UNIVERSITY OF KWAZULU-NATAL
INYUVESI YAKWAZULU-NATALI

COMP315

Individual Project Documentation

Members involved
1. Hakeem Hoopdeo - 218067924
2. Nikiel Ramawthar - 218007620

TABLE OF CONTENTS

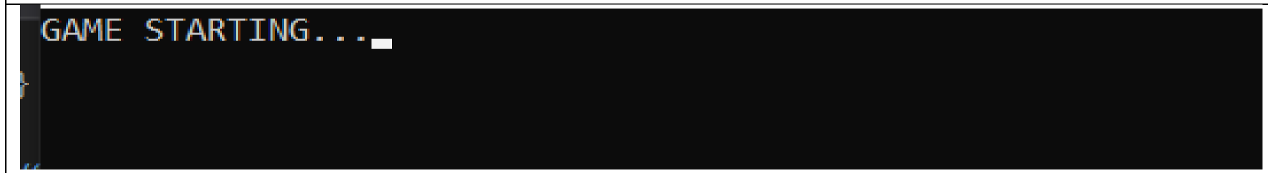
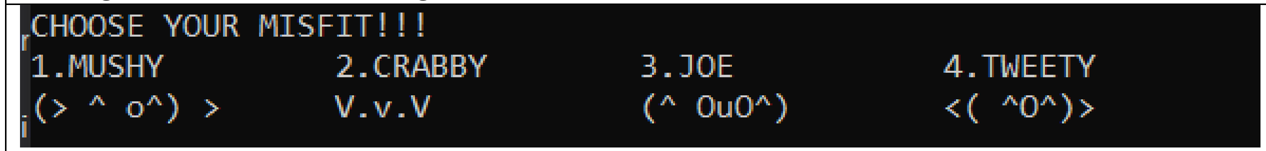
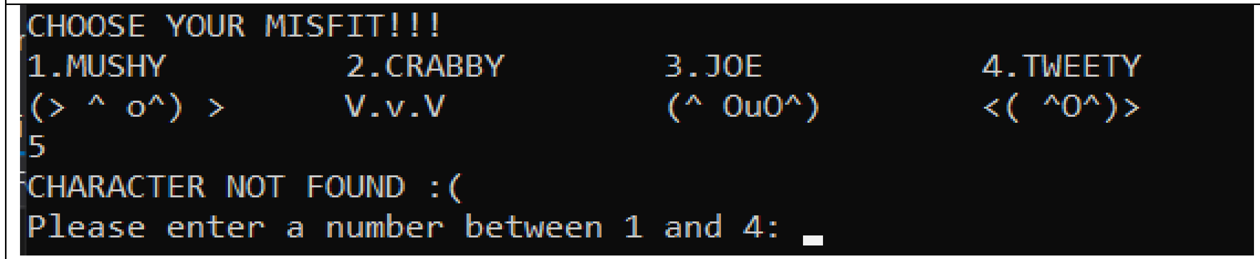
Introduction

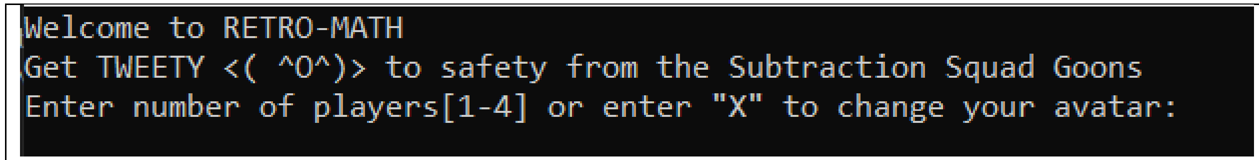
RETRO-MATH. A console-based Mathematical quiz game, which sees a player solve trivial math problems to save their character from the hands of the Subtraction Squad. The development of the game was intended to help people who suffer from dyscalculia as well as create a virtual safe-haven for our users who want to seek refuge from the mundane and repetitive nature of chalkboard and book learning.

The game is set up such that players are given a randomly generated mathematical equation and statement based on the difficulty level chosen. For example, the mathematical expression could be 2 x 2 and a statement which correlates to this would be: Is 4 incorrect? Players would need to answer true or false. Denoted by symbols: T, t, F, or f. Player scores are time-based, meaning it is possible for a player to answer correctly, but consume vast amount of time trying to solve the equation.

At the end of the game the player finds out if he has saved his character or not. This is done to prevent discouraging players from losing hope halfway through the game. We want the players to feel at home and comfortable in this gaming environment, and not make them feel as if they are sitting in a 2-hour examination hall. The game helps teach a mathematical concept which many are familiar with but still do not understand. This concept is BODMAS. The order a person should follow when solving math equations.

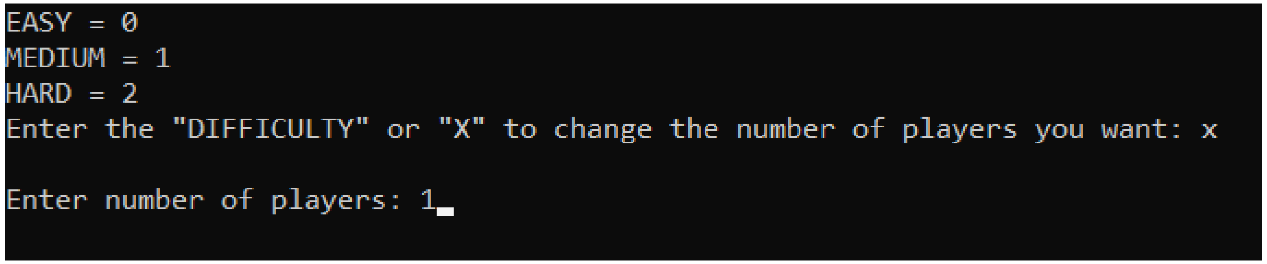
Screenshots

Screenshots with Explanation below
 GAME STARTING... Loading screen before the actual game commences. Screen loads for 2 seconds.
 CHOOSE YOUR MISFIT!!! 1.MUSHY (> ^ o^) > 2.CRABBY V.v.V 3.JOE (^ OuO^) 4.TWEETY <(^O^)> At the beginning of the game. The player is prompted to pick a character for the game.
 CHOOSE YOUR MISFIT!!! 1.MUSHY (> ^ o^) > 2.CRABBY V.v.V 3.JOE (^ OuO^) 4.TWEETY <(^O^)> 5 CHARACTER NOT FOUND :(Please enter a number between 1 and 4: This is our character selection screen. If the player enters a number out of range or a letter as input, we display that it was an invalid character and request the user to re-enter their choice.



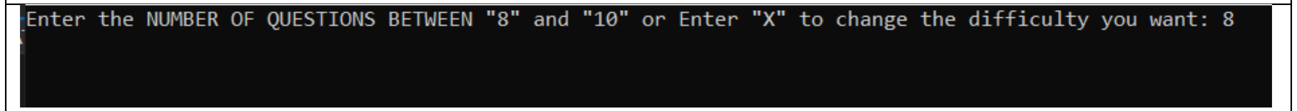
```
Welcome to RETRO-MATH
Get TWEETY <( ^O^)> to safety from the Subtraction Squad Goons
Enter number of players[1-4] or enter "X" to change your avatar:
```

The user is prompted to enter crucial input that governs how the game is played. Here we prompt the user to enter up to 4 players. The user also has the option to enter x and change their character choice.



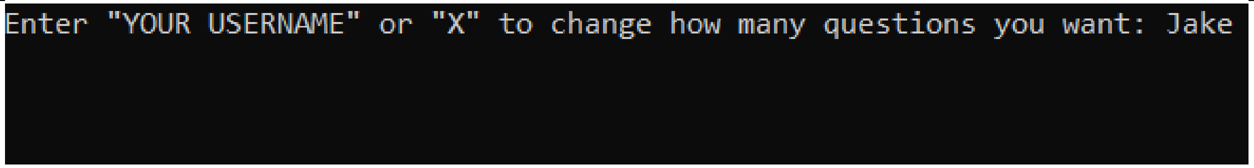
```
EASY = 0
MEDIUM = 1
HARD = 2
Enter the "DIFFICULTY" or "X" to change the number of players you want: x

Enter number of players: 1_
```

Here we prompt the user to enter the difficulty level using the given mapping. Should the user not be content with the previous input (i.e. the number of players that the user entered) due to a typo; we allow the user to rollback (repeat the previous action). This is simply done by entering an X.



