Attendance Project: Converting Zoom Participant Data for Gradescope

Tong Yi

Contents

1 Copyright Claim

1. This is copyrighted materials; you are not allowed to upload to the Internet.

2. Our project is different from similar products in Internet.

   (a) Ask help only from teaching staff of this course.

(b) Use solutions from artificial intelligence like ChatGPT or online tutoring websites like, but not limited to, chegg.com, violates academic integrity and is not allowed.

2 Goals

In this project, we will learn how to read and analyze a Comma-Separated-Values (CSV) file, process a string, formatted outputs, and generate a pdf file.

We work with Tab-Separated-Values (TSV) files in Lab 3. The difference between CSV and TSV files is the separator. In CSV, the separator is comma symbol, while in TSV, the separator is a tab. If a column data contains spaces, we should use CSV file.

3 Structure of the Project

1. Create directory `attendance` to hold codes of the project **if** you have not done so. Said differently, you only need to run the following command once.

   ```
   mkdir attendance
   ```

2. Move to the above directory.

   ```
   cd attendance
   ```

4 Task A: Add a Series of Integers and Output Their Sum

Enter a series of integers from console, then end the input by pressing

- ctrl and d at the same time in Mac / Linux or
- ctrl and z at the same time in Windows
- You need to press return key after the ctrl + d or ctrl + z.

Here is a sample input/output. After entering 6, press ctrl + d, then a return key in Mac / Linux. If you are use windows, use ctrl and z.

```
Enter a series of ints (use ctrl + d in Mac and ctrl + z in Windows to end
    input)): 1 2 3 6

sum = 12
```

1. Create a file named **add.cpp**.
2. In the program, prompt “Enter a series of ints: ".
3. Calculate the sum of these numbers.
4. Print the result.

Submit `add.cpp` to gradescope. Note that the grading script generated random int numbers to test. As a result, your output will be different in each running.

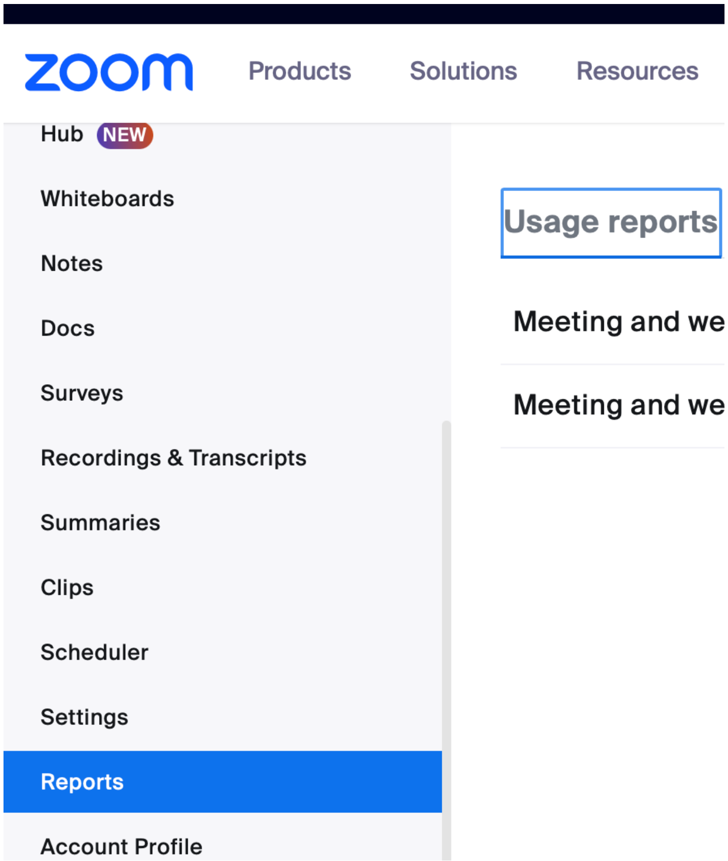
5 Task B: Calculate Percentage of Participants Exceeding Attendance Threshold

5.1 Optional: Download Zoom Attendance Report

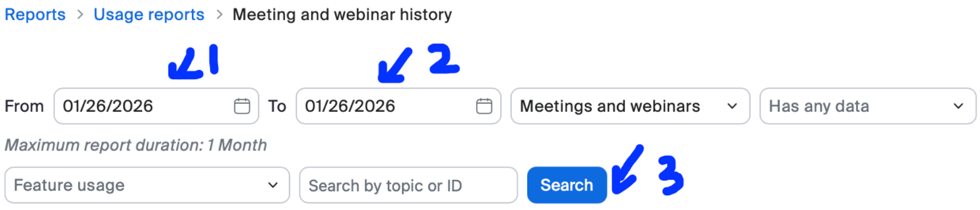
A zoom attendance report has four columns when you do the following

