PSEUDOCODE

**1. JobOffer.h :**

Class JobOffer

Private :   
 Define variables id , name , batch , writtenteststatus ,interviewstatus , placed , package to store the values.

Public:

Constructor for JobOffer:   
 To store the values of the private variables.

Getter for JobOffer:

To get the values stored in the class object.

Setter for JobOffer:

To set the values stored in class object.

FUNCTION change\_status:

* Take Id as input.
* Change the status of interview to pass/fail
* Change the status of placement to yes/no
* Print Updated details.

END CLASS

*FUNCTION display\_joboffers:*

*Time Complexity-O(1), Space Complexity-O(m)*

Here m is for number of strings taking entries from job offers header.

-Store the mapped value of Id in the job offer pointer.

-Print each of the members of mapped values.

Take c as input value and call the change function.

**2. Company.h:**

Class company

Public:

Define variables cmpname , cmpaddress, package , hiredstudents, criteria , no\_of\_alumni , at\_college.

Constructor for company:

To store the values of the public variables

Getter for company:   
 To get the values stored in the class object.

Setter for company:

To set the values stored in the class object.  
END CLASS

**3. Internship.h :**

Class Internship

Public:

Define variables cmpname1 , cmpname2 , duration1 , duration2 , type1 , type2, sr\_no,stdname.

Constructor for Internship:

To store the values of the public variables.

Getter for Internship:

To get the values stored in the class object.

Setter for Internship:

To set the values stored in the class object.

END CLASS

*FUNCTION display\_internship:*

*Time Complexity-O(1), Space Complexity-O(1).*

-Store the mapped value of Id in the internship pointer.

-Print each of the members of mapped values .

**4. Placement.h:**

Class placement

Define avgpackage , highestpackage , rate\_of\_placement , appeared , placed.

Constructor for placement :   
 To store the variables defined above .

Getter for placement :

To get the values stored in class object.

Setter for placement:   
 To set the values stored in class object.

END CLASS

*FUNCTION max\_package:*

*Time Complexity-O(n), Space Complexity-O(1).*

Procedure: FindHighestPackage(j)

Input: j(map<string, JobOffer\*>) -map containing JobOffer.

1.Start

2.Set 'max' to 0.

3.For each element 'i' in 'j':

If 'i->second->getpackage()' is greater than 'max':

i. Set 'max' to 'i->second->getpackage()'.

4.Print max.

5.End

*FUNCTION mean\_package:*

*Time Complexity-O(n), Space Complexity-O(1).*

N as the iteration is over whole job offer file csv

For Loop: CalculateMeanPackage(j)

Input:- j (map<string, JobOffer\*>) - a map containing JobOffer

1.Start

2.Set ‘sum’ and ‘count’ to 0.4.

3.For each element ‘i’ in ‘j’:

a. Add ‘i->second->getpackage()’ to ‘sum’.

b. Increment 'count' by 1.

4.Calculate the mean by dividing 'sum' by 'count' and store it in 'mean'.

5.Print the value of the mean.

6.End

*FUNCTION median\_package:*

*Time Complexity-O(n) + O(nlogn)=O(n), Space Complexity-O(n).*

Here O(n) is for iteration and O(nlogn) for sort function  
Procedure: CalculateMedianPackage(j)

Input:- j (map<string, JobOffer\*>) - a map containing JobOffer

1.Start

2.Create an empty vector ‘v’ to store values.

3.For each element ‘i’ in ‘j’:

Append ‘i->second->getpackage()’ to the vector ‘v’.

4.Sort the vector v in ascending order.

5.Calculate the length l of the vector v

If ‘l’ is even:

Calculate median ‘d’ as average of two middle elements:

i. ‘v[(l/2)-1]’ and ‘v[(l/2) + 1 - 1]’.

Else, if ‘l’is odd:

Calculate median ‘d’ as middle element: v[((l+1)/2)-1]’.

6.Print Median.

7.End.

**5. Schedule.h:**

Public:  
Define variables id , stdname , cmpname , s\_time,e\_time , date , location.

Constructor for Schedule:  
 To store the values defined above.

Getter for Schedule:

To get the values stored in class object.

Setter for Schedule:

To set the values stored in class object.

END CLASS

*FUNCTION display\_schedule:*

*Time Complexity-O(1), Space Complexity-O(n).*

N stands for the map  
 -Store the mapped value of Id in the pointer of schedule class.

