CSCI262 ASSIGNMENT THREE REPORT

Initial Input

1. How you are going to store the events and statistics internally. Explain both formats.

- The events and statistics are stored internally in a class called EventType. This class stores the following information about an event: event name, event type, event ID (a number), minimum, maximum, unit, weight, mean and standard deviation.
- Information on the types of events (C, D, E) is stored and set as an enum type. The only two variables that are affected by the event type are min and max, therefore we set union type for min and max that enables it to have the option between integer or double value.
- The default minimum and maximum for each event which does not have those established are set to -10,000,000 and 10,000,000 respectively.
- In addition, to keep track of the mapping of event IDs with event types, but still be able to quickly find the events by their name, we store the event details in two separate maps: **map <int, string>** and **map <string, pair<EventType, bool> >**. The first map stores the mapping between event ID and event name, the second stores the mapping between event name and the rest of the event details. That way, when we start reading in the statistics files, we are able to compare the strings to see if the event names match. If they do not match, we will report an error.

2. Potential inconsistencies between Events.txt and Stats.txt. You should attempt to detect those inconsistencies. If there are inconsistencies you are aware of but haven't attempted to detect them, note this in your report.

- Number of events, the first line of both text files is checked and made sure that it is consistent with the number of events below, as well as between the two text files.
- There may be inconsistencies with the ordering of the events, but since we use map to store events, this ordering problem does not affect the process as long as there are matching event names in both files.
- Negative number of events in the first line of the text file is also checked to prevent errors.
- Problems with duplicate events in each txt file are also checked and reported.
- In our data structures, both values from Events.txt and Stats.txt are stored in EventType. This resolves issues regarding inconsistency because when the Events.txt file is first read, it establishes the events for analysis. Therefore, when the reading of Stats.txt variables takes place, if an event is not already in the map, the program detects inconsistency and terminates. Also if the number of events input is not the same as the already existing map size, it also throws an error message and terminates. Also to disable double input of the same event from Stats.txt, we have a Boolean variable which is set to false in readEventFile function and as stats are read

into events in readStats function, it is set to true. The readStats function checks every line if the Boolean is false before setting mean and standard deviation.

map <string, pair<EventType, bool> > eventTypeMap;

- The value mean plus standard deviation should be less than the max value allowed. The value mean minus standard deviation should be greater than the min value allowed. We do this to ensure that there is an appropriate range to generate the event values. Otherwise, the min and max may just be too close to the value and we cannot get the distribution that we want.
- If there are empty lines or incorrect number of fields separated by colons in between the strings in Events.txt or Stats.txt, the program skips the empty line and continues reading. But if the line is not empty and not the correct format, it will report an error.
- All potential inconsistencies that we are aware of are checked in our program.

Activity Engine & the Logs

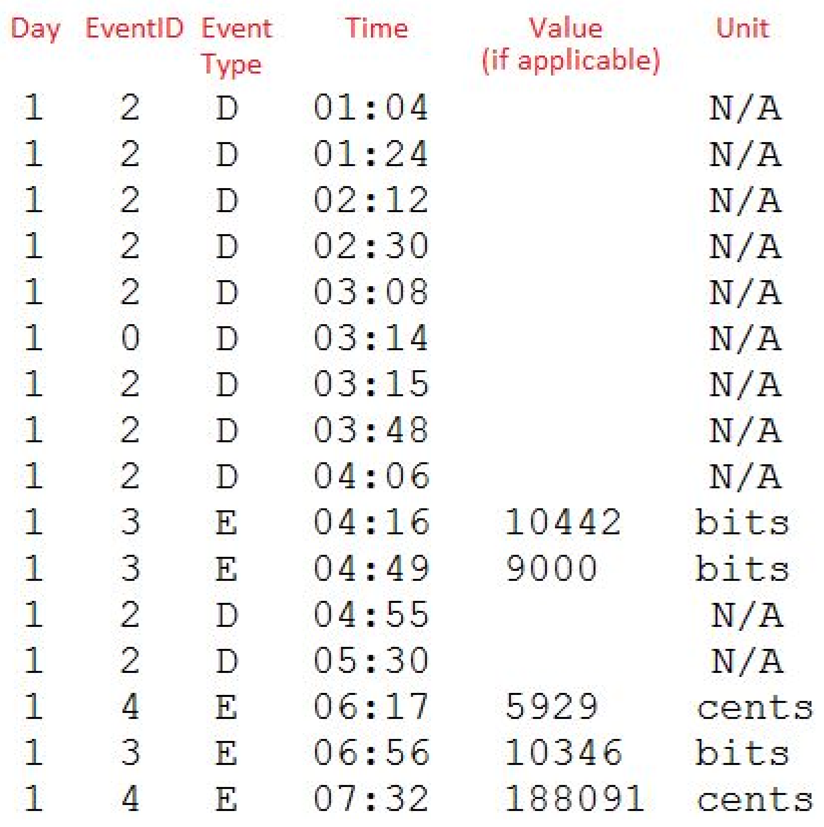
1. The process used to generate events approximately consistent with the particular distribution. This is likely to differ between discrete, continuous, and E events. If you can handle some but not all of those note this.