1. Sign in `zoom.us`.

2. Scroll down the left bar, find out "Reports".



3. Search the meeting in a given time range.

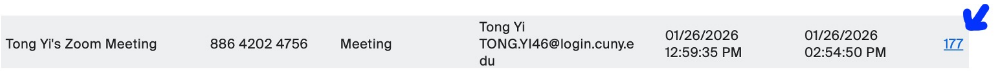


From 01/26/2026 To 01/26/2026 Meetings and webinars Has any data

Maximum report duration: 1 Month

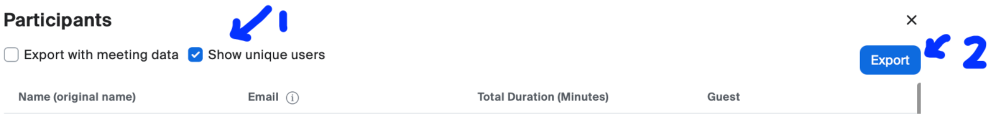
Feature usage Search by topic or ID

4. Click the blue number – which indicates the total count of participants who joined that specific meeting or webinar – to open a specific report.



Tong Yi's Zoom Meeting	886 4202 4756	Meeting	Tong Yi TONG.YI46@login.cuny.edu	01/26/2026 12:59:35 PM	01/26/2026 02:54:50 PM	177

5. Check the box "**Show Unique Users**". Then click "**Export**" button.



Participants

☐ Export with meeting data ☑ Show unique users

Name (original name) | Email | Total Duration (Minutes) | Guest

6. A CSV file with the following headers would be downloaded.

   `Name (original name),Email,Total duration (minutes),Guest`

   In the following participants file, Alex Smith attended 65 minutes of the meeting, Jordan Jones attended 25 minutes, and so on.

   If a Zoom attendance report shows empty email fields, it is likely due to privacy settings introduced in March 2022. To protect "Personally Identifiable Information" (PII), Zoom masks emails for participants who join as guests unless specific criteria are met.

```
Name (original name),Email,Total duration (minutes),Guest
Alex Smith,,65,yes
Jordan Jones,,25,yes
... //omit other data
```

5.2 Steps of Task B

1. Create a sample CSV file as follows. Call it `participants.csv`.

```
Name (original name),Email,Total duration (minutes),Guest
Alex Smith,,65,yes
Jordan Jones,,25,yes
Tylor Brown,,49,yes
Morgan Wilson,,24,yes
Casey Taylor,,81,yes
John Ou,,65,yes
Matt Johnson,,79,yes
Liam Williams,,38,yes
Olivia Garcia,,47,yes
```

Listing 1: participants.csv file

   (a) The first line is column header.

   (b) Each record, represented by one row in the table, records name, email, total duration (minutes) and whether this participant is a guest or not.

   (c) Give the above data file a meaningful name, say `participants.csv`. Warning: if you test your source code in onlinegdb, after you upload source code to the server, you need to upload a data file as well. However, you may need to rename the data file as `participants.txt`, since onlinegdb does not recognize a file whose suffix is `csv`.

2. Name the source code as `percentage.cpp`. You need to implements codes for the following steps.

   (a) Enter a file name for the data.

(b) Enter the duration in minutes (an integer).

(c) Calculate the percentage of participants whose total duration is at least that of the given duration.

(d) Print out the percentage.

3. Here is a sample input/output for the above data, where "Enter a csv file:" is a prompt, and spending.csv (with return key) is input from a user, and output is `sum = 12408.77`.

```
Enter a zoom-style csv file, containing Name (original name),Email,Total duration
    (minutes),Guest: participants.csv
Enter the number of minutes to attend: 75
percentage of students attend at least 75 minutes is 22.22%
```

4. To correctly read a file's contents, we must first understand its structure. This includes identifying introductory information, such as column headers, and knowing the meaning of each column. This process is similar to reading console input, but the data source is a file rather than user input from console.

5.3 Related: Read a Tab-Separated-Values (TSV) file

The class `ifstream` is used to read plain text files, `ofstream` is used to write plain text files, and `fstream` is generally used for binary files such as audio or video. In this course, we will focus only on `ifstream` and `ofstream`.

When a computer reads a file, it behaves much like a human reader: it processes the file from top to bottom, and within each line, from left to right.