-Print each of the members of mapped values .

**6. Student.h:**

Class Student

Public:

Define id , name,yearOfPlacement , program , cgpa , email , phone as variables.

Constructor for Student:

To store the values defined in public .

Getter for Student:   
To get the values stored in class object.

Setter for Student:   
To set the values stored in class object.

END CLASS

*FUNCTION display\_particularstudent:*

*Time Complexity-O(1), Space Complexity-O(n).*  
 -Store the mapped value of Id in the pointer of student class.

-Print each of the members of mapped values.

**7. Past\_Details.h:**   
Class Alumni

Public:   
Define id , name , phone , cmpname , branch, yearOfPlacement,salary variables.

Constructor for Alumni:  
 To store the values of the variables defined in public.

Getter for Alumni:   
 To get the values stored in class object.

Setter for Alumni:   
 To set the values stored in class object.

END CLASS

*FUNCTION display\_past\_details:*

*Time Complexity-O(n), Space Complexity-O(mn).*

M stands for the number of strings in each entries and the n stands space by map alumni  
 -Store the mapped value of Id in the pointer of student class.

For Loop(beginning of map alumni to end):

Print the all the values of the files.

END LOOP.

*FUNCTION past\_mean:*

*Time Complexity-O(n), Space Complexity-O(1).*

Same as mean  
Procedure: CalculatePastMean(a)

Input:- a (map<string, Alumni\*>) - a map containing Alumni objects indexed IDs

1. Start

2. Set sum and count to 0.

4. For each element ‘i’ in ‘a’:

a. Add ‘i->second->salary’ to sum.

b. Increment count by 1.

5. Calculate the mean by dividing sum by count and store it in mean.

6. Print Mean

7. End .

*FUNCTION past\_median:*

*Time Complexity-O(n)+O(nlogn)=O(n), Space Complexity-O(n).*

Same as median

Procedure: CalculatePastMedian(a)

Input:- a (map<string, Alumni\*>) - a map containing Alumni objects indexed by IDs

1. Start

2. Create an empty vector v to store salary values.

3. For each element ‘i’ in ‘a’:

a. Append 'i->second->salary' to the vector v.

4. Sort the vector v in ascending order.

5. Calculate the length l of the vector v.

6. If ‘l’is even:

a. Calculate median ‘d’ as average of two middle elements:

i. ‘v[(l/2)-1]’ and ‘v[(l/2) + 1 - 1]’.

7. Else, if ‘l’is odd:

Calculate median ‘d’ as middle element: ‘v[((l+1)/2)-1]’.

8. Print Median

9. End.

*FUNCTION past\_highest:*

*Time Complexity-O(n), Space Complexity-O(1).*  
Procedure: FindPastHighestSalary(a)

Input:- a (map<string, Alumni\*>) - a map containing Alumni objects indexed by IDs

1. Start

2. Set 'max' to 0.

3. For each element ‘i’ in ‘a’ :

If ‘i->second->salary’ is greater than ‘max’:

i. Set max to ‘i->second->salary’.

4. Print Highest Package

5. End.

*FUNCTION tokenize\_alumni:*

*Time Complexity-O(n), Space Complexity-O(n).*

N stands for the iteration in alumni map in time and size of map in space

Procedure: TokenizeAlumni(s, a)

Input:

- s (string) - the string to be tokenized

- a (map<string, Alumni\*>) - a map containing Alumni objects indexed by IDs

1. Start

2. Create a stringstream 'line' from the input string 's'.

3. Create a string variable 'temp' to store each token.

4. Create a vector 'row\_info' to store the tokens.

5. While there are tokens to extract from 'line':

a. Extract the next token from 'line' using ',' as the delimiter and store it in temp.

b. Add 'temp' to the 'row\_info' vector.

6. Create a new Alumni object using the tokens in 'row\_info':

a. Set the ID of the Alumni object to 'row\_info[0]'.

b. Set the name of the Alumni object to 'row\_info[1]'.

c. Set the branch of the Alumni object to 'row\_info[2]'.

d. Convert the year of placement in 'row\_info[3]' to an integer and set it as the year of placement of the Alumni object.

e. Set the phone number of the Alumni object to 'row\_info[4]'.

f. Set the company name of the Alumni object to 'row\_info[5]'.

g. Convert the salary in 'row\_info[6]' to an integer and set it as the salary of the Alumni object.