- We used the std::normal_distribution utility of C++ to generate the daily totals by providing the mean and the standard deviation.
- For each day, we randomly generate an integer for each event, which is the total number of times that event occurs in that day. The daily total is then divided by the number of occurrences to produce the average for each instance of that event. Then we narrow down the range that the event value can be generated to ensure that it is between the min and max for that particular event, and that the value of it will not cause the values of other succeeding instances to exceed the valid range.
  - Lower bound: max (value of the previous instance, the max values possible for the number of remaining instances to be generated so that they can even out the current value).
  - Upper bound: the new average value for the remaining instances to be generated.

- The final value of an event instance is calculated by subtracting the daily total by the total produced so far.
- Some factors that potentially contribute to the mismatch of statistics:
  - We do not have control over which value gets generated by C++ std::normal_distribution, the value of the daily totals may not produce the exact statistics as in the Stats.txt file, especially when the number of days of the simulation is small.
  - For discrete events D and E, the values are rounded to the nearest integer.
  - Values that are not between the minimum and maximum of an event are also discarded. Therefore, we may lose some important values that make the statistics more accurate.
- Additionally, because we also use the random function of C++ to generate the value of each event instance, and the range of the next instance to be generated gets narrowed down, eventually the range may just be too small so the last few values may cluster together, which makes it appear less random.
- Overall, we are able to handle all types of events and produce statistics quite consistent with the one provided in the file, and it becomes more accurate when we increase the number of days across which we generate the events.

2. The name and format of the log file, with justification for the format. You will need to be able to read the log entries for subsequent parts of the program. The log file needs to be human readable.

Baseline.log



Day	EventID	Event Type	Time	Value (if applicable)	Unit
1	2	D	01:04		N/A
1	2	D	01:24		N/A
1	2	D	02:12		N/A
1	2	D	02:30		N/A
1	2	D	03:08		N/A
1	0	D	03:14		N/A
1	2	D	03:15		N/A
1	2	D	03:48		N/A
1	2	D	04:06		N/A
1	3	E	04:16	10442	bits
1	3	E	04:49	9000	bits
1	2	D	04:55		N/A
1	2	D	05:30		N/A
1	4	E	06:17	5929	cents
1	3	E	06:56	10346	bits
1	4	E	07:32	188091	cents

***Disclaimer: This picture above has been slightly altered in comparison to the actual log file for the purpose of demonstration**

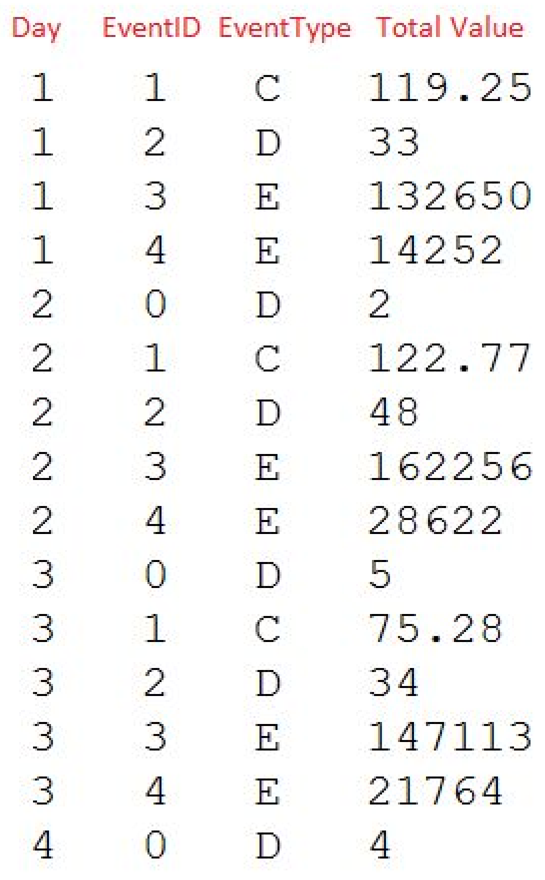
Day No: Day identifier
EventID: Instead of event name, event is defined by their number. If login is the first event, 0 would be used to identify that event.
EventType: Type of event CDE.
Time: The time an event instance occur, in chronological order.
Value: Value associated with each event instance (except for D events).
Unit: The unit of the value.

Justification:
These are all the basic details required to identify the events belong to each day and produce the daily totals for each event. The time sorted in chronological order so as to make it look like a real log file. We add the unit to give more information about the type of events. It will not be read in and used by the Analysis Engine.