To read the contents of a TSV file (tab-separated values), we can use the extraction operator `>>` (also called the pull operator). This operator reads input until it encounters the next whitespace character, which could be a space, a tab, or a newline.

Step 1 Before reading a file, it is important to understand its structure:

How many non-data lines (such as headers or comments) appear before the actual data.

The order of the columns: for example, what the first column represents, what the second column represents, and so on.

The meaning of each column (e.g., date, description, category, amount).

The data type of each column (e.g., integer, floating-point, string).

Step 2 Open a plain text file to read by instantiating an ifstream object.

```cpp
std::ifstream fin(fileName); //fileName is a string variable.
```

Step 3 Skip all explanation lines.

Step 4 Test the file can be read or not using fail method of fstream class. You may not be able to read the file since the file might be missing or corrupt, or lack of permission.

```cpp
if (fin.fail()) {
  std::cerr << "The file cannot be opened." << std::endl;
  exit(1); //leave the program
}
```

Step 5 Use `fin >> variableName;` to read from the file associated with fin. Make sure the data value matches the type of `variableName`.

Step 6 After reading, use `fin.close();` to close the file associated with fin.

5.3.1 Example of Reading a Tab-Separated-Values (TSV) file

Suppose we have a TSV file that records each student's name, midterm score, final score, and bonus points. To calculate the total score, assign a weight of 30% to the midterm and 70% to the final, then add the bonus points.

1. Here are the contents of a file named `scores_s26.tsv`. The first line are column header. Starting from the second line, there are data. For example, the student's name is Ann. Her midterm grade is 77, her final grade is 78, and bonus is 1.2.

   Use text editor tools like Visual Studio Code (Mac and Windows) or TextEdit (Mac) or notepad++ (Windows) to create the following file, name it `scores_s26.tsv`. Put in the same folder of your C++ source code.

```
Name Midterm Final Bonus
Ann 77  78  1.2
Bob 88  89  2
Charles 99 100 0
```

2. First open the file using ifstream (input file stream) from namespace std. Need to import fstream library to use ifstream class.

   The name of ifstream object `fin` is to distinct from standard input stream object `cin`. You can rename `fin` with any meaningful name.

   Note that we do not need to instantiate `cin`, it always associated with keyboard input. The name `cin` cannot be changed.

   However, `ifstream` object `fin` needs to be instantiated as follows. And the name `fin` can be renamed as other meaningful names, for example, `inp`.

```
//instantiate an ifstream object fin that reads from scores_s26.tsv.
std::ifstream fin("scores_s26.tsv"); //ifstream means input file stream
```

   The following statement instantiates a string object called `greeting`, whose initial value is `"hello, world"`.

```
std::string greeting("hello, world");
```

3. Test whether the file can be opened or not. Sometimes the file can be corrupted or you do not have authority to read it. If the file cannot be opened, we print out an error message, then quit the program by calling exit function.

```
if (fin.fail()) {
  //std::cerr is the destination of error messages.
  //By default, it is the console.
  std::cerr << "file cannot be open to read." << std::endl;

  //Stop running the program.
  //Exit to the operating system with error code 1.
  exit(1);
}
```

4. If the file can be opened properly, skip the first line "Name Midterm Final Bonus", since it does not contain data.

```
std::string line;
getline(fin, line);
//no further process of line is needed since line is column header,
//not the data we need in this program.
```

5. Every row of data starts with name (a string), midterm grade (an integer), and final grade (an integer), and a bonus (a double type number). Since each column is separated by a space, we can use >> operator.

   Here is the pesudocode of data processing.

```
as long as there is a name  //each data line starts with a name
begin
  read midterm
  read final
  read bonus

  calculate and print the total, which is 0.3 * midterm + 0.7 * final + bonus
end
```

   When writing the actual code, need to note that `final` is a keyword in C++ and cannot be used as a variable name. Here is an implementation.

   Note that we use `fin >> variableName` since the source of data comes from fin, not `cin`.

```
std::string name;
int midterm;
int finalGrade;
double bonus;
double total;
while (fin >> name) {
  fin >> midterm;
  fin >> finalGrade;
  fin >> bonus;

  total = 0.3 * midterm + 0.7 * finalGrade + bonus;

  std::cout << name << " " << total << std::endl;
}
```

