Project Report
“Student Group Maker”

Group: Searchless

1. Problem Statement

Traditional methods for professors to create groups of students can potentially be time consuming and inefficient due to the large amount of students in each class. And although there may be ways to quickly randomize groups of students, there can still be problems of having students with high scores in the same group instead of spread out throughout all groups, thus creating groups with unbalanced levels of skill. Since professors often manage student scores manually or in spreadsheets, we can take those scores to create balanced groups where students with high scores are equally shared among all groups without having to sort out the scores manually.

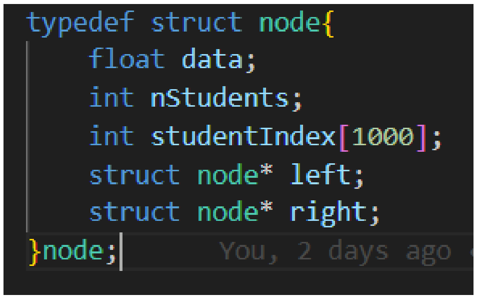
2. Solution with functionalities

2.1 Main solution and functionality

The solution for our grouping problem will be dealt with the CreateGroup function which can arrange students into desired number of groups based on score. The function will use the following data structures.

- Binary Search Tree:

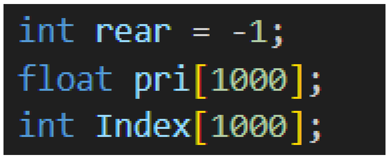
  Will be used to sort the student scores into 4 quadrants based on max score. Each node will store a number representing the quartile(float data) that will be used to determine what enters the quadrant. It will also store the amount of students in the quadrant(int nStudents) and an array storing the indexes of the students within the quadrant.



```
typedef struct node{
    float data;
    int nStudents;
    int studentIndex[1000];
    struct node* left;
    struct node* right;
}node;        You, 2 days ago
```

- Priority queue:

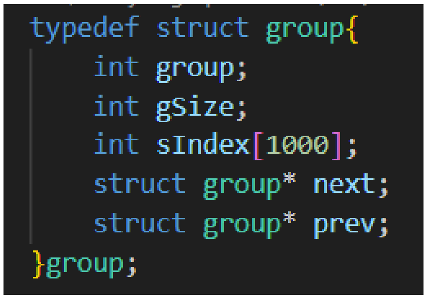
  Will be used to store all the students from a quadrant and then dequeues each student into each separate group, the highest score gets dequeued first.



```
int rear = -1;
float pri[1000];
int Index[1000];
```

- Doubly linked list:

  Will be used to represent a list of groups, each node represents one individual group. Each node will contain the group number(int group), the number of students in the group(int gSize) and an array containing the indexes of the students within the group(int sIndex).



```
typedef struct group{
    int group;
    int gSize;
    int sIndex[1000];
    struct group* next;
    struct group* prev;
}group;
```

2.1 Other Functionalities

2.1.1 File Edit Functions

1.addStudent

Add a student to the list if there's space.

• Stores new student data, updates max score if needed, and increments total student count.

2.deleteStudent

Deletes a student by ID, decrements total count of the students in the list, and recomputes max score.

3.editStudent

Edits an existing student by ID, updates name, section, and score, and recomputes max score.

4.saveToCSV

Saves all student data to a CSV file, saves headers and student data to the file.

2.1.2 Search Functions

1.searchbyID- Finds and prints student info by ID.

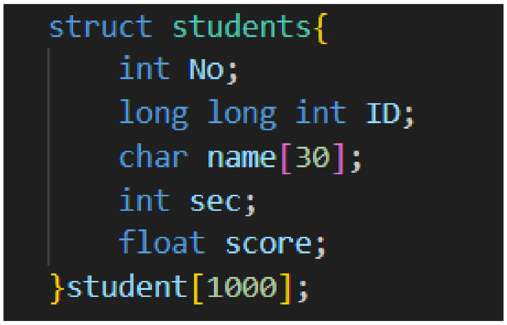
2.searchbyName- Finds and prints students whose names match.

3.searchScoreRange- Prints students with scores within a given range.