7. Add the newly created Alumni object to the map 'a' with its ID as the key.

8. End.

*FUNCTION Past\_Details:*

*Time Complexity-O(mn)+O(nlogn)=O(mn), Space complexity-O(n)*

M stands for entries in file and n for all maps’ size  
Procedure: PastDetails(year)

Input:

- year (integer) - the year for which past details are requested

1. Start

2. Declare ifstream 'fin' and string variable 'line'.

3. Declare map 'alumni' to store Alumni objects indexed by their IDs.

4. Based on the value of 'year', perform the following:

a. If 'year' is 2023:

i. Open the file "./Data/Student\_2023(Batch).csv" for reading.

ii. Read and discard the header line.

iii. Read each line from the file and tokenize it to create Alumni objects using the 'tokenize\_alumni' function.

iv. Close the file.

b. If 'year' is 2022, 2021, or 2020, perform similar steps as above with corresponding file paths.

5. Prompt user to choose an option:

a. If the user inputs 1, call the 'display\_past\_details' function with the 'alumni' map as the argument.

b. If the user inputs 2, call the 'past\_mean', 'past\_median', and 'past\_highest' functions with the 'alumni' map as the argument.

6. End.

**8. File\_Reader.h:**   
*FUNCTION tokenize\_student:*

*Time Complexity-O(n), Space Complexity-O(n).*   
Procedure: TokenizeStudent(s, ms)

Input:

- s (string) - the string to be tokenized

- ms (map<string, Student\*>) - map containing Student objects indexed by IDs

1. Start

2. Create stringstream 'line' from the input string 's'.

3. Create string variable 'temp' to store each token.

4. Create vector 'row\_info' to store the tokens.

5. While there are tokens to extract from 'line':

a. Extract the next token from 'line' using ',' as the delimiter and store it in 'temp'.

b. Add 'temp' to the 'row\_info' vector.

6. Create new Student object using the tokens in 'row\_info':

a. Set the ID of the Student object to 'row\_info[0]'.

b. Set the name of the Student object to 'row\_info[1]'.

c. Set the branch of the Student object to 'row\_info[2]'.

d. Convert the GPA in 'row\_info[3]' to a double and set it as the GPA of the Student object.

e. Set the email of the Student object to 'row\_info[4]'.

f. Set the phone number of the Student object to 'row\_info[5]'.

g. Set the batch of the Student object to 'row\_info[6]'.

7. Add the newly created Student object to the map 'ms' with its ID as the key.

8. End

*FUNCTION tokenize\_company:*

*Time Complexity-O(n), Space complexity-O(n).*  
Procedure: TokenizeCompany(s, data)

Input:

- s (string) - the string to be tokenized

- data (vector<company>&) - a vector to store Company objects

1. Start

2. Create string variable 'temp' to store each token.

3. Create stringstream 'line' from the input string 's'.

4. Create vector 'c\_row\_info' to store the tokens.

5. While there are tokens to extract from 'line':

a. Extract the next token from 'line' using ',' as the delimiter and store it in 'temp'.

b. Add 'temp' to the 'c\_row\_info' vector.

6. Create new Company object using the tokens in 'c\_row\_info':

a. Set name of Company object to 'c\_row\_info[0]'.

b. Set location of Company object to 'c\_row\_info[1]'.

c. Set field of Company object to 'c\_row\_info[2]'.

d. Set website of Company object to 'c\_row\_info[3]'.

e. Set email of Company object to 'c\_row\_info[4]'.

f. Set phone number of Company object to 'c\_row\_info[5]'.

g. Set package offered by Company object to 'c\_row\_info[6]'.

7. Create a pointer 'c' to the newly created Company object.

8. Add the newly created Company object to the 'data' vector.

9. End.

*FUNCTION tokenize\_internship:*

*Time Complexity-O(n), Space Complexity-O(n).*

Procedure: TokenizeInternship(s, mr)

Input:

- s (string) the string to be tokenized

- mr (map<string, Internship\*>) - map containing Internship objects indexed by IDs

1. Start

2. Create string variable 'temp' to store each token.

3. Create stringstream 'line' from the input string 's'.

4. Create vector 'i\_row\_info' to store the tokens.

5. While there are tokens to extract from 'line':

a. Extract the next token from 'line' using ',' as the delimiter and store it in 'temp'.

b. Add 'temp' to the 'i\_row\_info' vector.