Analysis Engine

1. Specify the file containing the daily totals for the events, and explain the format of it.

dayStat.txt



Day	EventID	EventType	Total Value
1	1	C	119.25
1	2	D	33
1	3	E	132650
1	4	E	14252
2	0	D	2
2	1	C	122.77
2	2	D	48
2	3	E	162256
2	4	E	28622
3	0	D	5
3	1	C	75.28
3	2	D	34
3	3	E	147113
3	4	E	21764
4	0	D	4

Day: Day identifier
EventID: Instead of event name, event is defined by their number. If login is the first event, 0 would be used to identify that event.
EventType: Type of event CDE.
Total Value: Total value for the day for that event.
Justification: The four column of data are the minimum information required to calculate the mean and standard of each event number. Day and EventID together make up the identifier for the DayStats object.

2. Possible anomalies in reading the logs.

- In the cases of where the log file may not exist because the user deleted the log file mid program execution, the program failing to open log file will report an error.
- If the baseline log file is somehow corrupted and some events are missing, then we cannot calculate the mean and standard deviation for an event that does not exist

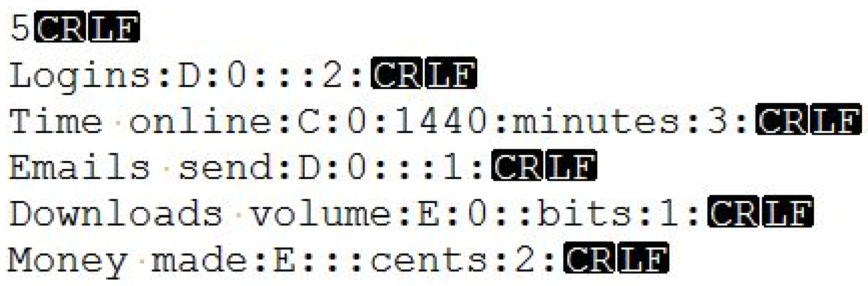
and those events will not be in the baseline statistics. We did not attempt to handle this problem as our program conducts the event generation, analysis, and alert all in one run. However, if our program belongs to a real-world system and it allows attacker to inject malicious codes, then it is compulsory that it should be resolved.

- The number of data per line is also checked to make sure that it is not more than 6 due to the formating of the log file. We could additionally check that event type C or E does not have an empty value field and event type D have an empty value field, but we did not implemented it yet.
- The program does not check if the event number is consistent with the Events.txt file as the baseline is generated from the Events.txt and Stats.txt file.
- The program does however checks to make sure that the read data for event value is double for C and int for E, and the day and event numbers are integer.
- The time is not a variable that is used with our calculations, therefore we do not check to make sure that the time is in a correct format.
- We do not check the units in the log against the units in the Events.txt in Analysis phase.

3. Possible anomalies in determining the statistics

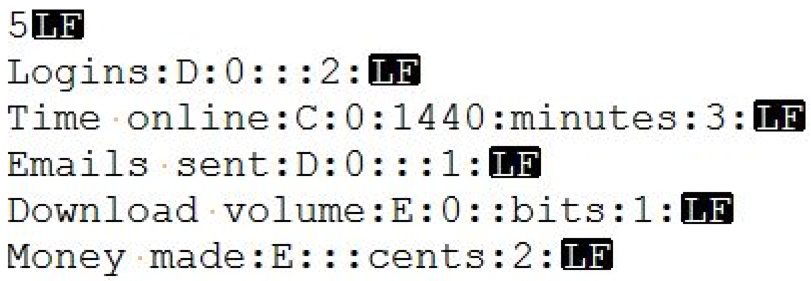
- The values in the log file are rounded to two decimal places, so it is possible that the baseline statistics is even further away from the statistics we actually generated.
- If there are events which somehow do not exist in Baseline.log, the anomaly counter will be incorrect as it is missing events that contribute to it.
- The program does not take into account if only one event instance is generated across the days resulting in a faulty mean and standard deviation calculation from the baseline. The activity engine is designed so that there will always be multiple instances of an event across the days; therefore, unless the baseline file is corrupted, this scenario will not happen. However, if the number of days entered is only 1, this is inevitable.
- We do not allow the days to be generated to be less than or equal to 0. If 0 is allowed, the anomaly counter will be 0 and it does not mean anything.

Other problems: Depending on the Operating System, if the carriage return (CR) is present in the file the program reports an error.



```
5CRLF
Logins:D:0:::2:CRLF
Time online:C:0:1440:minutes:3:CRLF
Emails send:D:0:::1:CRLF
Downloads volume:E:0::bits:1:CRLF
Money made:E:::cents:2:CRLF
```

This does not work



```
5LF
Logins:D:0:::2:LF
Time online:C:0:1440:minutes:3:LF
Emails sent:D:0:::1:LF
Download volume:E:0::bits:1:LF
Money made:E:::cents:2:LF
```

This works