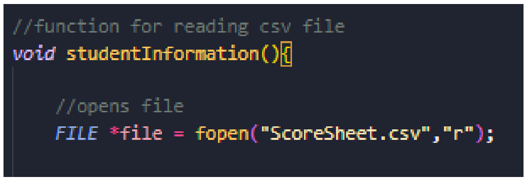
3. Code Walkthrough with relevant data structures

3.1 studentInformation function

This function is for collecting and storing all of the students information(No., student ID, Name, Sec, and scores) from the csv file.



```c
struct students{
    int No;
    long long int ID;
    char name[30];
    int sec;
    float score;
}student[1000];
```

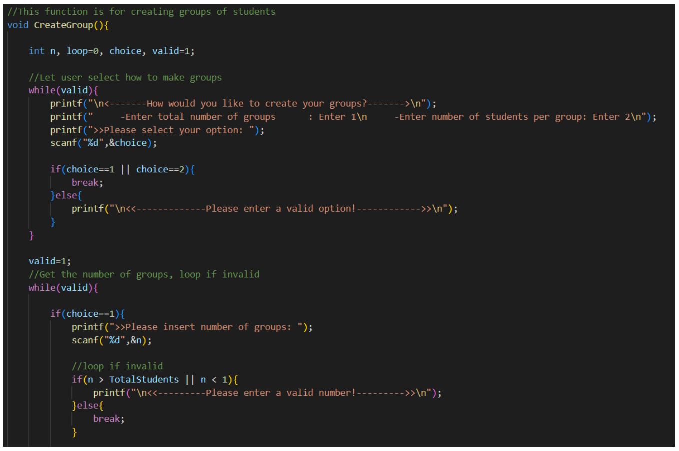


```c
//function for reading csv file
void studentInformation(){

    //opens file
    FILE *file = fopen("ScoreSheet.csv","r");
```

3.2 Functions for group creation

3.2.1 CreateGroup function

This function is for creating groups of students based on their score to create groups with balanced skill levels.



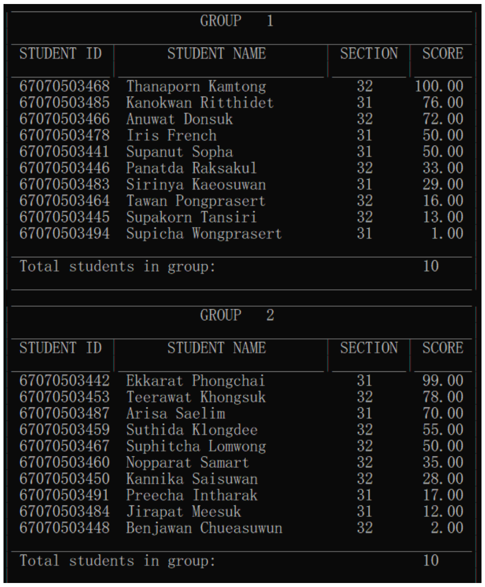
```c
//This function is for creating groups of students
void CreateGroup(){

    int n, loop=0, choice, valid=1;

    //Let user select how to make groups
    while(valid){
        printf("\n<-------How would you like to create your groups?------->\n");
        printf("     -Enter total number of groups      : Enter 1\n     -Enter number of students per group: Enter 2\n");
        printf(">>Please select your option: ");
        scanf("%d",&choice);

        if(choice==1 || choice==2){
            break;
        }else{
            printf("\n<<------------Please enter a valid option!------------>>\n");
        }
    }

    valid=1;
    //Get the number of groups, loop if invalid
    while(valid){

        if(choice==1){
            printf(">>Please insert number of groups: ");
            scanf("%d",&n);

            //loop if invalid
            if(n > TotalStudents || n < 1){
                printf("\n<<---------Please enter a valid number!--------->>\n");
            }else{
                break;
            }
```

Process:

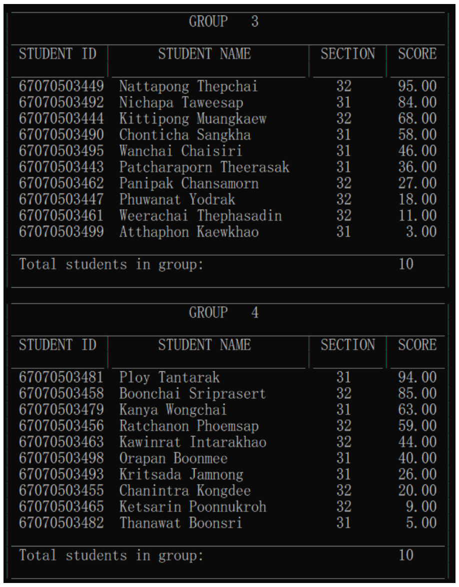
1. It starts by receiving the desired amount of groups from the user.
2. Then it creates the binary tree and sorts the students into quadrants based on score.
3. A doubly linked list with nodes equal to the amount of groups will be made, each node representing 1 group.
4. Then this function will enqueue a quadrant of students(staring from the top quadrant) and then dequeue each student into the doubly linked list, each student going into a different node by going to the next node, after 1 loop is completed the students will start going to previous node instead, going back and forth between all the nodes in the doubly linked list(1 loop = 1 end of the linked list to the other end of the linked list).
5. When the priority queue is empty, step 4 is repeated with the next quadrant in line. Repeat it until the students from all the quadrants are put into groups.
6. Then traverse through the linked list to display all the groups and the students in each group.
7. An option to create an excel file will be given.
8. Delete the doubly linked list to prepare for the next operations.