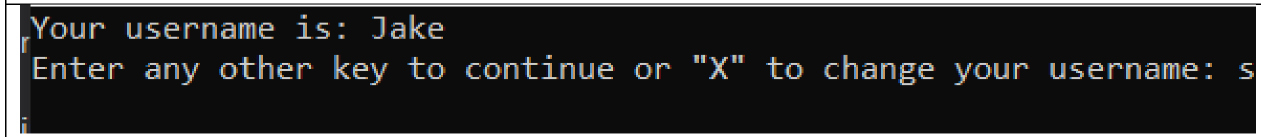
```
Enter the NUMBER OF QUESTIONS BETWEEN "8" and "10" or Enter "X" to change the difficulty you want: 8
```

The user is then prompted to enter the number of questions between 8 and 10; or once again enter X should the user not be content with the previous input.



```
Enter "YOUR USERNAME" or "X" to change how many questions you want: Jake
```

Once the defining characteristics of the quiz has been gathered, each player will then be prompted to enter their own unique username. Like before, users have the option to rollback by entering X.

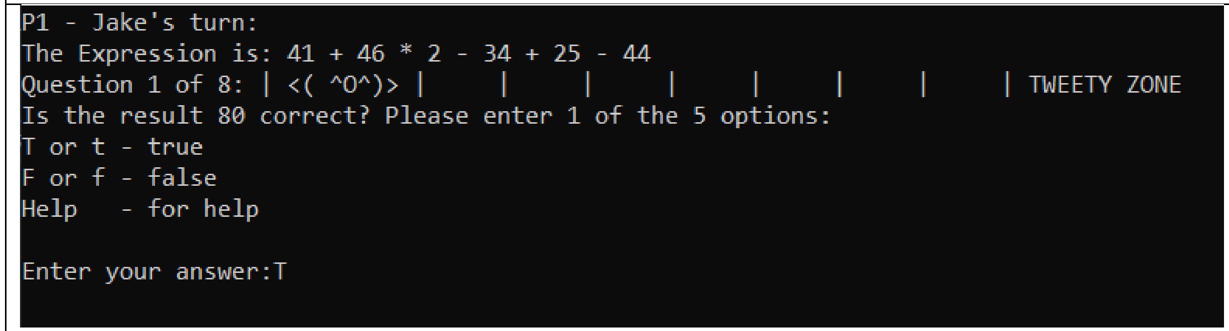


```
Your username is: Jake
Enter any other key to continue or "X" to change your username: s
```

Users are then requested to confirm their username, whilst having the option to go back and change it, once again by entering X.
Once all usernames have been confirmed, the game begins.

```
Generating Question...
```

This is the loading screen to inform the current player that their question is on its way. The loading time is based on the difficulty level - the greater the difficulty level, the shorter the loading times.



```
P1 - Jake's turn:
The Expression is: 41 + 46 * 2 - 34 + 25 - 44
Question 1 of 8: | <( ^O^)> |     |     |     |     |     |     |     | TWEETY ZONE
Is the result 80 correct? Please enter 1 of the 5 options:
T or t - true
F or f - false
Help   - for help

Enter your answer:T
```

Line 1 – this represents the current player's turn.
Line 2 – this is our randomly generated expression, based on the difficulty selected. To keep our program user-friendly, every number generated is an integer and every result of any randomly generated expression is an integer.
Line 3 – this line represents our quiz's completion accumulator, depicting the current progression of the player(s). As the question progresses, so does the avatar (i.e. TWEETY).

Line 4 – there are four possible ways of generating a question to ask:

- We start of by calculating the correct answer to the expression.
- We then calculate an incorrect answer within the range of the correct answer, based on the difficulty, whilst making sure that the randomly generated incorrect answer is not the same as the correct answer. The higher the difficulty, the closer our incorrect answers are generated to the correct answer – this adds to the deceptiveness of our question, as our incorrect answers look similar to the correct one.
- We then randomly pick whether we work with the correct or incorrect answer in our question.
- Lastly, we decide whether we want to attach the word *correct* or *incorrect* at the end of our question.

This is how we provide the user four possible ways of being asked a question – this provides variety in our program and thus reduces the monotonous styles of modern-day quizzes.

Example: 40 + 23 * 5 + 32 * 3 + 30

- The correct answer is 281.
- Since the difficulty level is 2 (HARD), the range of our randomly generated Incorrect answer is as follows: 281 – 2 ≤ **incorrect answer** ≤ 281 + 2 ➔ 279 ≤ **incorrect answer** ≤ 283.
- In this case, thanks to the random factor, we will be working with the correct answer instead of the incorrect answer in our statement – i.e. 281.
- There is a 50% chance of attaching *correct* or *incorrect*, however in this case, the odds favour the word *correct.*

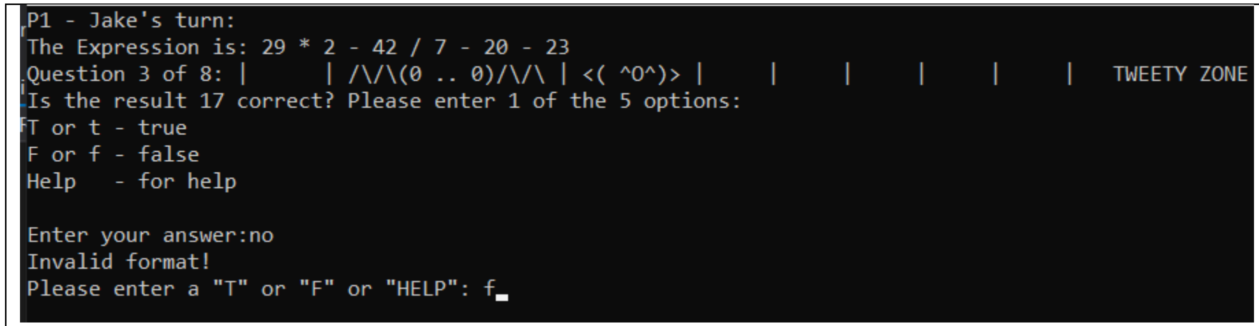
*Note:

If the difficulty level were 0 (EASY), the incorrect answer would lie in the following range:

n - 10 ≤ **incorrect answer** ≤ n + 10, in this example, 271 ≤ **incorrect answer** ≤ 291, where n is the correct answer. This makes it easier for the player to tell that the answer is not correct – should the incorrect answer have been used in the question.

If the difficulty level were 1 (MEDIUM), the incorrect answer would lie in the following range:

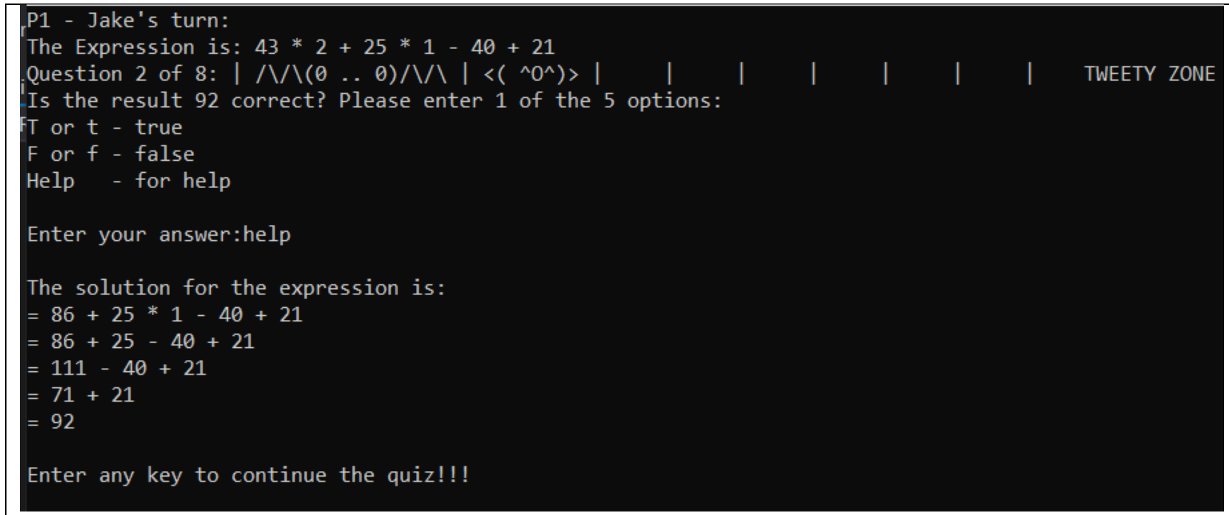
n - 5 ≤ **incorrect answer** ≤ n + 5, in this example 276 ≤ **incorrect answer** ≤ 286, where n is the correct answer. This makes it moderately difficult for the player to tell that the answer is not correct – should the incorrect answer have been used in the question.



```
P1 - Jake's turn:
The Expression is: 29 * 2 - 42 / 7 - 20 - 23
Question 3 of 8: |      | /\/\(0 .. 0)/\/\ | <( ^O^)> |     |     |     |     |     |   TWEETY ZONE
Is the result 17 correct? Please enter 1 of the 5 options:
T or t - true
F or f - false
Help   - for help

Enter your answer:no
Invalid format!
Please enter a "T" or "F" or "HELP": f
```