6. Create a new Internship object using the tokens in 'i\_row\_info':

a. Set ID of Internship object to 'i\_row\_info[0]'.

b. Set name of Internship object to 'i\_row\_info[1]'.

c. Set location of Internship object to 'i\_row\_info[2]'.

d. Set field of Internship object to 'i\_row\_info[3]'.

e. Set stipend of Internship object to 'i\_row\_info[4]'.

f. Set start date of Internship object to 'i\_row\_info[5]'.

g. Set end date of Internship object to 'i\_row\_info[6]'.

h. Set eligibility criteria of Internship object to 'i\_row\_info[7]'.

7. Create pointer 'data' to newly created Internship object.

8. Add newly created Internship object to 'mr' map with its ID as key.

9. End.

*FUNCTION tokenizing\_schedule:*

*Time Complexity-O(n), Space Complexity-O(n).*  
Procedure: TokenizeSchedule(s, data)

Input:

- s (string) - string to be tokenized

- data (map<string, Schedule\*>) - map containing Schedule objects indexed by IDs

1. Start

2. Create string variable 'temp' to store each token.

3. Create stringstream 'line' from the input string 's'.

4. Create vector 'row' to store the tokens.

5. While there are tokens to extract from 'line':

a. Extract the next token from 'line' using ',' as the delimiter and store it in 'temp'.

b. Add 'temp' to the 'row' vector.

6. Create new Schedule object using the tokens in 'row':

a. Set ID of Schedule object to 'row[0]'.

b. Set date of Schedule object to 'row[1]'.

c. Set time of Schedule object to 'row[2]'.

d. Set event name of Schedule object to 'row[3]'.

e. Set location of Schedule object to 'row[4]'.

f. Set description of Schedule object to 'row[5]'.

g. Set organizer of Schedule object to 'row[6]'.

7. Create pointer 'sche' to newly created Schedule object.

8. Add newly created Schedule object to 'data' map with its ID as key.

9. End.

*FUNCTION tokenize\_jobOffers:*

*Time Complexity-O(n), Space Complexity-O(n).*  
Procedure: TokenizeJobOffers(s, mj)

Input:

- s (string) -string to be tokenized

- mj (map<string, JobOffer\*>) - map containing JobOffer objects indexed by IDs

1. Start

2. Create string variable 'temp' to store each token.

3. Create stringstream 'line' from the input string 's'.

4. Create vector 'j\_row' to store the tokens.

5. While there are tokens to extract from 'line':

a. Extract the next token from 'line' using ',' as the delimiter and store it in 'temp'.

b. Add 'temp' to the 'j\_row' vector.

6. Create new JobOffer object using the tokens in 'j\_row':

a. Set ID of JobOffer object to 'j\_row[0]'.

b. Set name of JobOffer object to 'j\_row[1]'.

c. Set batch of JobOffer object to 'j\_row[2]'.

d. Set written test status of JobOffer object to 'j\_row[3]'.

e. Set interview status of JobOffer object to 'j\_row[4]'.

f. Set placement status of JobOffer object to 'j\_row[5]'.

g. Convert 'j\_row[6]' to an integer and set it as the package of the JobOffer object.

7. Create pointer 'job' to newly created JobOffer object.

8. Add newly created JobOffer object to 'mj' map with its ID as key.

9. End.

*FUNCTION read\_file:*

*Time Complexity-O(n), Space Complexity-O(n).*

Procedure: ReadFile(students, companies, internships, schedule, jobOffers)

Input:

- students (map<string, Student\*>) - a map containing Student objects indexed by their IDs

- companies (vector<company>&) - a vector to store Company objects

- internships (map<string, Internship\*>) - a map containing Internship objects indexed by their IDs

- schedule (map<string, Schedule\*>) - a map containing Schedule objects indexed by their IDs

- jobOffers (map<string, JobOffer\*>) - a map containing JobOffer objects indexed by their IDs

1. Start

2. Declare input file stream 'fin' and string variable 'line'.

3. Open file for reading.

4. Read and discard the header line.

5. While there are lines to read from the file:

a. Read a line from the file.

b. Tokenize the line and create a Student object using the 'tokenize\_student' function.