Example output:

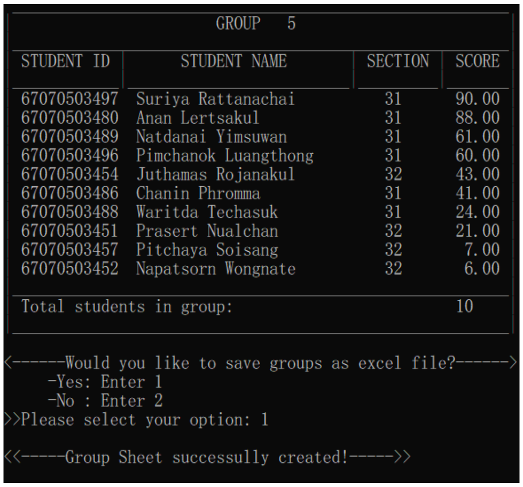
Input 5 groups to be made-



```
                      GROUP   1

STUDENT ID   STUDENT NAME                SECTION   SCORE

67070503468  Thanaporn Kamtong             32     100.00
67070503485  Kanokwan Ritthidet            31      76.00
67070503466  Anuwat Donsuk                 32      72.00
67070503478  Iris French                   31      50.00
67070503441  Supanut Sopha                 31      50.00
67070503446  Panatda Raksakul              32      33.00
67070503483  Sirinya Kaeosuwan             31      29.00
67070503464  Tawan Pongprasert             32      16.00
67070503445  Supakorn Tansiri              32      13.00
67070503494  Supicha Wongprasert           31       1.00

Total students in group:                          10

                      GROUP   2

STUDENT ID   STUDENT NAME                SECTION   SCORE

67070503442  Ekkarat Phongchai             31      99.00
67070503453  Teerawat Khongsuk             32      78.00
67070503487  Arisa Saelim                  31      70.00
67070503459  Suthida Klongdee              32      55.00
67070503467  Suphitcha Lomwong             32      50.00
67070503460  Nopparat Samart               32      35.00
67070503450  Kannika Saisuwan              32      28.00
67070503491  Preecha Intharak              31      17.00
67070503484  Jirapat Meesuk                31      12.00
67070503448  Benjawan Chueasuwun           32       2.00

Total students in group:                          10
```