6. Close the ifstream object fin after finishing reading the corresponding file.

```
fin.close();
```

7. Here is a **link** to work on a TSV file.

```
//code link: https://onlinegdb.com/cJjWnhsLTL
//Purpose:
//1. Read student data from a TSV file (Name, Midterm, Final, and Bonus).
```

```cpp
//2. Compute the final grade: (30% Midterm) + (70% Final) + Bonus.
//3. Output the name and calculated total for each student.

//contents of scores_s26.tsv

//Name midterm final bonus
//Ann 77  78  1
//Bob 88  89  2
//Charles 99 100 0.5

//Sample output:

//Ann 78.7
//Bob 90.7
//Charles 100.2

#include <iostream>
#include <string>
#include <fstream>

int main() {
  //create an ifstream object fin that will read from scores_s26.tsv.
  std::ifstream fin("scores_s26.tsv");

  if (fin.fail()) {
    //std::cerr is the destination of error messages.
    //By default, it is the console.
    std::cerr << "file cannot be open to read." << std::endl;

    //Stop running the program.
    //Exit to the operating system with error code 1.
    exit(1);
  }

  std::string line;
  getline(fin, line);
  //no further process of line is needed since line is column header,
  //not the data we need in this program.

  std::string name;
  int midterm;
  int finalGrade;
  double bonus;
  double total;
  while (fin >> name) {
    fin >> midterm;
    fin >> finalGrade;
    fin >> bonus;

    total = 0.3 * midterm + 0.7 * finalGrade + bonus;

    std::cout << name << " " << total << std::endl;
  }

  //Close the file after finishing processing it.
  fin.close();

  return 0;
}
```

5.4 Read a CSV file

The steps to read a CSV file is similar to those of reading a TSV file, they only differ in Step 5.

Step 5.1 For each row of data in a csv file, every item **except** the last one is ended by ’,’. Use `getline(fin, variableName, ’,’);` to read a value before ’,’ and put that value in `variableName`, a string type variable.

Step 5.2 The last item in a row of data is ended by a new line character. Use `getline(fin, variableName);` to read the value ofthe last item to appropriate `variableName`, a string variable.

Step 5.3 Unlike extraction operator `>>`, which can convert the read value into a corresponding type not limited to string, function `getline` reads a value to a string variable. To convert a string to an int, use function `stoi`, which takes in a string and converts to an integer. For example, `stoi("15")` returns 15.

Similarly, function `stod` converts a string to a corresponding double number. For example, `stod("123.56")` returns a double number 123.56.

5.4.1 Example of Reading a Comma-Separated-Values (CSV) file

Suppose we have a CSV file recording name, midterm, final and bonus of students. Calculate the total where the weight of midterm 30% and the of final is 70%, then plus bonus.

1. Here are the contents of a file named `scores_s26.csv`. The first line are column header. Starting from the second line, there are data. For example, the midterm grade of Ann is 77, her final grade is 78, and bonus is 1.2.

   Use text editor tools like Visual Studio Code (Mac and Windows) or TextEdit (Mac) or notepad++ (Windows) to create the following file, name it `scores_s26.csv`. Put in the same folder of your C++ source code.

   Read the contents of the following CSV file, a name may contain spaces, since the delimiter (separator) of column data is a comma, not a space.

```
Name,Midterm,Final,Bonus
Ann Johnson,77,78,1.2
Bob Smith,88,89,2
Charles Chan,99,100,0.5
```

2. First open the file using ifstream (input file stream). Need to import fstream library to use ifstream class.

   The name of ifstream object `fin` is to distinct from standard input stream object `cin`. You can rename `fin` with any meaningful name.

```
//create an ifstream object fin that will read from scores_s26.csv.
//if you test the code in onlinegdb, rename the file as scores_s26.txt.
std::ifstream fin("scores_s26.csv");
```

3. Test whether the file can be opened or not. Sometimes the file can be corrupted or you do not have authority to read it. If the file cannot be opened, we print out an error message, then quit the program by calling exit function.

```
if (fin.fail()) {
  //std::cerr is the destination of error messages.
  //By default, it is the console.
  std::cerr << "file cannot be open to read." << std::endl;

  //Stop running the program.
  //Exit to the operating system with error code 1.
  exit(1);
}
```

4. If the file can be opened properly, skip the first line “Name,Midterm,Final,Bonus”, since it does not contain data.

```
std::string line;
getline(fin, line);
//no further process of line is needed since line is column header,
//not the data we need in this program.
```

5. Every row of data starts with name (a string), midterm grade (an integer), and final grade (an integer), and a bonus (a double type number). Since each column is separated by a space, we can use >> operator.

   Here is the pesudocode of data processing.

```
as long as there is a name  //the line has data, starting with column name
begin
  read a string and convert it to midterm
  read the next string and convert it to final
  read the next string and convert it to bonus

  calculate and print the total, which is 0.3 * midterm + 0.7 * final + bonus
end
```

   When writing the actual code, need to note that `final` is a keyword in C++ and cannot be used as a variable name. Here is an implementation.