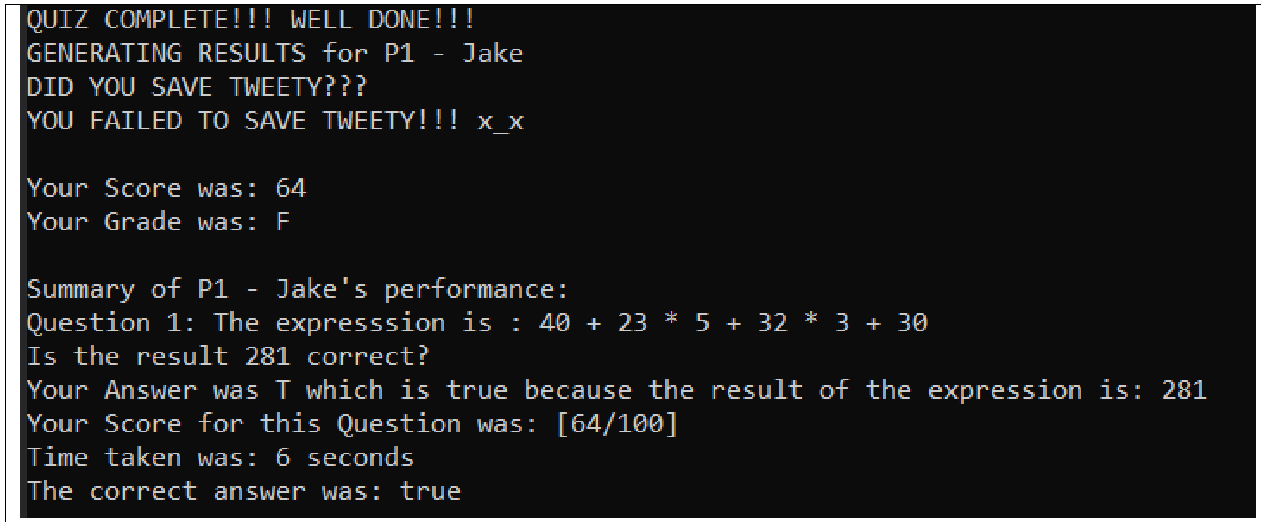
Lines 5 to 8 - for the following lines, the player is presented with the choice of acceptable answers. If the player enters any other character besides those presented to him, we display that it was an invalid format and ask the user to re-enter their choice until it meets the acceptable format. i.e. it is 1 of the 5 mentioned options.

Once the user enters a valid answer, their score is calculated. The players character also moves along the progress bar as an indication as a graphical representation of how many more questions the user must answer. The users' character is also chased by a villain dependant on the category they have chosen. This is for story purposes.



```
P1 - Jake's turn:
The Expression is: 43 * 2 + 25 * 1 - 40 + 21
Question 2 of 8: | /\/\(0 .. 0)/\/\ | <( ^O^)> |     |     |     |     |     |    TWEETY ZONE
Is the result 92 correct? Please enter 1 of the 5 options:
T or t - true
F or f - false
Help   - for help

Enter your answer:help

The solution for the expression is:
= 86 + 25 * 1 - 40 + 21
= 86 + 25 - 40 + 21
= 111 - 40 + 21
= 71 + 21
= 92

Enter any key to continue the quiz!!!
```

If the player decides they need help for the question they enter the word help. The steps in the solution are displayed for the user to see. The users do not gain any points if they enter the word help or get the question wrong.



```
QUIZ COMPLETE!!! WELL DONE!!!
GENERATING RESULTS for P1 - Jake
DID YOU SAVE TWEETY???
YOU FAILED TO SAVE TWEETY!!! x_x

Your Score was: 64
Your Grade was: F

Summary of P1 - Jake's performance:
Question 1: The expresssion is : 40 + 23 * 5 + 32 * 3 + 30
Is the result 281 correct?
Your Answer was T which is true because the result of the expression is: 281
Your Score for this Question was: [64/100]
Time taken was: 6 seconds
The correct answer was: true
```

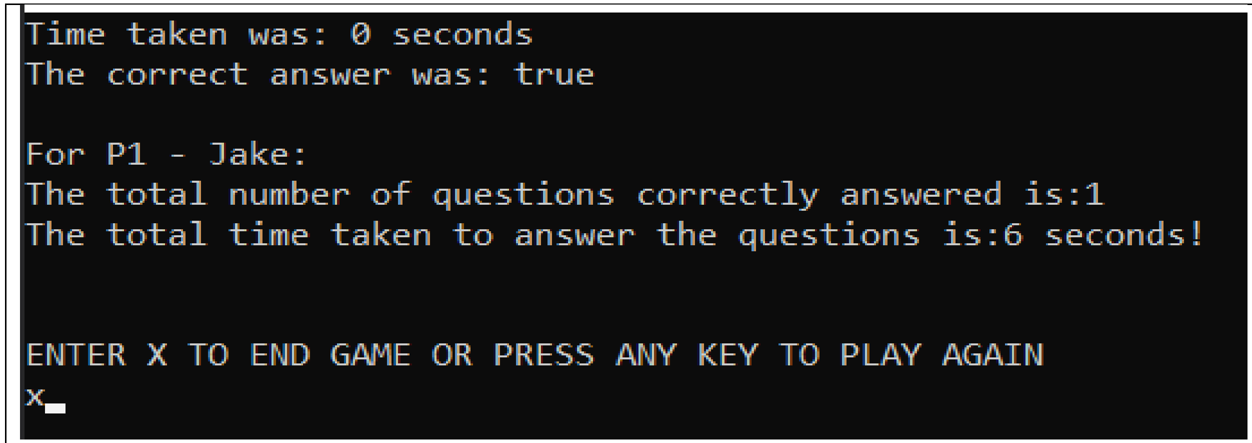
Line 1 and 2 – Indicates that the user is done answering the questions and their results are on the way.
Line 3 – the aim of the game was to save the players character from the villain based on difficulty.
Line 4 – the program sleeps for two seconds to add suspense to the moment when the user finds out if they saved their character. This is done based of the users score. If the user gets at least 50% of the questions right, their character is saved, if not then we display that the user had not saved their character

Line 6 – the program once again sleeps for 3 seconds before displaying the users score. This is done once again to create suspense for the user
Line 7 – Program sleeps again before displaying the users' grade for the quiz. Their grade is dependent on the players score divided by (the number of questions they chose multiplied by 100)

- If player gets below 50% of the questions wrong the grade is an F
- If the players get between 50%(inclusive) and 60% of the questions right their grade is a D
- If the players get between 60%(inclusive) and760% of the questions right their grade is a C