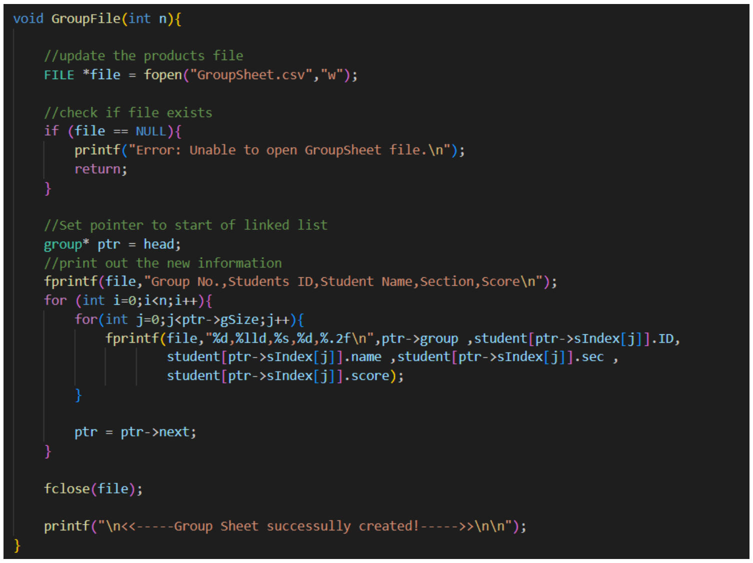
```
                      GROUP   3

STUDENT ID   STUDENT NAME                SECTION   SCORE

67070503449  Nattapong Thepchai            32      95.00
67070503492  Nichapa Taweesap              31      84.00
67070503444  Kittipong Muangkaew           32      68.00
67070503490  Chonticha Sangkha             31      58.00
67070503495  Wanchai Chaisiri              31      46.00
67070503443  Patcharaporn Theerasak        31      36.00
67070503462  Panipak Chansamorn            32      27.00
67070503447  Phuwanat Yodrak               32      18.00
67070503461  Weerachai Thephasadin         32      11.00
67070503499  Atthaphon Kaewkhao            31       3.00

Total students in group:                          10

                      GROUP   4

STUDENT ID   STUDENT NAME                SECTION   SCORE

67070503481  Ploy Tantarak                 31      94.00
67070503458  Boonchai Sriprasert           32      85.00
67070503479  Kanya Wongchai                31      63.00
67070503456  Ratchanon Phoemsap            32      59.00
67070503463  Kawinrat Intarakhao           32      44.00
67070503498  Orapan Boonmee                31      40.00
67070503493  Kritsada Jamnong              31      26.00
67070503455  Chanintra Kongdee             32      20.00
67070503465  Ketsarin Poonnukroh           32       9.00
67070503482  Thanawat Boonsri              31       5.00

Total students in group:                          10
```



```
                      GROUP   5

STUDENT ID   STUDENT NAME                SECTION   SCORE

67070503497  Suriya Rattanachai            31      90.00
67070503480  Anan Lertsakul                31      88.00
67070503489  Natdanai Yimsuwan             31      61.00
67070503496  Pimchanok Luangthong          31      60.00
67070503454  Juthamas Rojanakul            32      43.00
67070503486  Chanin Phromma                31      41.00
67070503488  Waritda Techasuk              31      24.00
67070503451  Prasert Nualchan              32      21.00
67070503457  Pitchaya Soisang              32       7.00
67070503452  Napatsorn Wongnate            32       6.00

Total students in group:                          10

<------Would you like to save groups as excel file?------>
    -Yes: Enter 1
    -No : Enter 2
>>Please select your option: 1

<<-----Group Sheet successully created!----->>
```

3.2.2 GroupFile function

This function is for printing out all the groups and its members into an excel file.

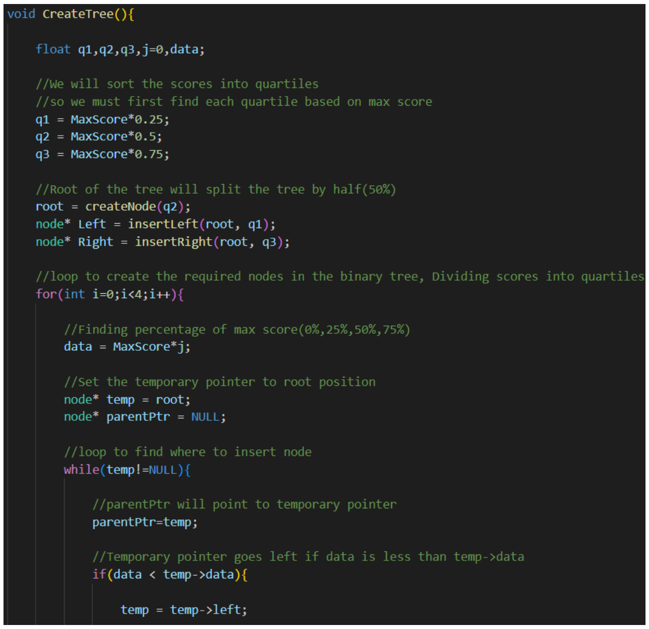


```c
void GroupFile(int n){

    //update the products file
    FILE *file = fopen("GroupSheet.csv","w");

    //check if file exists
    if (file == NULL){
        printf("Error: Unable to open GroupSheet file.\n");
        return;
    }

    //Set pointer to start of linked list
    group* ptr = head;
    //print out the new information
    fprintf(file,"Group No.,Students ID,Student Name,Section,Score\n");
    for (int i=0;i<n;i++){
        for(int j=0;j<ptr->gSize;j++){
            fprintf(file,"%d,%lld,%s,%d,%.2f\n",ptr->group ,student[ptr->sIndex[j]].ID,
                    student[ptr->sIndex[j]].name ,student[ptr->sIndex[j]].sec ,
                    student[ptr->sIndex[j]].score);
        }

        ptr = ptr->next;
    }

    fclose(file);

    printf("\n<<-----Group Sheet successully created!----->>\n\n");
}
```