   Note that we use `getline(fin, variableName, ’,’);` since the source of data comes from fin, and stop at the next ’,’.

```
std::string name;
```

```
std::string midtermStr;
int midterm;
std::string finalGradeStr;
int finalGrade;
std::string bonusStr;
double bonus;
double total;

//cannot use fin >> name; to replace getline(fin, name, ’,’)
//extraction operator >> stops before a white space symbol,
//which can be a space, a tab, or a new line character
while (getline(fin, name, ’,’)) {
  //cannot replace getline(fin, midtermStr, ’,’); with
  //fin >> midtermStr;
  //extraction operator >> reads all the contents until a white space
  getline(fin, midtermStr, ’,’);
  midterm = stoi(midtermStr); //midterm is an int

  getline(fin, finalGradeStr, ’,’);
  finalGrade = stoi(finalGradeStr);

  //getline(fin, bonusStr, ’,’); //WRONG, bonus is the last item in the row,
  //the delimiter after bonus is a new line character, not a ,
  getline(fin, bonusStr); //this version of getline reads bonusStr before the first new line character

  //use stod function to convert a string to a double number
  bonus = stod(bonusStr);

  total = 0.3 * midterm + 0.7 * finalGrade + bonus;

  std::cout << name << " " << total << std::endl;
}
```

6. Close the ifstream object fin after finishing reading the corresponding file.

```
fin.close();
```

7. Here is a **link** to work on a CSV file.

```
//code link: https://onlinegdb.com/pruMdoTA9
//Purpose:
//1. Read student data from a CSV file (Name, Midterm, Final, and Bonus).
//2. Compute the final grade: (30% Midterm) + (70% Final) + Bonus.
//3. Output the name and calculated total for each student.

//contents of scores_s26.csv

//Name,Midterm,Final,Bonus
```

```
//Ann Johnson,77,78,1.2
//Bob Smith,88,89,2
//Charles Chan,99,100,0.5

//Sample output:

//Ann Johnson 78.9
//Bob Smith 90.7
//Charles Chan 100.2

#include <iostream>
#include <string>
#include <fstream>

int main() {
  //create an ifstream object fin that will read from scores_s26.csv.
  //if you test the code in onlinegdb, rename the file as scores_s26.txt.
  std::ifstream fin("scores_s26.csv");

  if (fin.fail()) {
     //std::cerr is the destination of error messages.
     //By default, it is the console.
     std::cerr << "file cannot be open to read." << std::endl;

     //Stop running the program.
     //Exit to the operating system with error code 1.
     exit(1);
  }

  std::string line;
  getline(fin, line);
  //no further process of line is needed since line is column header,
  //not the data we need in this program.

  std::string name;
  std::string midtermStr;
  int midterm;
  std::string finalGradeStr;
  int finalGrade;
  std::string bonusStr;
  double bonus;
  double total;

  //cannot use fin >> name; to replace getline(fin, name, ’,’)
  //extraction operator >> stops before a white space symbol,
  //which can be a space, a tab, or a new line character
  while (getline(fin, name, ’,’)) {
    //cannot replace getline(fin, midtermStr, ','); with
    //fin >> midtermStr;
    //extraction operator >> reads all the contents until a white space
    getline(fin, midtermStr, ',');
    midterm = stoi(midtermStr); //midterm is an int

    getline(fin, finalGradeStr, ',');
    finalGrade = stoi(finalGradeStr);

    //getline(fin, bonusStr, ','); //WRONG, bonus is the last item in the row,
    //the delimiter after bonus is a new line character, not a ,
    getline(fin, bonusStr); //this version of getline reads bonusStr before the first new line character
    bonus = stod(bonusStr); //bonus is a double,
      //use stod function to convert a string to double number

    total = 0.3 * midterm + 0.7 * finalGrade + bonus;

    std::cout << name << " " << total << std::endl;
  }

  fin.close();

  return 0;
}
```

6 Task C: Generate a pdf File with Content “John Doe"

In this task, do the following:

1. **File Naming:** Name your source code file `generate_john_doe.cpp`.
2. **User Input:** Prompt for or define an output filename ending with the .pdf suffix (e.g., `john_doe.pdf`).
3. **Stream Initialization:** Create an `ofstream` (output file stream) object to write data into the specified PDF file.
4. **Error Handling:** Implement a check to verify if the file was opened successfully before proceeding.
5. **Data Output:** Write the required content into the file using the stream object.

