DAP2 Praktikum für ETIT und IKT, SoSe 2018, Langaufgabe L3

Fällig am 11.06. um 14:15

Die Regeln für das Programmieren gelten unerbittlich weiter.

Langaufgabe L3.1 (1 Punkt)

- Extrahieren Sie `BeispielDaten.zip` in dem Verzeichnis wo sich das exe-File des in dieser Aufgabe zu erstellenden Programms `Scheduler` befinden wird. In den Dateien befinden sich zeilenweise durch ein Komma getrennte Anfangs- und Endzeitpunkte von Jobs. Die Werte sind ganzzahlig.
- Schreiben Sie eine Klasse `JobSpecification` , die die zeitlichen Spezifikationen eines Jobs jeweils in einer `int FirstTime` und `int SecondTime` Variable abspeichern kann. Beide Variablen dürfen `public` sein.
- Schreiben Sie zu `Jobspecification` als Member-Funktion `operator<` eine Vergleichsfunktion, die die `SecondTime` zweier Jobs vergleicht, und `true` zurückliefert, wenn die `SecondTime` des ersten Job früher ist als die `SecondTime` des zweiten Jobs. Im anderen Fall soll sie `false` zurückliefern.
- Schreiben Sie durch Vererbung von `std::vector` eine Klasse

  ```
  class JobVector: public vector<JobSpecification>
  ```

  die geeignet ist, um Jobs abzulegen. Die Klasse darf einen Konstruktor

  ```
  JobVector() { ; }
  ```

  beinhalten, der einen leeren `JobVector` liefert. Sie muss aber den Konstruktor

  ```
  JobVector(char *Filename)
  ```

  bereitstellen, der aus der mit `Filename` bezeichneten Datei die `FirstTime` und die `SecondTime` der mit einem Komma getrennten Zahlenpaare aus der Datei ausliest und sie im `JobVector` ablegt. Weitere Konstruktoren sind nicht erlaubt.

  Es ist sinnvoll, für das Lesen der Datei die Header

  ```
  #include <iostream>
  #include <fstream>
  #include <sstream>
  #include <limits>
  #include <cstdlib>
  #include <cstring>
  ```

  zu verwenden. Sie können dann aus Textdateien elegant unter Nutzung von durch Sie in die richtige Reihenfolge zu bringenden und zu ergänzenden Fragmenten wie

  ```
  ifstream file(Filename);

  string line;
  while(std::getline(file,line,'\n')) {/* Your Code */ }
  ```

```
if (line.empty()) { /* Do something or don't */ };

istringstream buffer(line);
buffer.exceptions(istringstream::failbit |
                  istringstream::badbit);

int IntegerVariable;
buffer >> dec >> IntegerVariable;

buffer.ignore( /* Insert Your Code here! */ );

try {
        /* Anything that may fail */
}
catch(const char *what) { throw what; }
catch( ... ) { throw "Any Message You like."; }
```

die Werte aus der Datei auslesen. Es bleibt natürlich Ihnen überlassen, wie Sie den Konstruktor im Endeffekt implementieren. Allerdings sind C-typische Funktionen wie `open`, `fopen`, `sscanf`, `scanf`, `atoi` etc. nicht erlaubt.

- Damit Sie die Funktion der Klasse nachweisen können, schreiben Sie ein geeignetes `main`, dem der Dateinamen `Filename` übergeben werden kann. Bei fehlerhaften Dateien soll sich das Programm so verhalten, wie am Ende der Aufgabenstellung in den Beispielen gezeigt ist. Wenn Sie Teil 2 dieser Langaufgabe erfüllen, benötigen Sie natürlich kein eigenes `main` zum Funktionsnachweis.

- `assert` können Sie diesmal weglassen.

Langaufgabe L3.2 (1 Punkt)

- Implementieren Sie eine Funktion

  `JobVector IntervalScheduling(JobVector &Jobs)`

  und eine Funktion

  `JobVector LatenessScheduling(JobVector &Jobs)`

  die die aus der Vorlesung bekannten Scheduling-Verfahren implementieren und die geplanten Jobs zurückliefern. Der `JobVektor Jobs` beinhaltet die eingelesenen Jobs.
- Beachten Sie, dass die `Jobs` für beide Algorithmen aufsteigend nach `SecondTime` sortiert werden müssen. Verwenden Sie zum Sortieren der eingelesenen Jobs die Funktion `std::sort` aus `#include <algorithm>` . Verwenden Sie keine eigene Sortierfunktion! Wenden sie `sort` direkt auf die Klasse `JobVector` an!
- Bei `IntervalScheduling` beinhaltet `Jobs` die einzuhaltenden Start- bzw. Endzeitpunkte der Jobs aus der ausgelesenen Datei. Jobs die nicht unterzubringen sind, werden verworfen.
- Bei `LatenessScheduling` entspricht die in `Jobs` enthaltene `SecondTime` die Deadline für den Job. Die Dauer des Jobs entspricht der Differenz