3.2.3 Functions involving Binary Search Tree

CreateTree function

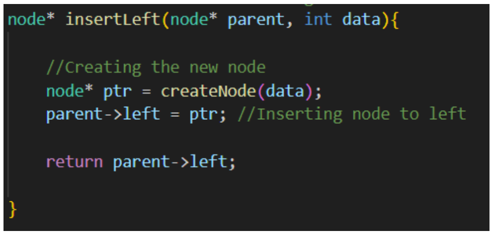
This function is for creating the binary search tree that will be used to separate all of the students into 4 quadrants based on score, each quadrant will store the number of students in the quadrant and an array of the indexes of the students within the quadrant. The quartiles are calculated based on the max score. This function calls upon the insertLeft and insertRight functions to help construct the tree.



```c
void CreateTree(){

    float q1,q2,q3,j=0,data;

    //We will sort the scores into quartiles
    //so we must first find each quartile based on max score
    q1 = MaxScore*0.25;
    q2 = MaxScore*0.5;
    q3 = MaxScore*0.75;

    //Root of the tree will split the tree by half(50%)
    root = createNode(q2);
    node* Left = insertLeft(root, q1);
    node* Right = insertRight(root, q3);

    //loop to create the required nodes in the binary tree, Dividing scores into quartiles
    for(int i=0;i<4;i++){

        //Finding percentage of max score(0%,25%,50%,75%)
        data = MaxScore*j;

        //Set the temporary pointer to root position
        node* temp = root;
        node* parentPtr = NULL;

        //loop to find where to insert node
        while(temp!=NULL){

            //parentPtr will point to temporary pointer
            parentPtr=temp;

            //Temporary pointer goes left if data is less than temp->data
            if(data < temp->data){

                temp = temp->left;
```

insertLeft function

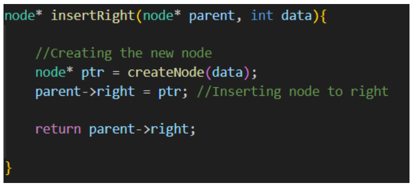
This function inserts a new node to the left of the parent node.



```c
node* insertLeft(node* parent, int data){

    //Creating the new node
    node* ptr = createNode(data);
    parent->left = ptr; //Inserting node to left

    return parent->left;

}
```

insertRight function

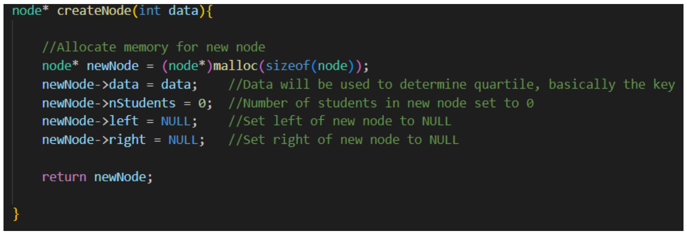
This function inserts a new node to the right of the parent node.



```c
node* insertRight(node* parent, int data){

    //Creating the new node
    node* ptr = createNode(data);
    parent->right = ptr; //Inserting node to right

    return parent->right;

}
```

createNode function

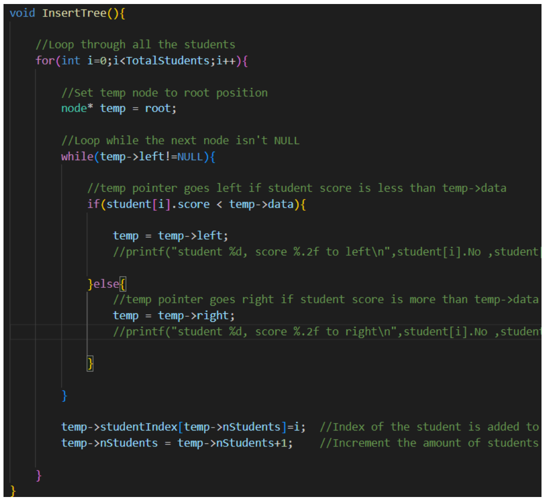
This function creates a new binary tree node and initializes the information within the node. This function is used in the insertLeft and insertRight functions.



```c
node* createNode(int data){

    //Allocate memory for new node
    node* newNode = (node*)malloc(sizeof(node));
    newNode->data = data;    //Data will be used to determine quartile, basically the key
    newNode->nStudents = 0;  //Number of students in new node set to 0
    newNode->left = NULL;    //Set left of new node to NULL
    newNode->right = NULL;   //Set right of new node to NULL

    return newNode;

}
```