   Important: in Lines 40 - 45, there is a trailing space after the last visible character `f` or `n`.

   Copying from the list below can be difficult; instead, you may download the file directly: `john_doe.txt`. Once downloaded, you can proceed with the content.

```
1 %PDF-1.4
2 1␣0␣obj
3 <<␣/Type␣/Catalog␣/Pages␣2␣0␣R␣>>
4 endobj
```

```
2␣0␣obj
<<␣/Type␣/Pages␣/Kids␣[3␣0␣R]␣/Count␣1␣>>
endobj
3␣0␣obj
<<
␣␣/Type␣/Page
␣␣/Parent␣2␣0␣R
␣␣/MediaBox␣[0␣0␣612␣792]
␣␣/Contents␣4␣0␣R
␣␣/Resources␣<<
␣␣␣␣/Font␣<<
␣␣␣␣␣␣/F1␣5␣0␣R
␣␣␣␣>>
␣␣>>
>>
endobj
4␣0␣obj
<<␣/Length␣44␣>>
stream
BT
/F1␣24␣Tf
100␣692␣Td
(John␣Doe)␣Tj
ET
endstream
endobj
5␣0␣obj
<<
␣␣/Type␣/Font
␣␣/Subtype␣/Type1
␣␣/BaseFont␣/Helvetica-Bold
>>
endobj
xref
0␣6
0000000000␣65535␣f␣
0000000009␣00000␣n␣
0000000058␣00000␣n␣
0000000115␣00000␣n␣
0000000267␣00000␣n␣
0000000357␣00000␣n␣
trailer
<<␣/Size␣6␣/Root␣1␣0␣R␣>>
startxref
438
%%EOF
```

6. **Stream Closure:** Close the `ofstream` object once all contents have been successfully written to the file.

7. Run your C++ program. It the codes work, the specified PDF file will be generated.

   Open this file using any PDF viewer, such as Adobe Acrobat or acroread, the file will contain the text "John Doe" (without quotes).

7 Task D: Generate All pdfs for Names in Participants CSV File

7.1 Motivation

Converting Zoom participant data to Gradescope isn't always straightforward. A common manual method involves using `VLOOKUP` in Excel or Google Sheets to match Zoom emails against the Gradescope roster. However, this often fails because Zoom emails may be missing or may not match the official addresses stored in Gradescope. While third-party software exists to bridge this gap, it can pose significant privacy risks.

Our Recommended Workflow:

To ensure accuracy and data privacy, we use the following automated approach:

- **Export Data:** Download the Zoom participants CSV, which includes name, email, duration, and guest status. (Note: Email fields may be blank depending on your privacy settings).

- **Generate PDF Proofs:** Programmatically generate an individual PDF file for every participant listed in the CSV.

- **Create Gradescope Assignment:** Set up an attendance column/assignment in Gradescope with a "Name" field.

- **Bulk Upload:** Upload the generated PDFs to Gradescope. The system will auto-match most students based on the names on the PDFs.

- **Manual Review:** Perform a quick manual check for any "unmatched" names to resolve discrepancies.

The Programmer's Rule: If a task must be performed more than once, automate it. Automating this workflow not only saves hours of manual data entry throughout the semester but also ensures students receive timely attendance feedback.

The core strategy is to generate an individual PDF for every participant. In the following subsection, we will demonstrate how to programmatically modify a template PDF to replace a placeholder like "John Doe" with any name from your list.

7.2 Modidy a Template PDF to Replace "John Doe" with Any Name

7.2.1 A Name with Fewer Characters Than "John Doe"

Suppose we need to print a name shorter than "John Doe", say, "Ann Yi." To achieve this, we must modify the PDF's internal parameters.

"John Doe" consists of 8 characters (including the space), whereas "Ann Yi" consists of 6. Because "Ann Yi" is 2 characters shorter, the specific length parameter within the PDF structure must be decreased by 2.

The following comparison illustrates the differences in the PDF source code for both names. In addition to updating the length parameter, the display text string itself must be changed from “John Doe" to “Ann Yi."

