Assignment 2

Summary

- **Due date: Sun, Sep 25, 23:59** (Adelaide time)
- **Forming group:** Take this survey (https://myuni.adelaide.edu.au/courses/75053/quizzes/141591) to let us know your preference (due Fri, Aug 26, 17:00)
- **Task**: Download Assignment 2 files (https://myuni.adelaide.edu.au/courses/75053/files/11345668/download) . Write a CPU scheduler in C++ in a team
- **Submission**:
  - The team should collectively develop "*scheduler.cpp*"
  - Make sure this file starts with comments that list team members and team name. See example in provided "*baseline.cpp*"
  - We will reject your submission if we cannot identify your team members
  - One of team members should submit this file through MyUni
  - It is OK if accidentally more than one team member submits the file. Note that only the last submission will be used
- **Marking**:
  - This assignment weights 15% of the final mark
  - This assignment will be marked automatically. Thus, testing is strongly recommended
  - Each of the team members will receive the same mark
  - We will reject submissions not made in a team
- **Academic Integrity**:
  - OK to use slides, textbook, etc.
  - **Not OK to get someone outside your team help you**
  - **Not OK to copy code** from outside your team
  - If we detect code duplicated between teams (other than provided code), both teams will receive 0 marks
- **Got a question?** Please use the discussion board (https://myuni.adelaide.edu.au/courses/75053/discussion_topics)

*** make your submission here (https://myuni.adelaide.edu.au/courses/75053/assignments/294976) ***

Setup

Consider a single person arcade game machine located in a shopping centre. While many customers wish to play, there is only one machine, so it is a shared resource. Customers willing to play arrive at random times and if the machine is occupied, they need to wait. This is a very advanced shopping centre and the use of the machine is facilitated by a robot manager (programmed by you!). In particular, the robot may decide that a customer is playing for too long and ask the customer to release the machine and move back to the waiting queue.

Based on the membership with the gaming company, there are two customer priorities: high for members and regular for non-members. Also, to facilitate scheduling, each customer must declare how many time units they will be playing for in total. Once a customer spends this amount of time at the machine (in one or multiple sessions), they leave.

You need to write a program that decides which customer should be playing when.

Data Format

An input data file is a simulated list of arriving customers. Input data files will be named "*data_*.txt*", where asterisk will be replaced by a random number (not relevant for you). Each row in such a file describes customer ID, priority (**0 for high, 1 for regular**), arrival time, and the desired amount of play time.

Example:

```
c00 0 6 23
c01 1 20 13
c02 1 28 17
c03 1 38 28
...
```

In the first line, customer c00 arrived at time 6. This customer is a member and has a high priority. This customer wishes to play for 23 units of time in total. Note that 0 denotes high priority, while 1 denotes regular priority.

An output data file is a log of machine usage as a result of a particular scheduling approach. For a given input file, an output file can be generated either using a program in "*baseline.cpp*" (provided) or your program in "*scheduler.cpp*" (your submission). Each such output file is also called a **scheduling**.

Each row of the output file describes who was occupying the machine at a particular time slot (starting from 0). If the machine was not occupied, the row will list -1, otherwise it will show customer ID (without letter "c").

Example:

```
0 -1
1 -1
2 -1
3 -1
4 -1
5 -1
```

```
6 0
7 0
...
```

In this example, the machine was unoccupied in time slots 0, ..., 5. Then starting from time slot 6, the machine was used by customer 0.

Valid Scheduling

- If a customer arrives at time X, it can start occupying the machine starting from time slot X, but not earlier
- If a customer wishes to play for N time slots, the total number of time slots occupied by that customer in the output must be N
- Play time demand must be satisfied as above for every customer in the input file
- There not need to be a gap between customers, two consecutive time slots can be occupied by two different customers
- The output file must not contain customers that were not present in the input
- The output file must list all time frames sequentially, starting from 0. There should be no time gaps. The output must cover enough time frames to satisfy demand of all customers
- The last time slot in your scheduling should be after all customers are satisfied, and it should denote an empty machine (-1 as customer ID)
- If the above conditions are satisfied, the scheduling is considering valid. However, some valid scheduling are better than others (see below)

Task

You need to write a program that takes an input data file and generates an output file with a valid scheduling for this input. Implement your program in a single file named "*scheduler.cpp*". Your program should be a standalone console based application requiring exactly two command line arguments: name of the input file and name of the output file.