InsertTree function

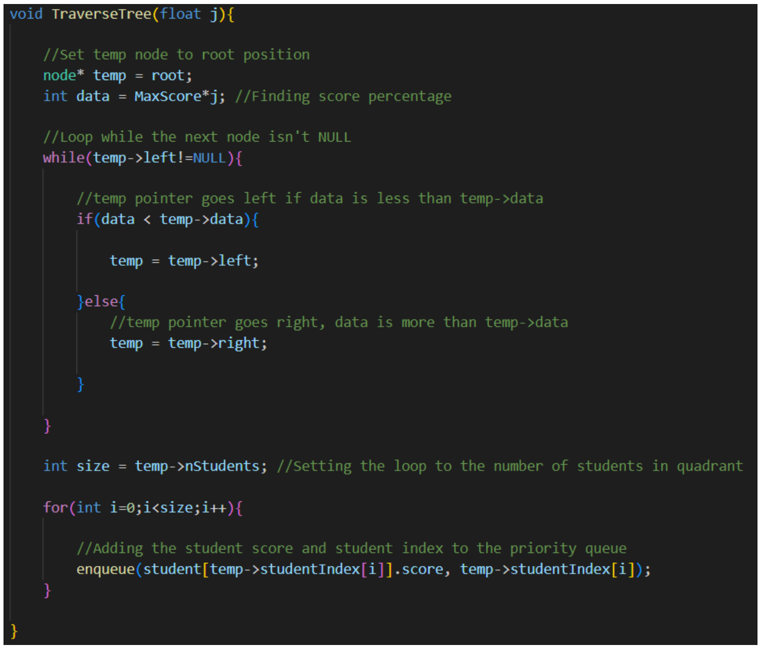
This function is for inserting the index of each student into their respective quadrants, while also updating the number of students in each quadrant.



```c
void InsertTree(){

    //Loop through all the students
    for(int i=0;i<TotalStudents;i++){

        //Set temp node to root position
        node* temp = root;

        //Loop while the next node isn't NULL
        while(temp->left!=NULL){

            //temp pointer goes left if student score is less than temp->data
            if(student[i].score < temp->data){

                temp = temp->left;
                //printf("student %d, score %.2f to left\n",student[i].No ,student

            }else{
                //temp pointer goes right if student score is more than temp->data
                temp = temp->right;
                //printf("student %d, score %.2f to right\n",student[i].No ,studen

            }

        }

        temp->studentIndex[temp->nStudents]=i;  //Index of the student is added to
        temp->nStudents = temp->nStudents+1;    //Increment the amount of students

    }
}
```

TraverseTree function

This function is for going into a specific quadrant and then adding all the students within the quadrant to the priority queue. This function calls upon the enqueue function.



```c
void TraverseTree(float j){

    //Set temp node to root position
    node* temp = root;
    int data = MaxScore*j; //Finding score percentage

    //Loop while the next node isn't NULL
    while(temp->left!=NULL){

        //temp pointer goes left if data is less than temp->data
        if(data < temp->data){

            temp = temp->left;

        }else{
            //temp pointer goes right, data is more than temp->data
            temp = temp->right;

        }

    }

    int size = temp->nStudents; //Setting the loop to the number of students in quadrant

    for(int i=0;i<size;i++){

        //Adding the student score and student index to the priority queue
        enqueue(student[temp->studentIndex[i]].score, temp->studentIndex[i]);
    }

}
```

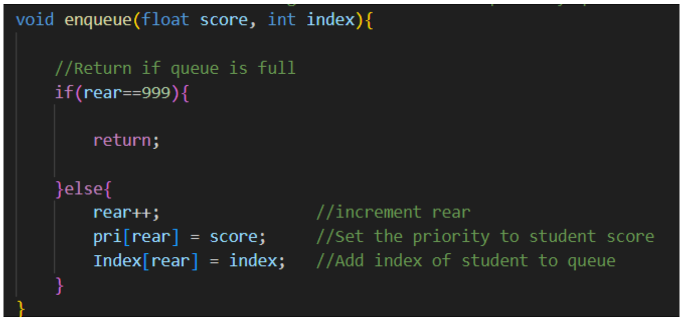
DeleteTree function

This function is for emptying the tree and freeing all the nodes within it.