John Doe	**Ann Yi**
`<< /Length 44 >>`	`<< /Length 42 >>`
`(John Doe) Tj`	`(Ann Yi) Tj`
`0000000357 00000n`	`0000000355 00000n`
438 (the line after startxref)	436 (the line after starxref)

The following presents a side-by-side comparison of the complete contents of the PDF files for “John Doe" and “Ann Yi", with differences highlighted.

```
%PDF-1.4
1 0 obj
<< /Type /Catalog /Pages 2 0 R >>
endobj
2 0 obj
<< /Type /Pages /Kids [3 0 R] /Count 1 >>
endobj
3 0 obj
<<
  /Type /Page
  /Parent 2 0 R
  /MediaBox [0 0 612 792]
  /Contents 4 0 R
  /Resources <<
    /Font <<
      /F1 5 0 R
    >>
  >>
>>
endobj
4 0 obj
<< /Length 44 >>
stream
BT
/F1 24 Tf
100 692 Td
(John Doe) Tj
ET
endstream
endobj
5 0 obj
<<
  /Type /Font
  /Subtype /Type1
  /BaseFont /Helvetica-Bold
>>
endobj
xref
0 6
0000000000 65535 f␣
0000000009 00000 n␣
0000000058 00000 n␣
0000000115 00000 n␣
0000000267 00000 n␣
0000000357 00000 n␣
trailer
<< /Size 6 /Root 1 0 R >>
startxref
438
%%EOF
```

```
%PDF-1.4
1 0 obj
<< /Type /Catalog /Pages 2 0 R >>
endobj
2 0 obj
<< /Type /Pages /Kids [3 0 R] /Count 1 >>
endobj
3 0 obj
<<
  /Type /Page
  /Parent 2 0 R
  /MediaBox [0 0 612 792]
  /Contents 4 0 R
  /Resources <<
    /Font <<
      /F1 5 0 R
    >>
  >>
>>
endobj
4 0 obj
<< /Length 42 >>
stream
BT
/F1 24 Tf
100 692 Td
(Ann Yi) Tj
ET
endstream
endobj
5 0 obj
<<
  /Type /Font
  /Subtype /Type1
  /BaseFont /Helvetica-Bold
>>
endobj
xref
0 6
0000000000 65535 f␣
0000000009 00000 n␣
0000000058 00000 n␣
0000000115 00000 n␣
0000000267 00000 n␣
0000000355 00000 n␣
trailer
<< /Size 6 /Root 1 0 R >>
startxref
436
%%EOF
```

7.2.2 A Name with More Characters Than “John Doe”

Suppose we need to print a name longer than “John Doe", say, “George Washington." To achieve this, we must modify the PDF’s internal parameters.

“John Doe" consists of 8 characters (including the space), whereas “George Washington" consists of 17. Because “George Washington" is 9 characters longer, the specific length parameter within the PDF structure must be increased by 9.

The following comparison illustrates the differences in the PDF source code for both names. In addition to updating the length parameter, the display text string itself must be changed from “John Doe" to “George Washington."

John Doe	**George Washington**
`<< /Length 44 >>`	`<< /Length 53 >>`
`(John Doe) Tj`	`(George Washington) Tj`
`0000000357 00000n`	`0000000366 00000n`
438 (the line after startxref)	447 (the line after starxref)

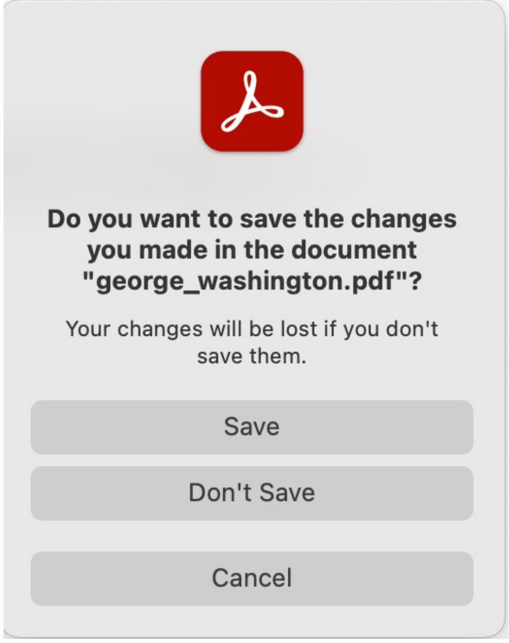
The following presents a side-by-side comparison of the complete contents of the PDF files for “John Doe" and “George Washington", with differences highlighted.

```
%PDF-1.4
1 0 obj
<< /Type /Catalog /Pages 2 0 R >>
endobj
2 0 obj
<< /Type /Pages /Kids [3 0 R] /Count 1 >>
endobj
3 0 obj
<<
  /Type /Page
  /Parent 2 0 R
  /MediaBox [0 0 612 792]
  /Contents 4 0 R
  /Resources <<
    /Font <<
      /F1 5 0 R
    >>
  >>
>>
endobj
4 0 obj
<< /Length 44 >>
stream
BT
/F1 24 Tf
100 692 Td
(John Doe) Tj
ET
endstream
endobj
5 0 obj
<<
  /Type /Font
  /Subtype /Type1
  /BaseFont /Helvetica-Bold
>>
endobj
xref
0 6
0000000000 65535 f␣
0000000009 00000 n␣
0000000058 00000 n␣
0000000115 00000 n␣
0000000267 00000 n␣
0000000357 00000 n␣
trailer
<< /Size 6 /Root 1 0 R >>
startxref
438
%%EOF
```

```
%PDF-1.4
1 0 obj
<< /Type /Catalog /Pages 2 0 R >>
endobj
2 0 obj
<< /Type /Pages /Kids [3 0 R] /Count 1 >>
endobj
3 0 obj
<<
  /Type /Page
  /Parent 2 0 R
  /MediaBox [0 0 612 792]
  /Contents 4 0 R
  /Resources <<
    /Font <<
      /F1 5 0 R
    >>
  >>
>>
endobj
4 0 obj
<< /Length 53 >>
stream
BT
/F1 24 Tf
100 692 Td
(George Washington) Tj
ET
endstream
endobj
5 0 obj
<<
  /Type /Font
  /Subtype /Type1
  /BaseFont /Helvetica-Bold
>>
endobj
xref
0 6
0000000000 65535 f␣
0000000009 00000 n␣
0000000058 00000 n␣
0000000115 00000 n␣
0000000267 00000 n␣
0000000366 00000 n␣
trailer
<< /Size 6 /Root 1 0 R >>
startxref
447
%%EOF
```