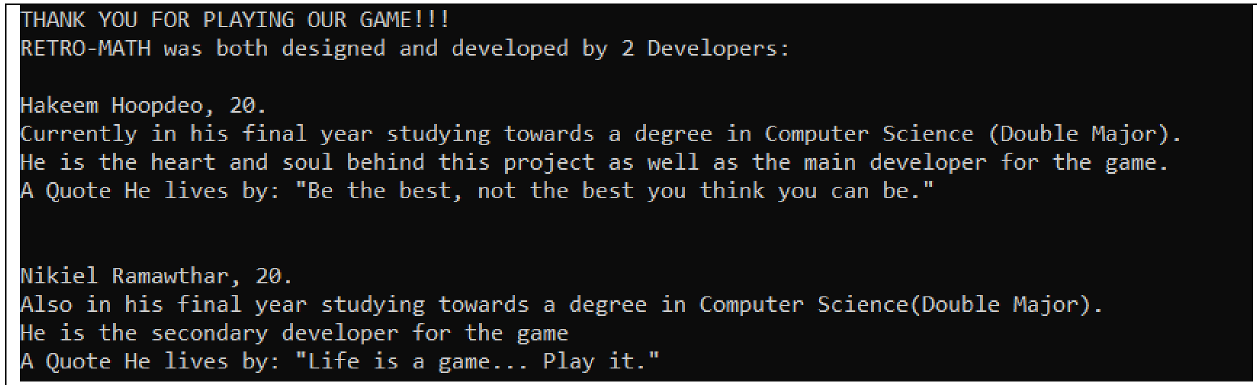
- If the players get between 70% (inclusive) and 80% of the questions right their grade is a B
- If the players get 80% and over of the questions right their grade is an A

After this a player review is displayed to the user depicting:
Line 1 after summary - The question number and the expression associated to the question.
Line 2 after summary - The question presented to the user to answer.
Line 3 after summary - The players answer choice, and whether their answer was correct or incorrect. And explanation is also given as to why their answer is correct or incorrect. Line is skipped if the player entered help.
Line 4 after summary - The players score for this question. Since the players score is dependent on the time taken to answer the question, it is possible that a player could get the answer correct but take up too much of time answering. If the player has 50/100 for the question it means he/she had gone over the time limit given for each question.
Line 5 after summary - The time the user took to answer the question.
Line 6 after summary - What was the answer we were expecting for the question



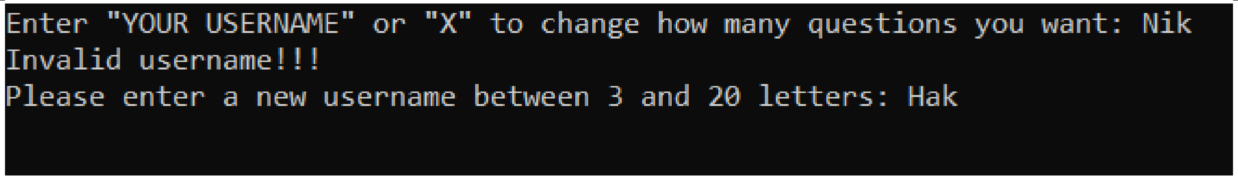
```
Time taken was: 0 seconds
The correct answer was: true

For P1 - Jake:
The total number of questions correctly answered is:1
The total time taken to answer the questions is:6 seconds!


ENTER X TO END GAME OR PRESS ANY KEY TO PLAY AGAIN
x
```

After all the details for each question in the review is displayed. The players total time and the number of questions he/she had answered correctly is displayed. At this point the game is completed, the player has the choice of playing the game again or display the credits (by entering x). If the player chooses to play again, they are taken back to the character selection screen and will go through each process again. All attributes are reset or cleared to prevent an overlap in detail.



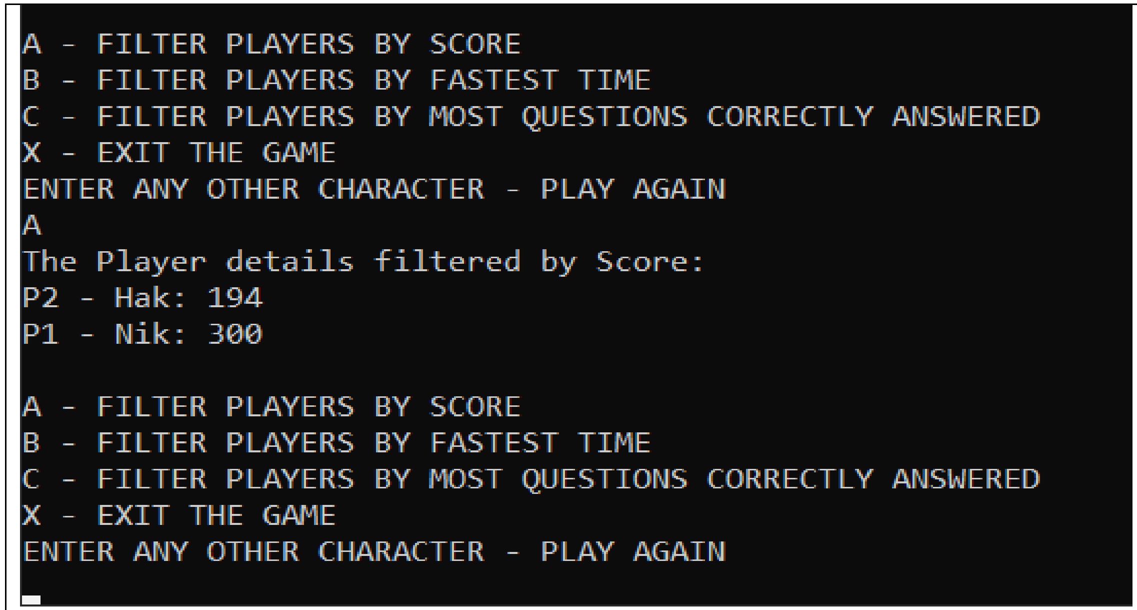
```
THANK YOU FOR PLAYING OUR GAME!!!
RETRO-MATH was both designed and developed by 2 Developers:

Hakeem Hoopdeo, 20.
Currently in his final year studying towards a degree in Computer Science (Double Major).
He is the heart and soul behind this project as well as the main developer for the game.
A Quote He lives by: "Be the best, not the best you think you can be."


Nikiel Ramawthar, 20.
Also in his final year studying towards a degree in Computer Science(Double Major).
He is the secondary developer for the game
A Quote He lives by: "Life is a game... Play it."
```

If the player enters x to end game the credits roll and the game is completed



```
Enter "YOUR USERNAME" or "X" to change how many questions you want: Nik
Invalid username!!!
Please enter a new username between 3 and 20 letters: Hak
```

In multiplayer. Two people cannot have the same username. If 2 players enter the same username, the second player would be prompted to change their username



```
A - FILTER PLAYERS BY SCORE
B - FILTER PLAYERS BY FASTEST TIME
C - FILTER PLAYERS BY MOST QUESTIONS CORRECTLY ANSWERED
X - EXIT THE GAME
ENTER ANY OTHER CHARACTER - PLAY AGAIN
A
The Player details filtered by Score:
P2 - Hak: 194
P1 - Nik: 300

A - FILTER PLAYERS BY SCORE
B - FILTER PLAYERS BY FASTEST TIME
C - FILTER PLAYERS BY MOST QUESTIONS CORRECTLY ANSWERED
X - EXIT THE GAME
ENTER ANY OTHER CHARACTER - PLAY AGAIN
```

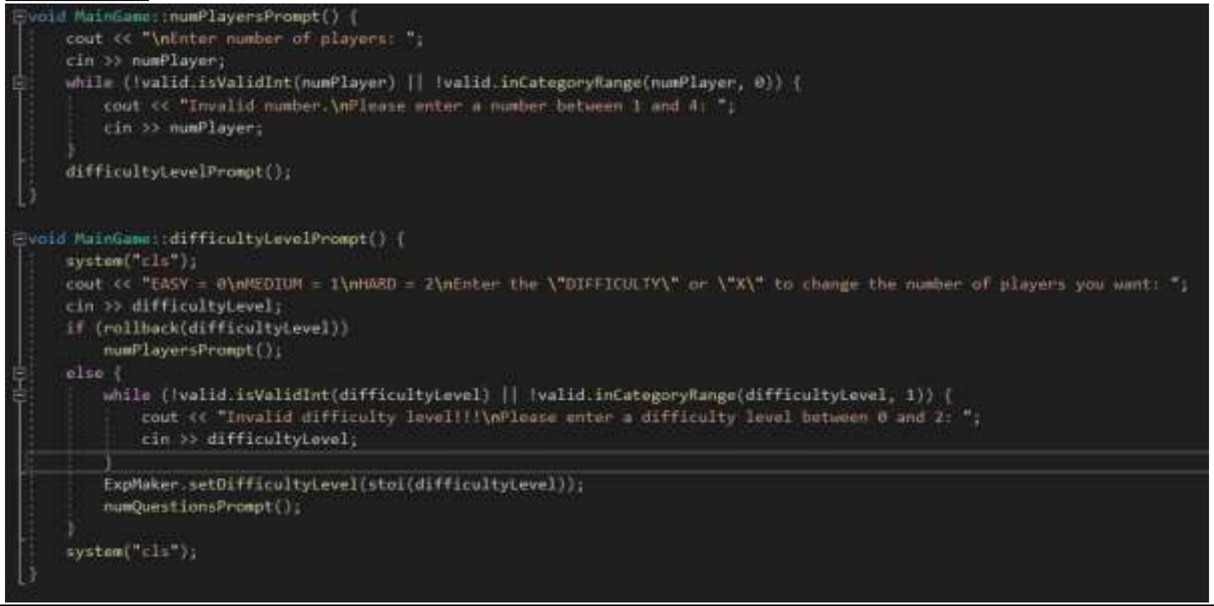
In multiplayer at the end of the player review. Players are presented with a list of options to display details about certain aspects in ascending order.