6. Close file.

7. Clear end-of-file flag.

8. Repeat steps 3-7 for the Company, Internship, JobOffer, and Schedule files.

9. End .

**9. Display.h:**

*FUNCTION display\_allstudents:*

*Time Complexity-O(n), Space Complexity-O(m).*

Procedure: DisplayAllStudents(Student)

Input:- Student (map<string, Student\*>) - a map containing Student objects indexed by IDs

1. Start

2. Print a header indicating the list of students.

3. Print column headers for ID, Name, Branch, CGPA, and Year.

4. For each student in the 'Student' map:

a. Print the student's ID, name, branch, CGPA, and year of placement.

b. If the student's CGPA is less than 3.0, indicate that they are not eligible for placement.

c. If the student's CGPA is 3.0 or higher, indicate that they are eligible for placement.

5. End.

*FUNCTION company\_based\_details:*

*Time Complexity-O(n), Space Complexity:O(kn).*  
Procedure: CompanyBasedDetails(j)

Input:- j (map<string, JobOffer\*>) - a map containing JobOffer objects indexed by IDs

1. Start

2. Prompt user to enter name of the company.

3. Read the company name from user.

4. Print a header indicating the list of students placed in the specified company.

5. Print column headers for ID, Name, and Placement Status.

6. For each job offer in the 'j' map:

a. If the company name of the job offer matches the specified company:

i. Print the ID, name, and placement status of the student associated with the job offer.

7. End.

*FUNCTION batchwise\_placement\_details:*

*Time Complexity-O(n) Space Complexity-O(kn).*  
Procedure: BatchwisePlacementDetails(j)

Input:- j (map<string, JobOffer\*>) - a map containing JobOffer objects indexed by IDs

1. Start

2. Prompt the user to specify the batch (M.Tech ICT/B.Tech ICT).

3. Read the batch name from the user.

4. Print a header indicating the list of students in the specified batch and their placement details.

5. Print column headers for ID, Name, Batch, and Placement Status.

6. For each job offer in the 'j' map:

a. If the batch of the job offer matches the specified batch:

i. Print the ID, name, batch, and placement status of the student associated with the job offer.

ii. Print a separator line.

7. End.

*FUNCTION display\_choice1:*

*Time Complexity-O1, Space Complexity-O(1).*

Procedure: DisplayChoice1(choice1, students, internships, schedule, jf)

Input:

- choice1 (int) - the user's choice

- students (map<string, Student\*>) - a map containing Student objects indexed by their IDs

- internships (map<string, Internship\*>) - a map containing Internship objects indexed by their IDs

- schedule (map<string, Schedule\*>) - a map containing Schedule objects indexed by their IDs

- jf (map<string, JobOffer\*>) - a map containing JobOffer objects indexed by their IDs

1. Start

2. If choice1 is 1:

a. Print a message indicating that the data of all students will be displayed.

b. Call the 'display\_allstudents' function to display the data of all students.

3. Else if choice1 is 2:

a. Print a message indicating that the user can view details of a particular student.

b. Prompt the user to select the type of details to view (My Details, Internship Details, Schedule Details, Placement Process Status).

c. Read the user's choice.

d. Prompt the user to enter the ID number of the student.

e. Based on the user's choice:

i. If the choice is 1, call the 'display\_particularstudent' function to display details of the particular student.

ii. If the choice is 2, call the 'display\_internship' function to display internship details of the particular student.

iii. If the choice is 3, call the 'display\_schedule' function to display schedule details of the particular student.

iv. If the choice is 4, call the 'display\_joboffers' function to display placement process status of the particular student.

4. End.

*FUNCTION display\_allcompany:*

*Time Complexity-O(n), Space Complexity-O(m).*

Here m stands for the entries in company vector with type company.  
Procedure: DisplayAllCompany(company)

Input:- company (vector<Company>&) - a vector containing Company object.

1. Start

2. Print a header indicating the list of companies.

3. Print column headers for Serial No, Company Name, Address of Company, Package, Hired Students, Criteria, No. of Alumni, and Present at Campus.

4. Initialize index variable to 1.

5. For each company in the 'company' vector:

a. Print the company details including name, address, package, hired students, criteria, number of alumni, and presence at campus.

b. Increment the index variable.