`SecondTime-FirstTime` . Alle Jobs werden untergebracht. Bei zu zahlreichen Jobs ergeben sich naturgemäß für bestimmte Jobs Verspätungen (Delays).

- Der Programmaufruf hat als ersten Parameter den `Filenamen` und als zweiten, optionalen Parameter die Optionen `–i` oder `-l` für die Auswahl der beiden Algorithmen. Defaultmäßig ist `IntervalScheduling` aktiviert.
- Am Ende des Programmlaufes werden bei `IntervalScheduling` die die Nummer des Jobs in der geplanten Ausführungsreihenfolge sowie die Start- und Endzeiten der Jobs ausgegeben. Zudem wird am Ende die Anzahl der zu verwerfenden Jobs ausgegeben.
- Bei `LatenessScheduling` ist neben der Jobnummer die Dauer des Jobs, der zur Deadline passende Startzeitpunkt, die Deadline selber sowie das Delay auszugeben. Am Ende ist das maximal aufgetretene Delay auszugeben.
- Halten Sie sich bei der Ausgabe peinlich genau an die nachfolgenden Beispiele.
- Beispiele für Scheduler sind:

```
>Scheduler.exe
Usage: Scheduler Filname [ -i | -l ]
  Filename must be a name of a file
  holding a 2 column csv list with start and end times.
 Options:
  -l : Lateness Scheduling
  -i : Interval Scheduling

>Scheduler.exe Err1
Garbage in file.

>Scheduler.exe Err2
Nothing usable in file.

>Scheduler.exe Err3
End of Job prior to Start of Job.

>Scheduler.exe Err4
File not found.

>Scheduler.exe Err5
Job: 0 is from 10 to 11
Job: 1 is from 13 to 14
0 Jobs were discarded.

>Scheduler.exe Err6
Garbage in file.

>Scheduler.exe datenBsp4.zahlen –u
Wrong Switch, use only one of [ -i | -l ]

>Scheduler.exe datenBsp4.zahlen
Job: 0 is from 1 to 6
Job: 1 is from 8 to 13
Job: 2 is from 15 to 20
Job: 3 is from 22 to 27
Job: 4 is from 29 to 33
Job: 5 is from 39 to 43
7 Jobs were discarded.
```

```
>Scheduler.exe datenBsp4.zahlen –i
Job: 0 is from 1 to 6
Job: 1 is from 8 to 13
Job: 2 is from 15 to 20
Job: 3 is from 22 to 27
Job: 4 is from 29 to 33
Job: 5 is from 39 to 43
7 Jobs were discarded.

>Scheduler.exe datenBsp4.zahlen –l
Job: 0 takes 5 had Deadline of 6 is from 0 to 5
Job: 1 takes 4 had Deadline of 9 is from 5 to 9
Job: 2 takes 4 had Deadline of 9 is from 9 to 13 is delayed by 4
Job: 3 takes 4 had Deadline of 9 is from 13 to 17 is delayed by 8
Job: 4 takes 4 had Deadline of 9 is from 17 to 21 is delayed by 12
Job: 5 takes 5 had Deadline of 13 is from 21 to 26 is delayed by 13
Job: 6 takes 4 had Deadline of 16 is from 26 to 30 is delayed by 14
Job: 7 takes 5 had Deadline of 20 is from 30 to 35 is delayed by 15
Job: 8 takes 4 had Deadline of 23 is from 35 to 39 is delayed by 16
Job: 9 takes 4 had Deadline of 23 is from 39 to 43 is delayed by 20
Job: 10 takes 5 had Deadline of 27 is from 43 to 48 is delayed by 21
Job: 11 takes 4 had Deadline of 33 is from 48 to 52 is delayed by 19
Job: 12 takes 4 had Deadline of 43 is from 52 to 56 is delayed by 13
Max. Delay is : 21
```