- If the player enters A/a - the player details are filtered by score to produce all players in order from those with the smallest score to the one with the highest. Displaying the player number, their name and the score associated with this player.
- If the player enters B/b - the player details are filtered by their time to produce all players in order from those with the fastest time to those with the slowest time. Displaying the player number, their name and the time associated with the player.
- If the player enters C/c – the player details are filtered by the number of questions answered correctly to produce all the players with the least number of questions correct to the most number of questions correct. Displaying the player number, their name and the number of correct questions associated to that player.
- If the player enters X/x – the game comes to an end and the credits are displayed.
- If the player enters any other character. All stats and information pertaining to the current game are cleared and the player has the option of playing the game again

Programming Techniques

1. Function

<u>Screenshot:</u>



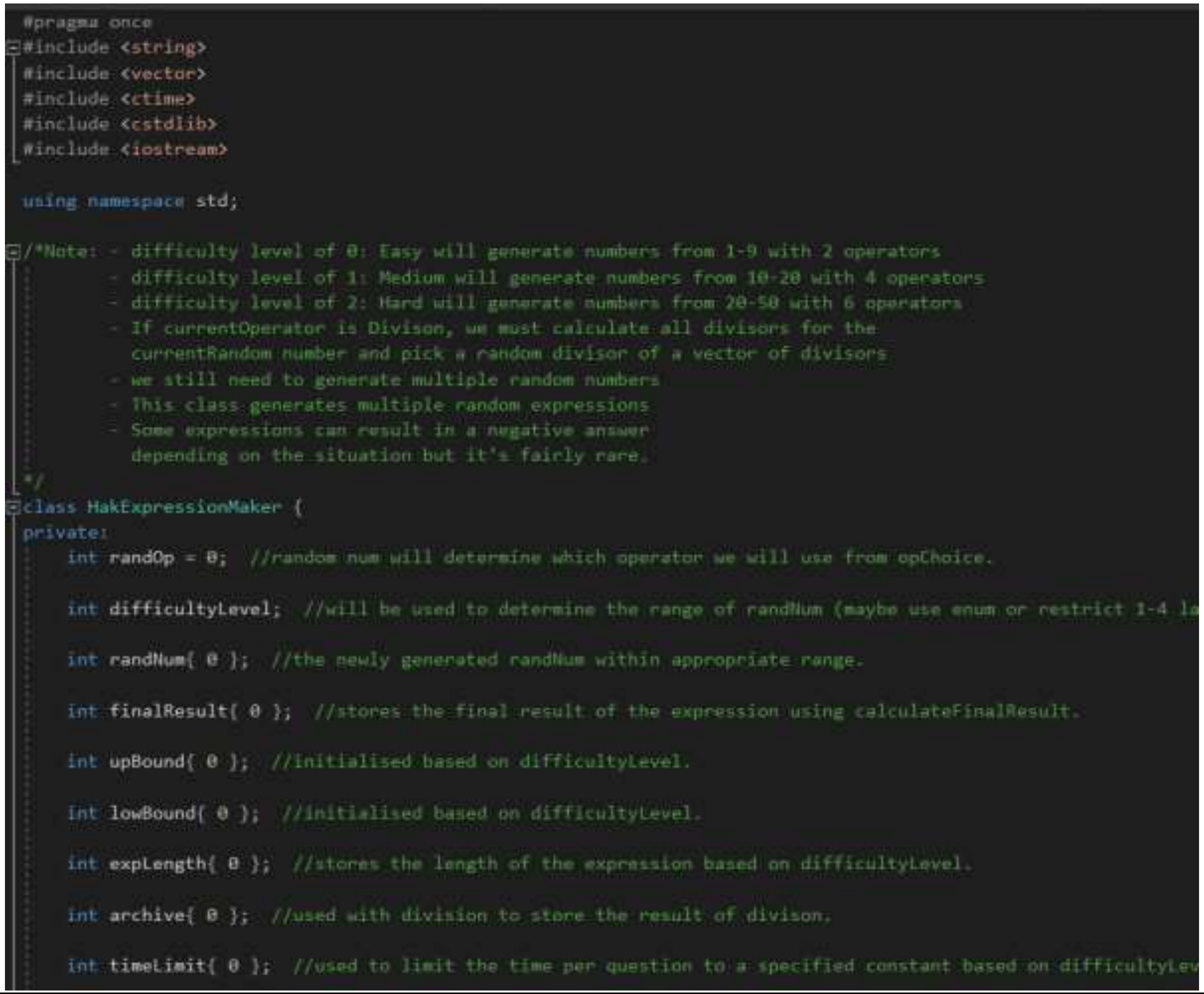
```cpp
void MainGame::numPlayersPrompt() {
    cout << "\nEnter number of players: ";
    cin >> numPlayer;
    while (!valid.isValidInt(numPlayer) || !valid.inCategoryRange(numPlayer, 0)) {
        cout << "Invalid number.\nPlease enter a number between 1 and 4: ";
        cin >> numPlayer;
    }
    difficultyLevelPrompt();
}

void MainGame::difficultyLevelPrompt() {
    system("cls");
    cout << "EASY = 0\nMEDIUM = 1\nHARD = 2\nEnter the \"DIFFICULTY\" or \"X\" to change the number of players you want: ";
    cin >> difficultyLevel;
    if (rollback(difficultyLevel))
        numPlayersPrompt();
    else {
        while (!valid.isValidInt(difficultyLevel) || !valid.inCategoryRange(difficultyLevel, 1)) {
            cout << "Invalid difficulty level!!!\nPlease enter a difficulty level between 0 and 2: ";
            cin >> difficultyLevel;
        }
        ExpMaker.setDifficultyLevel(stoi(difficultyLevel));
        numQuestionsPrompt();
    }
    system("cls");
}
```