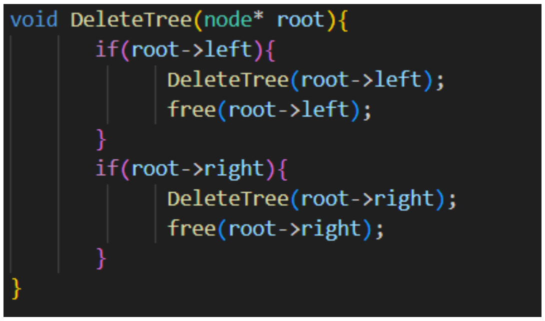
3.2.4 Functions involving priority queue

enqueue function

This Function is for adding the student index to the priority queue.



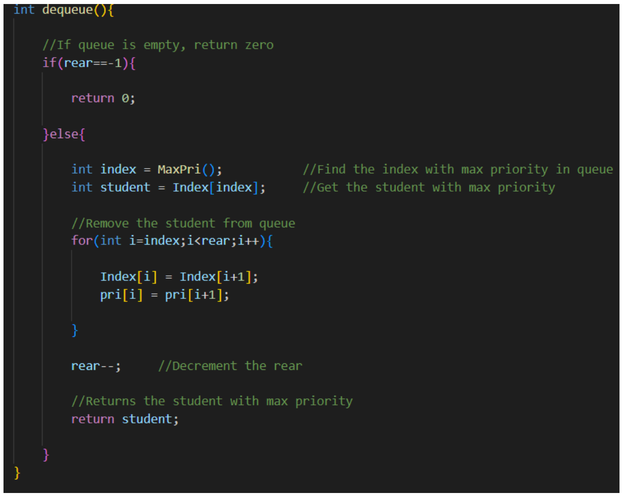
```c
void enqueue(float score, int index){

    //Return if queue is full
    if(rear==999){

        return;

    }else{
        rear++;                 //increment rear
        pri[rear] = score;      //Set the priority to student score
        Index[rear] = index;    //Add index of student to queue
    }
}
```



```c
void DeleteTree(node* root){
        if(root->left){
            DeleteTree(root->left);
            free(root->left);
        }
        if(root->right){
            DeleteTree(root->right);
            free(root->right);
        }
}
```

dequeue function

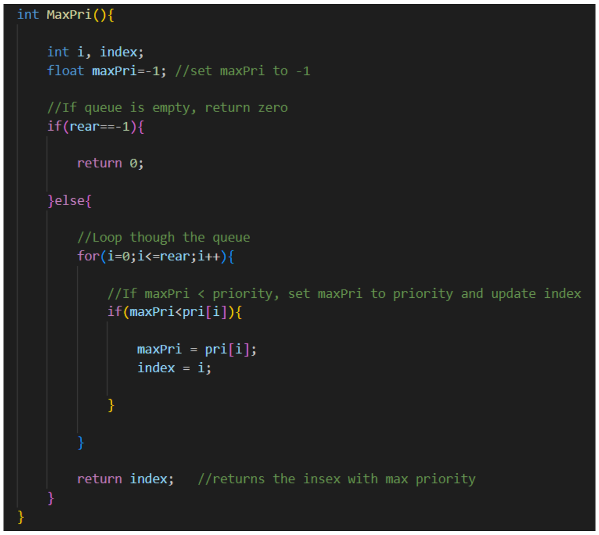
This function is for getting student indexes out of the queue, the highest score goes out first. Uses the function MaxPri to find the student with the highest score.



```
int dequeue(){

    //If queue is empty, return zero
    if(rear==-1){

        return 0;

    }else{

        int index = MaxPri();           //Find the index with max priority in queue
        int student = Index[index];     //Get the student with max priority

        //Remove the student from queue
        for(int i=index;i<rear;i++){

            Index[i] = Index[i+1];
            pri[i] = pri[i+1];

        }

        rear--;     //Decrement the rear

        //Returns the student with max priority
        return student;

    }
}
```

MaxPri function

This function is for getting the item with the max priority within the queue.