6. End.

*FUNCTION display\_choice2:*

*Time Complexity-O(1), Space Complexity-O(1).*  
Procedure: DisplayChoice2(choice2, j, student)

Input:

- choice2 (int) - the user's choice

- j (map<string, JobOffer\*>) - a map containing JobOffer objects indexed by their IDs

- student (map<string, Student\*>) - a map containing Student objects indexed by IDs

1. Start

2. Based on the user's choice:

a. If choice2 is 1:

i. Print a message indicating that the user opted to view placement details through company names.

ii. Call the 'company\_based\_details' function to display placement details based on company names.

b. If choice2 is 2:

i. Print a message indicating that the user opted to view placement details batch-wise.

ii. Call the 'batchwise\_placement\_details' function to display placement details batch-wise.

c. If choice2 is 3:

i. Print a message indicating that the user opted to view placement details based on statistics.

ii. Call the 'max\_package', 'mean\_package', and 'median\_package' functions to display maximum, mean, and median package statistics respectively.

d. If choice2 is 4:

i. Print a message indicating that the user opted to view past placement details.

ii. Prompt the user to enter the year in YYYY format.

iii. Call the 'Past\_details' function to display past placement details for the specified year.

3. End.

*FUNCTION display\_choice:*

*Time Complexity-O(1), Space Complexity-O(1).*  
Procedure: DisplayChoice(choice, students, company, internships, schedule, jf)

Input:

- choice (int) - the user's choice

- students (map<string, Student\*>) -map containing Student objects indexed by IDs

- company (vector<Company>&) - vector containing Company objects

- internships (map<string, Internship\*>) - map containing Internship objects indexed by IDs

- schedule (map<string, Schedule\*>) - map containing Schedule objects indexed by IDs

- jf (map<string, JobOffer\*>) -map containing JobOffer objects indexed by IDs

1. Start

2. Based on the user's choice:

a. If choice is 1:

i. Print a message indicating that the user opted to display ongoing process of placement.

ii. Prompt the user to select further options:

A. 1 - View details of all students.

B. 2 - View detail of a particular student.

iii. Read the user's choice.

iv. Call the 'display\_choice1' function with the user's choice and other necessary parameters.

b. If choice is 2:

i. Print a message indicating that the user opted to view placement details.

ii. Present options for different types of placement details:

A. 1 - Display current placement details based on company name.

B. 2 - Display current placement details batch-wise.

C. 3 - Display current placement statistics.

D. 4 - Display past placement details.

iii. Read the user's choice.

iv. Call the 'display\_choice2' function with the user's choice and other necessary parameters.

c. If choice is 3:

i. Print a message indicating that the user opted to view company details.

ii. Call the 'display\_allcompany' function to display details of all companies.

3. End.

*FUNCTION display:*

*Time Complexity-O(1), Space Complexity-O(1).*  
Procedure: Display(students, company, internships, schedule, jf)

Input:

- students (map<string, Student\*>) - map containing Student objects indexed by IDs

- company (vector<Company>&) - vector containing Company objects

- internships (map<string, Internship\*>) - map containing Internship objects indexed by IDs

- schedule (map<string, Schedule\*>) - map containing Schedule objects indexed by IDs

- jf (map<string, JobOffer\*>) - map containing JobOffer objects indexed by IDs

1. Start

2. Print options for the user:

a. Enter 1 to display ongoing process of placement.

b. Enter 2 to display the placement program details.

c. Enter 3 to display company details.

d. Prompt user to enter their choice.

3. Read user's choice.

4. Call 'display\_choice' function with user's choice.

5. End.

**10. Main.cpp: Time Complexity-O(1),Space Complexity-O(mn)**

Here m is the sum of number of entries in each file, and n is the sum of size of each vector.  
*Function main():*

Output: Returns 0 upon successful completion

1. Start

2. Declare variables:

- students: a map of Student pointers indexed by their IDs

- schedule: a map of Schedule pointers indexed by their IDs

- Joboffers: a map of JobOffer pointers indexed by their IDs

- company: a vector of Company objects

- internships: a map of Internship pointers indexed by their IDs

3. Call the 'read\_file' function to populate the data structures with information from files.

a. Pass students, company, internships, schedule, and Joboffers as arguments to the 'read\_file' function.

4. Call the 'display' function to provide options for displaying information and interact with the user.

a. Pass students, company, internships, schedule, and Joboffers as arguments to the 'display' function.

5. End.