<u>Motivation:</u>

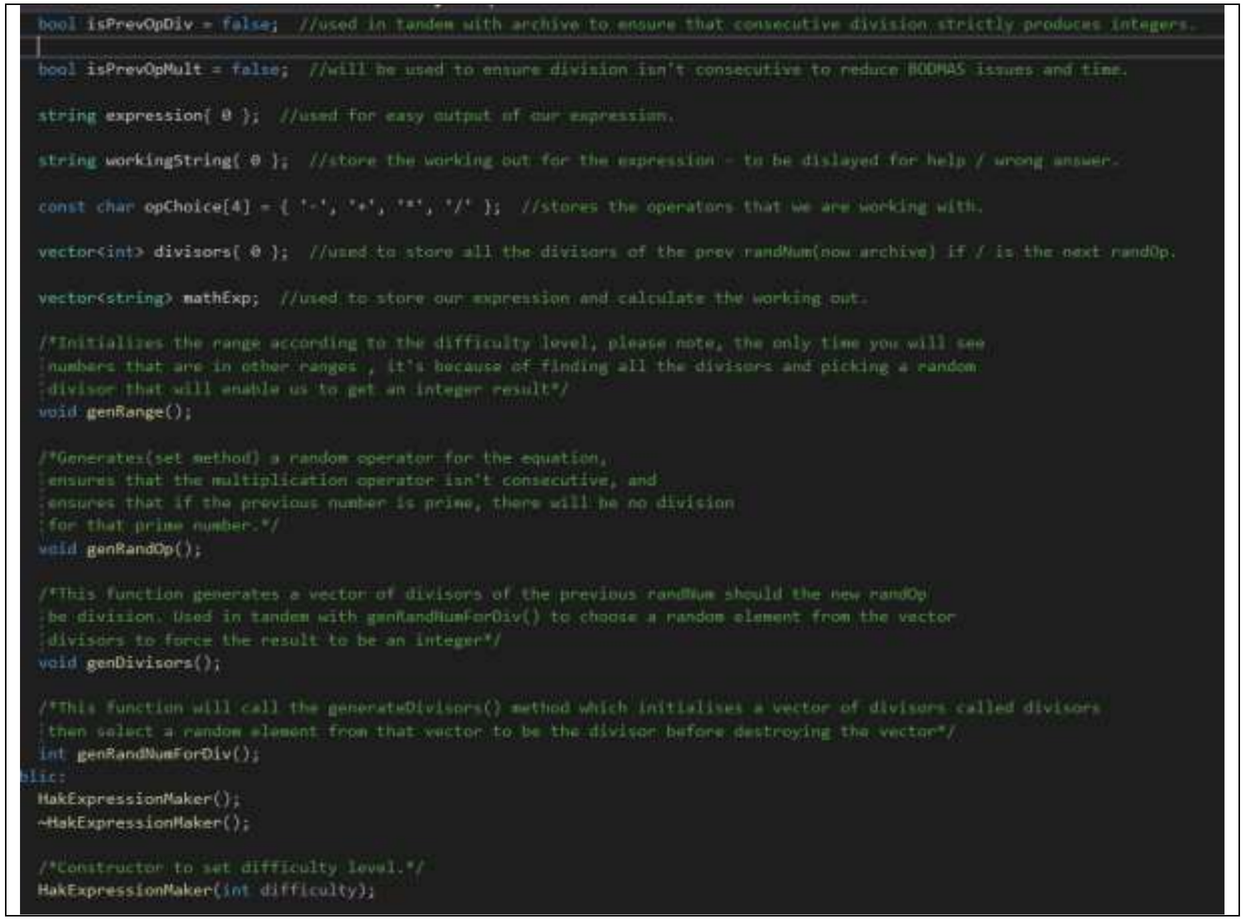
We had opted for using functions since we developed a rollback feature. With rollback if a player would like to change details or information pertaining to the quiz, the option is available to them. For example, if a player decides they are not happy with their choice of Difficulty or the username they chose, they will be allowed to change them. Typically, you would have to write consecutive while loops and write redundant code to make a feature like this work. With the use of functions, we would just need to call a single statement. Breaking up our code into smaller modules showed great success when it came to the debugging part of the process.

2. Class

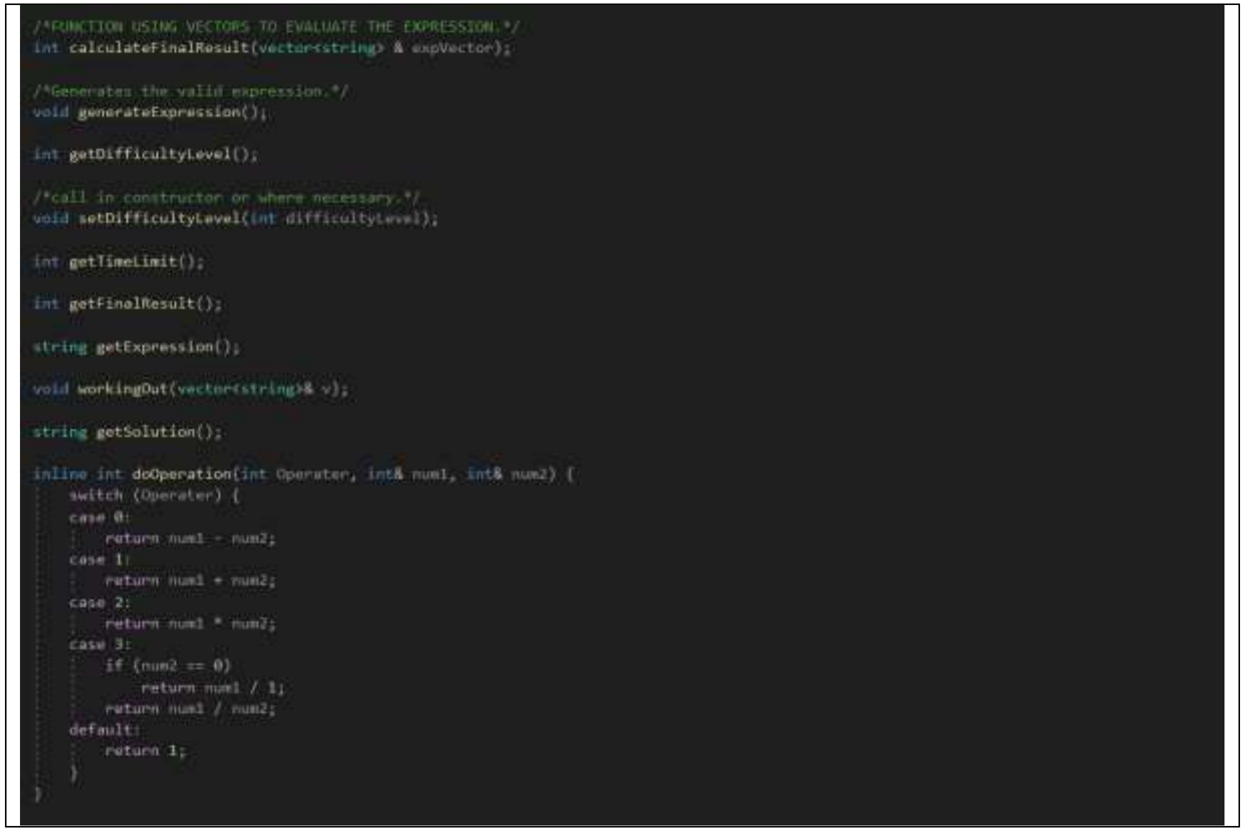
Screenshot:



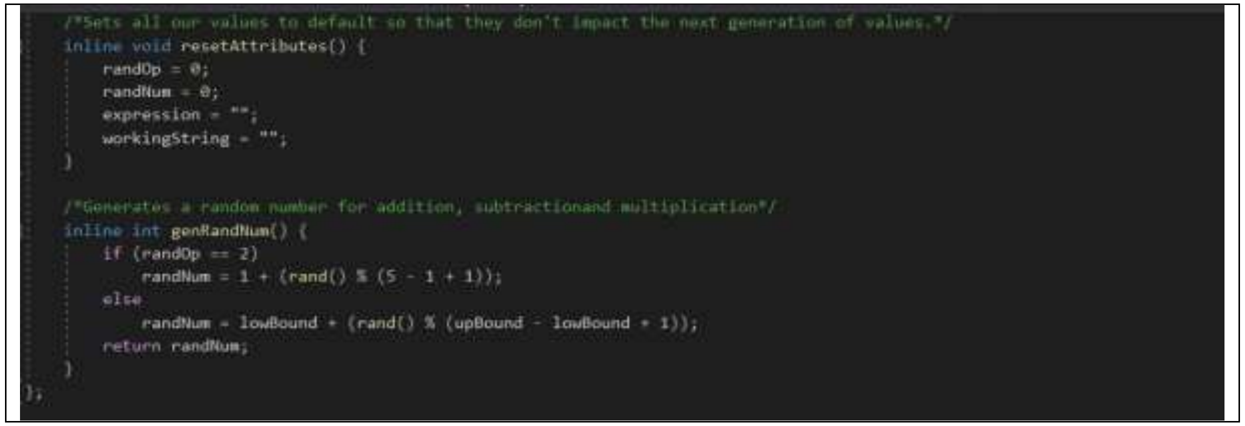
```cpp
#pragma once
#include <string>
#include <vector>
#include <ctime>
#include <cstdlib>
#include <iostream>

using namespace std;

/*Note: - difficulty level of 0: Easy will generate numbers from 1-9 with 2 operators
        - difficulty level of 1: Medium will generate numbers from 10-20 with 4 operators
        - difficulty level of 2: Hard will generate numbers from 20-50 with 6 operators
        - If currentOperator is Divison, we must calculate all divisors for the
          currentRandom number and pick a random divisor of a vector of divisors
        - we still need to generate multiple random numbers
        - This class generates multiple random expressions
        - Some expressions can result in a negative answer
          depending on the situation but it's fairly rare.
*/
class HakExpressionMaker {
private:
    int randOp = 0;  //random num will determine which operator we will use from opChoice.

    int difficultyLevel;  //will be used to determine the range of randNum (maybe use enum or restrict 1-4 la

    int randNum{ 0 };  //the newly generated randNum within appropriate range.

    int finalResult{ 0 };  //stores the final result of the expression using calculateFinalResult.

    int upBound{ 0 };  //initialised based on difficultyLevel.

    int lowBound{ 0 };  //initialised based on difficultyLevel.

    int expLength{ 0 };  //stores the length of the expression based on difficultyLevel.

    int archive{ 0 };  //used with division to store the result of divison.

    int timeLimit{ 0 };  //used to limit the time per question to a specified constant based on difficultyLev
```