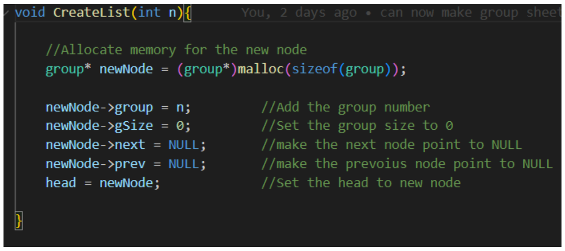


```
int MaxPri(){

    int i, index;
    float maxPri=-1; //set maxPri to -1

    //If queue is empty, return zero
    if(rear==-1){

        return 0;

    }else{

        //Loop though the queue
        for(i=0;i<=rear;i++){

            //If maxPri < priority, set maxPri to priority and update index
            if(maxPri<pri[i]){

                maxPri = pri[i];
                index = i;

            }

        }

        return index;   //returns the insex with max priority
    }
}
```

3.2.5 Functions involving doubly linked list

CreateList function

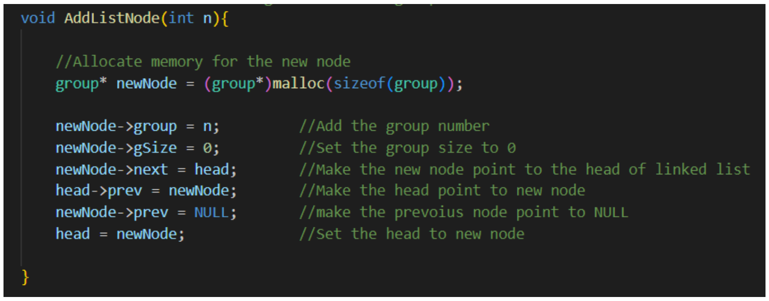
This function is for creating the last node of the linked list, creating the list by starting from the last node.



```
void CreateList(int n){      You, 2 days ago • can now make group sheet

    //Allocate memory for the new node
    group* newNode = (group*)malloc(sizeof(group));

    newNode->group = n;         //Add the group number
    newNode->gSize = 0;         //Set the group size to 0
    newNode->next = NULL;       //make the next node point to NULL
    newNode->prev = NULL;       //make the prevoius node point to NULL
    head = newNode;             //Set the head to new node

}
```

AddListNode function

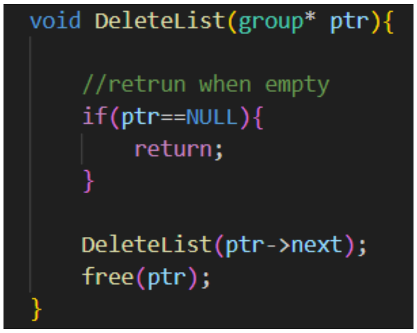
This function is for adding more nodes to the doubly linked list



```c
void AddListNode(int n){

    //Allocate memory for the new node
    group* newNode = (group*)malloc(sizeof(group));

    newNode->group = n;         //Add the group number
    newNode->gSize = 0;         //Set the group size to 0
    newNode->next = head;       //Make the new node point to the head of linked list
    head->prev = newNode;       //Make the head point to new node
    newNode->prev = NULL;       //make the prevoius node point to NULL
    head = newNode;             //Set the head to new node

}
```

DeleteList function

This function is for deleting the doubly linked list



```c
void DeleteList(group* ptr){

    //retrun when empty
    if(ptr==NULL){
        return;
    }

    DeleteList(ptr->next);
    free(ptr);
}
```

4. Time Complexity Analysis

4.1 Time complexity of adding students to binary tree

worst case and best case is $O(n^2)$

4.2 Time complexity of adding students to groups

worst case and best case is O(n log n)

5. Team Member Responsibilities

1. Iris French 67070503478: In charge of making the group creation feature.
2. Supanut Sopha 67070503441: Main program menu and interface
3. Al Xander James Ybanez 67070503450: In charge of creating functions to manage student records and searching for students based on ID, name, or score range.

6. References as Needed

EnjoyAlgorithm. (2022, November 16). *Analysis of loop in programming*. Medium. https://medium.com/enjoy-algorithm/analysis-of-loop-in-programming-cc9a644ef8cd

GeeksforGeeks. (2022, September 6). *Student record management system using linked list*. https://www.geeksforgeeks.org/student-record-management-system-using-linked-list/

GeeksforGeeks. (2022, November 29). *Student information management system*. https://www.geeksforgeeks.org/student-information-management-system/