bedienung am Schalter ermittelt. Für die Simulation wird daraus üblicherweise ein mathematisches Modell entwickelt: z.B. der Abstand zwischen den Ankunftszeitpunkten zweier Kunden ist exponentiell verteilt und die Aufwandszeiten für die Bedienung sind normalverteilt. Wir wollen einfachheitshalber annehmen, daß bereits eine Datei zur Verfügung steht, die für jeden Kunden seinen Ankunftszeitpunkt (in Min.) und seine Bedienzeit (in Min.) chronologisch sortiert enthält. Beispiel:

Ankunftzeitpunkt	Bedienzeit
5	6
7	5
8	10
10	3

Zwei Ereignisarten müssen in unserer Simulation betrachtet werden:

- Ankunftsereignis: Eine Kunde betritt die Bank und reiht sich an einem der Schalter ein.
- Abgangsereignis: Eine Kunde wird an einem Schalter fertig bedient und verläßt die Bank.

In einem naiven Simulationsansatz würde man die Zeit kontinuierlich (minutenweise) verlaufen lassen und dabei auftretende Ereignisse behandeln:

```
time = 0;
while (time <= endTime)
{
      if (es gibt zum Zeitpunkt time ein Ankunftsereignis)
            behandle Ankunft;

      if (es gibt zum Zeitpunkt time ein Abgangsereignis)
            behandle Abgang;

      time++;
}
```

Es ist jedoch zweckmäßiger zum jeweils nächsten Ereigniszeitpunkt zu springen (ereignisgesteuerte Simulation):

```
while (es gibt noch unbehandelte Ereignisse)
{
        e = nächstes Ereignis;

        if (e ist ein Ankunftsereignis)
                behandle Ankunftsereignis;

        if (e ist ein Abgangsereignis)
                behandle Abgangsereignis;
}
```

In der ereignisgesteuerten Simulation ist ein effizienter Zugriff auf das jeweils nächste Ereignis wichtig. Dazu werden in einer Ereignisliste die nächsten Ereignisse (nicht unbedingt alle) chronologisch geordnet gespeichert. Typische Operationen für eine Ereignisliste sind:

insert(Ereignis):	fügt neues Ereignis ein.
next(Ereignis):	liefert das chronologisch nächste Ereignis und löscht es aus der Ereignisliste. Ist die Ereignisliste leer, wird 0 sonst 1 zurückgeliefert.

Gehen wird davon aus, daß ein Kunde sich immer an der kürzesten Schlange anstellt (gibt es mehrere kürzeste Schlangen, dann wird die erstbeste gewählt), dann kann die Simulationsschleife wie folgt präzisiert werden:

```
hole aus Datei erstes Ankunftsereignis
und füge es in Ereignisliste ein;

while (Ereignisliste nicht leer)
{
        e = nächstes Ereignis;

        if (e ist ein Ankunftsereignis)
        {
                füge e in kürzeste Warteschlange ein;

                falls e einziger Kunde in Warteschlange, dann wird er gleich
                bedient; erzeuge daher neues Abgangsereignis
                und füge es in Ereignisliste ein;

                hole aus Datei nächstes Ankunftsereignis
                und füge es in Ereignisliste ein;
        }
        else // e ist ein Abgangsereignis
        {
                entferne Kunde aus Warteschlange;

                erzeuge für nächsten Kunden aus Warteschlange neues
                Abgangsereignis und füge es in Ereignisliste ein;

        }

}
```

Schreiben Sie für die oben beschriebene Bedienung von Bankkunden ein Simulationsprogramm.

Im einzelnen soll folgendes gelöst werden:

a) Eine Klasse EventList mit den oben beschriebenen Operationen insert und next. Verwenden Sie dabei ein Strukturdatentyp Event (Ereignis) mit folgenden Komponenten:

- type: Ereignisart: 'I' (Ankunft) oder 'O' (Abgang)
- time: Ereignis-Zeitpunkt
- d: Bedienzeit; nur bei Ankunftsereignis
- nrSchalter: Nummer des Schalters, in dem sich der Kunde eingereiht hat; nur bei Abgangsereignis

b) Eine Klassenschablone Queue für die Warteschlangen mit einer zusätzlichen Methode length(), die die Größe einer Schlange zurückliefert.

c) Mit SimulationMain.cpp steht Ihnen bereits ein Hauptprogramm zur Verfügung gestellt, das unter Verwendung dieser Klassen die oben beschriebene Simulationsschleife realisiert. Folgende Punkte sind zu ergänzen:

- Anzahl der Bank-Schalter festlegen.
- Kürzeste Warteschlange berechnen.
- Visualisierung der Längen der einzelnen Warteschlangen durch entsprechende Ausgaben von **'*'** zu den einzelnen Simulationszeitpunkten. Beispiel:

```
Zeit:        35.34 Min
Ereignis:    Kunde trifft ein und reiht sich in Schalter 2 ein.

Schalter1:   ***
Schalter2:   **
Schalter3:   ****
```

  Um den visuellen Effekt zu verstärken, geben Sie die Ausgabe immer auf einen gelöschten Bildschirm aus (unter Unix: **`#include <stdlib.h>`** und **`system("clear");`**) und verzögern die Ausgabe mit einer Warteschleife.

  Zur Kontrolle lenken Sie die Ausgabe auch auf eine Datei um.

- Ermittlung und Ausgabe der Anzahl der insgesamt bedienten Kunden und der durchschnittlichen, minimalen und maximalen Warte- und Bedienzeiten der Kunden.

d) Mit KundenGeneratorMain.cpp können Sie Ankunftszeitpunkte und Bedienzeiten für die eintreffenden Kunden generieren, die dann chronologisch geordnet in eine Datei mit dem Namen kunden.dat (Beispiel siehe oben) geschrieben werden.

Hinweis zum Übersetzen von Schablonen (Templates):

Bei Verwendung des MS-Visual-C++- oder des GNU-C++-Compilers sollte Ihre Header-Datei (z.B. queue.h) außer der Definition der Klassenschablone auch die Definition der Methoden-schablonen enthalten. Die cpp-Datei (z.B. queue.cpp) wird dann nicht benötigt.