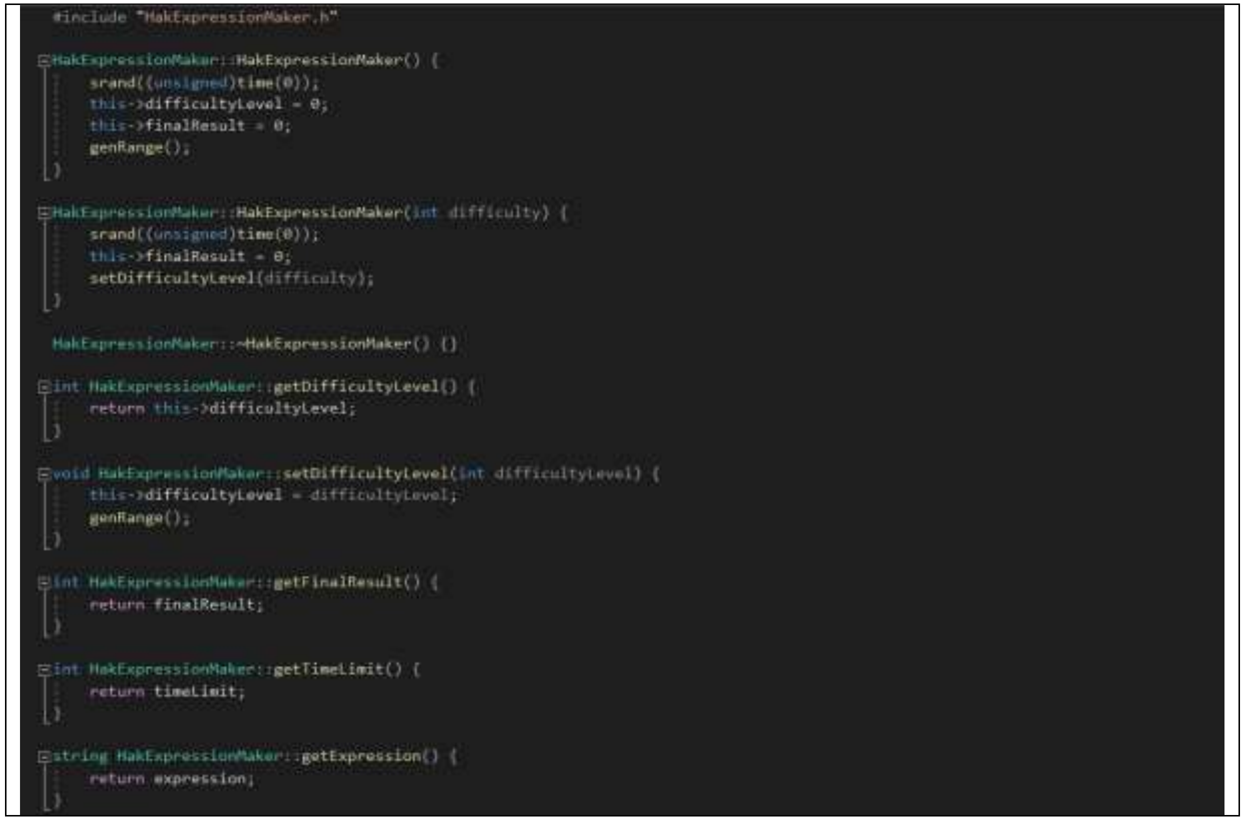
```
  bool isPrevOpDiv = false;  //used in tandem with archive to ensure that consecutive division strictly produces integers.

  bool isPrevOpMult = false;  //will be used to ensure division isn't consecutive to reduce BODMAS issues and time.

  string expression{ 0 };  //used for easy output of our expression.

  string workingString{ 0 };  //store the working out for the expression - to be dislayed for help / wrong answer.

  const char opChoice[4] = { '-', '+', '*', '/' };  //stores the operators that we are working with.

  vector<int> divisors{ 0 };  //used to store all the divisors of the prev randNum(now archive) if / is the next randOp.

  vector<string> mathExp;  //used to store our expression and calculate the working out.

  /*Initializes the range according to the difficulty level, please note, the only time you will see
   numbers that are in other ranges , it's because of finding all the divisors and picking a random
   divisor that will enable us to get an integer result*/
  void genRange();

  /*Generates(set method) a random operator for the equation,
   ensures that the multiplication operator isn't consecutive, and
   ensures that if the previous number is prime, there will be no division
   for that prime number.*/
  void genRandOp();

  /*This function generates a vector of divisors of the previous randNum should the new randOp
   be division. Used in tandem with genRandNumForDiv() to choose a random element from the vector
   divisors to force the result to be an integer*/
  void genDivisors();

  /*This function will call the generateDivisors() method which initialises a vector of divisors called divisors
   then select a random element from that vector to be the divisor before destroying the vector*/
  int genRandNumForDiv();
blic:
  HakExpressionMaker();
  ~HakExpressionMaker();

  /*Constructor to set difficulty level.*/
  HakExpressionMaker(int difficulty);
```



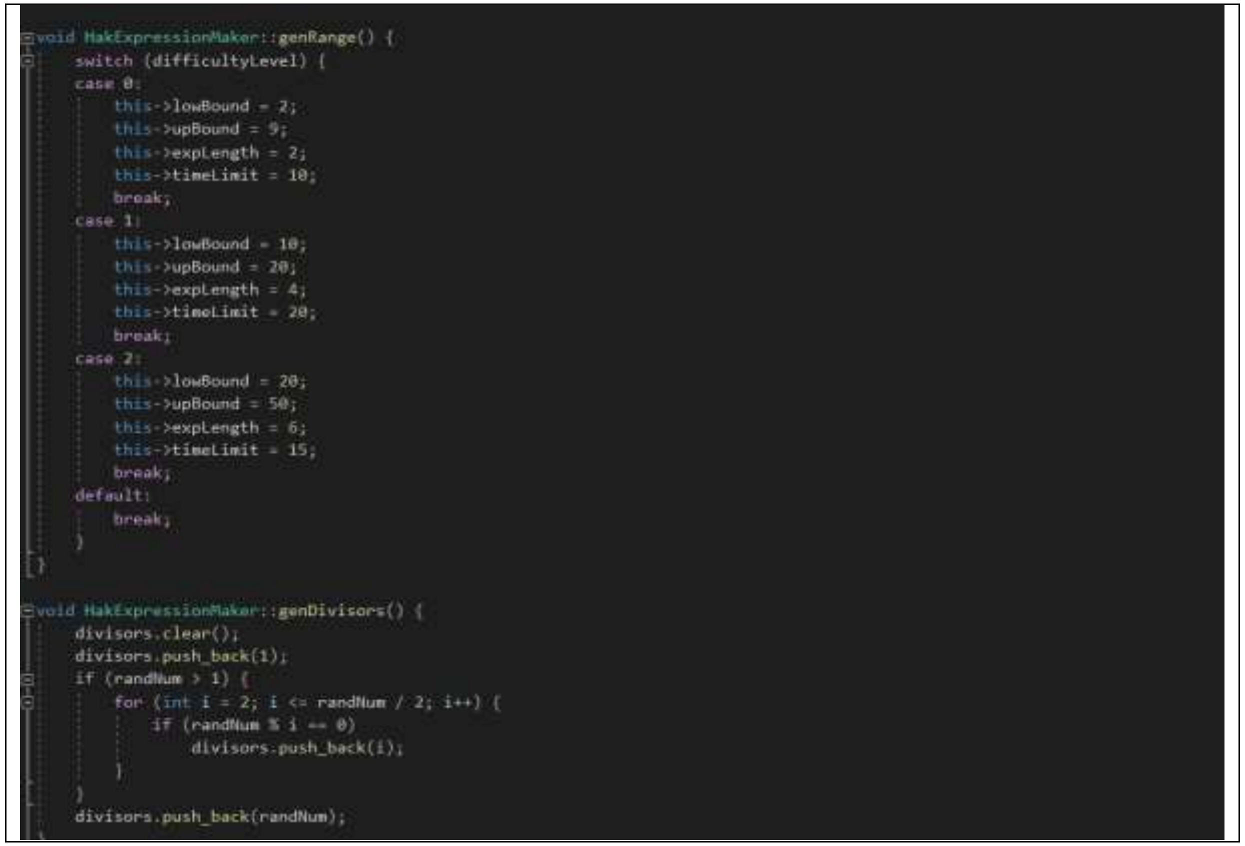
```
/*FUNCTION USING VECTORS TO EVALUATE THE EXPRESSION.*/
int calculateFinalResult(vector<string> & expVector);

/*Generates the valid expression.*/
void generateExpression();

int getDifficultyLevel();

/*call in constructor or where necessary.*/
void setDifficultyLevel(int difficultyLevel);

int getTimeLimit();

int getFinalResult();

string getExpression();

void workingOut(vector<string>& v);

string getSolution();

inline int doOperation(int Operater, int& num1, int& num2) {
    switch (Operater) {
    case 0:
        return num1 - num2;
    case 1:
        return num1 + num2;
    case 2:
        return num1 * num2;
    case 3:
        if (num2 == 0)
            return num1 / 1;
        return num1 / num2;
    default:
        return 1;
    }
}
```