For example, suppose that your program is compiled into `./scheduler`, and you also have `./data_1111.txt` in the current directory. Then

```
> ./scheduler data_1111.txt out_1111.txt
```

should produce `out_1111.txt` in this directory, and this file should describe a valid scheduling with respect to input in `data_1111.txt`. For example, demand of each customer in `data_1111.txt` must be satisfied in `out_1111.txt`.

Marking

How to get 0 marks:

- If your submission is not made in a team of 3 or 4, you will receive 0 marks
- If we find copied code (other than provided) in two teams, both teams will receive 0 marks
- If the submitted "*scheduler.cpp*" doesn't compile or doesn't handle input arguments as requested, you will receive 0 marks

We will test your submission on 5 randomly generated test data files. These test data files will not be released, but they are generated by the same stochastic (random) process used to generate the provided data files. For each test file, we will compare your scheduling with the scheduling produced by "*baseline.cpp*" for the same input.

For each of the 5 test input files:

- Your program produces a valid output scheduling **within 5 minutes**: **+0.3 marks**
- Only if the above is achieved, we will proceed as follows:
  - Your scheduling is different from baseline: **+0.3 marks**
  - Your scheduling has a smaller total wait time as compared to the baseline: **+0.6 mark**
  - Your scheduling has a smaller longest response time as compared to the baseline: **+0.6 marks**
  - Your scheduling contains fewer switches between customers as compared to the baseline: **+0.6 marks**
  - Your scheduling has a smaller total wait time for high priority customers as compared to the total wait time for regular customers: **+0.6 marks** (this one is not compared against baseline)

Maximum total mark: 15 = 5 x (0.3 + 0.3 + 0.6 + 0.6 + 0.6 + 0.6)

Each member of the team will receive the same mark.

Definitions

- total wait time = sum of wait times for all customers in the given input file
- in some cases above, the total wait time includes only high priority or only regular priority customers
- wait time for a customer = number of time slots such that (i) the customer has arrived at or before that time slot (ii) customer's play time demand has not been satisfied by this time, and (iii) the customer is not scheduled on the machine at this slot
- longest response time = maximum response time across customers in the given input
- response time for a customer = index of the time slot where the customers plays for the first time minus index of time slot where the customer arrived
  - example: customer 0 arrived at time 5, but the first time they have a chance to play is time slot 10. Response time for this customer is 10 - 5 = 5
- number of switches = number of time slots (starting from 1), such that the occupation in the previous time slot doesn't match the occupation in this time slot
  - example: there are 3 switches in this schedule

```
0 -1
1 -1
2 0
3 0
4 1
5 -1
```

Hints

- Understand how "*baseline.cpp*" works, note comments in the provided files
  - Copy "*baseline.cpp*" as "*scheduler.cpp*" (your submission) then modify and improve it
  - However, you can also write your solution from scratch
  - Use provided "*data_*.txt*" files to test your submission. You might also want to generate more example input files
- Use the provided "*compute_stats.cpp*" to compute properties of your scheduling, such as total wait time
  - This program also checks whether the scheduling is valid
  - This program accepts two command line arguments: input data file and the output (scheduling) file generated for this input
  - You don't need to understand the code in detail, you can just compile and use this program
- You can implement any scheduling algorithm or a combination of algorithms
  - To get an inspiration, review different CPU scheduling algorithms from the lectures and think whether they can be useful to achieve a good mark
  - You can invent new algorithms

Other

If that matters to you, we will officially use C++14 standard. This is because C++14 is the default mode for g++ (versions 6.1 to 10 inclusive) and Clang. We expect that most students will use common C++ features available in any modern standard revision, so exact standard version won't matter. If you are a C++ guru, please avoid "exotic" C++ features that could potentially result in interpretation and compatibility issues.

*** make your submission here

(https://myuni.adelaide.edu.au/courses/75053/assignments/294976) ***

GOOD LUCK!