7.2.3 Prompt to "Save Changes" After Opening Generated PDF in Adobe Acrobat

If you open the generated PDF in Adobe Acrobat or Acrobat Reader and a "Save Changes" prompt appears, as shown below, the file structure is likely incorrect.



Do you want to save the changes you made in the document "george_washington.pdf"?

Your changes will be lost if you don't save them.

This usually occurs due to one of the following errors:

- The parameter after `\Length` is not correct.
- The string to display in `(string_to_display) Tj` is either too long or too short.
- Miss a space in the end of Lines 40-45 after letter 'f' or 'n'.
- The first parameter in Line 45 is not correct.
- The number between `startxref` and `%%EOF` is not correct.

7.3 Steps of Task D

7.3.1 Directory Setup

Under the directory `attendance` created in Section 3, create a subdirectory `data`. This folder will store all generated PDF files to keep your workspace organized.

Navigate to your `attendance` directory (if you are not already there):

```
cd attendance
```

Create a subdirectory `data` under `attendance`. You only need to run the command **exactly once**.

```
mkdir data
```

Note that you should remain in the `attendance` directory. There is no need to move into the `data` subdirectory.

7.3.2 Develop the PDF Generation Program

Create a file named `generate_all_pdfs.cpp` and implement the following logic:

1. **User Input:** Prompt a user to enter the file name of a participants file in CSV format. This file should contain information of name, email (may be empty), duration (in minutes), and guest status.

2. **File Handling:**

   Open the above file, skip column header information.

   Extract name information. For each name in the format of “FirstName LastName”,

   (a) **Generate Output Filename**: Replace the space between the first and last name with an underscore (_) and append .pdf. For example, “Ann Yi" becomes `Ann_Yi.pdf`.

   Save the output file name in string variable `outPutFileName`.

   (b) **Initialize Output Stream:** Open the file for writing within the `data/` subdirectory, where `data` subdirectory was created in Section 7.3.1.

   The `ofstream` object is named `outF`, you can give it an appropriate name like `fout` if you like.

```
std::ofstream outF("data/" + outputFileName);
//instantiate ofstream object outF to write to outputFileName in data subdirectory.
```

   (c) **Write PDF Content:** Write the necessary PDF data to `outF`. Ensure the content is dynamically adjusted based on the length and value of the current name, as detailed in Section 7.2.

   (d) **Close Stream:** Close the `outF` object once the file is complete.

3. **Cleanup:** Close the input CSV file stream once all participants have been processed.

7.3.3 Testing Your Code Before Submission

Use the following `participants.csv` file for testing:

```
name,email,duration,guest
Alex Smith,,79,yes
John Ou,,91,yes
Jordan Jones,,88,yes
```

Your program should generate three PDF files in the `data` subdirectory, with each file displaying the corresponding name in the format of `FirstName LastName`:

```
Alex_Smith.pdf
John_Ou.pdf
Jordan_Jones.pdf
```

Once you have verified that your code works correctly on your local computer, upload `generate_all_pdfs.cpp` to the Gradescope server.