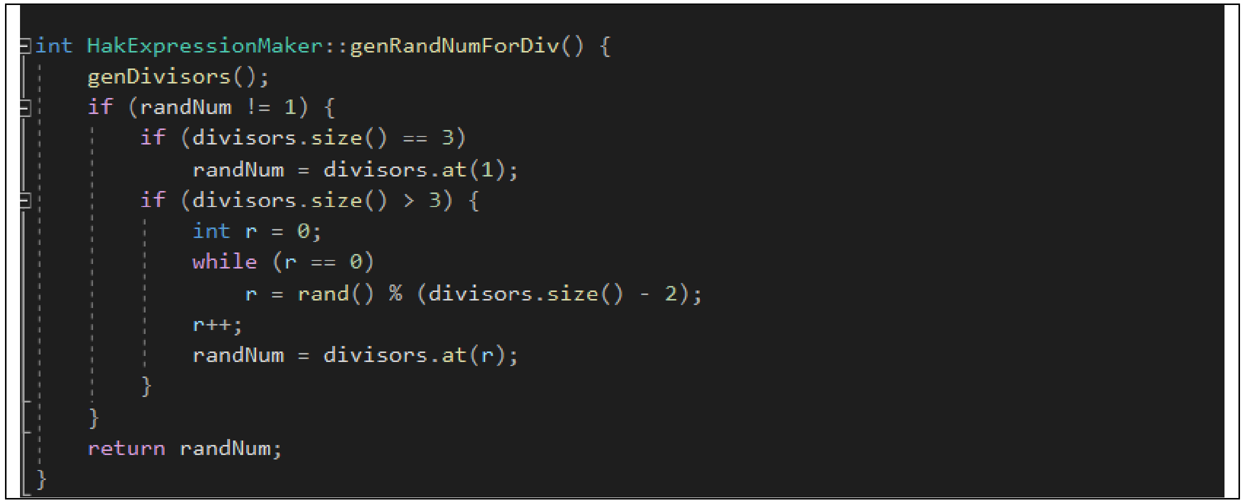
```
    /*Sets all our values to default so that they don't impact the next generation of values.*/
    inline void resetAttributes() {
        randOp = 0;
        randNum = 0;
        expression = "";
        workingString = "";
    }

    /*Generates a random number for addition, subtractionand multiplication*/
    inline int genRandNum() {
        if (randOp == 2)
            randNum = 1 + (rand() % (5 - 1 + 1));
        else
            randNum = lowBound + (rand() % (upBound - lowBound + 1));
        return randNum;
    }
};
```



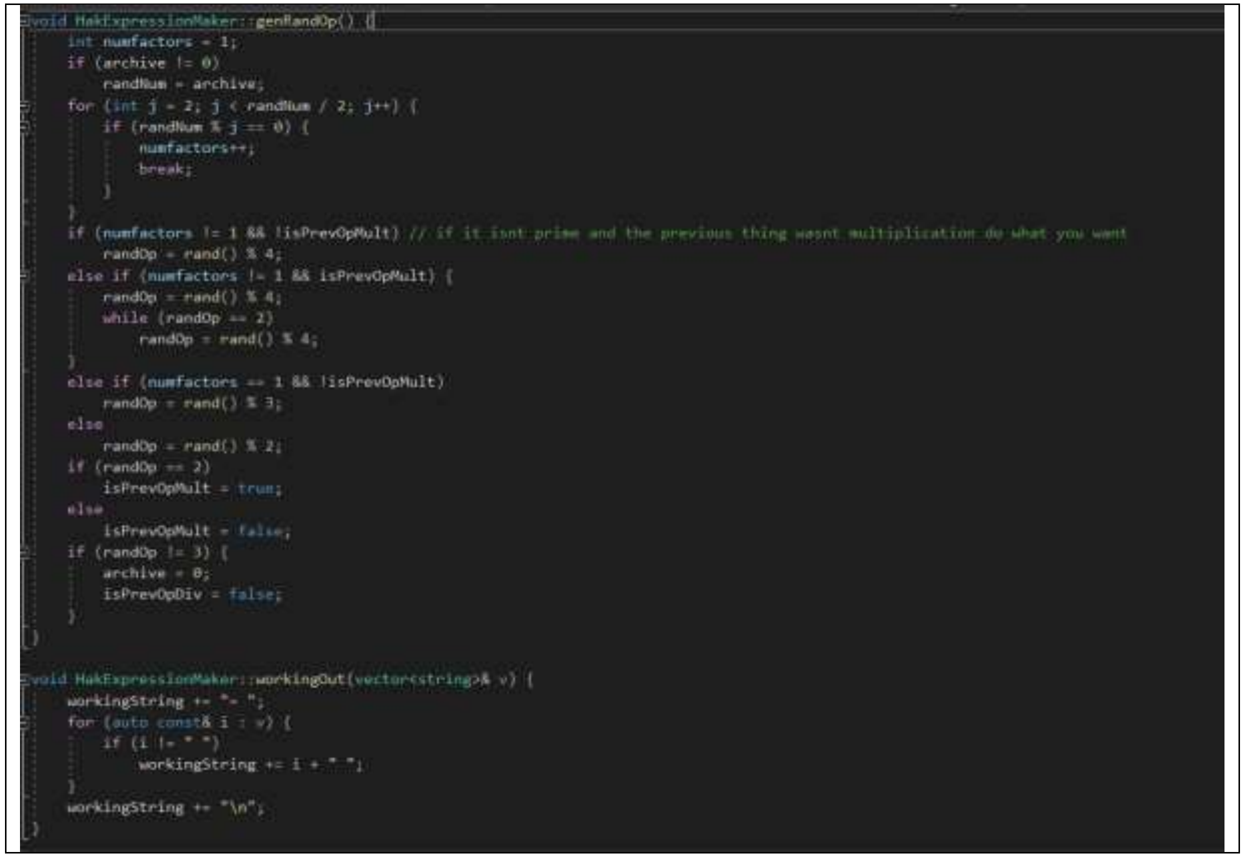
```cpp
#include "HakExpressionMaker.h"

HakExpressionMaker::HakExpressionMaker() {
    srand((unsigned)time(0));
    this->difficultyLevel = 0;
    this->finalResult = 0;
    genRange();
}

HakExpressionMaker::HakExpressionMaker(int difficulty) {
    srand((unsigned)time(0));
    this->finalResult = 0;
    setDifficultyLevel(difficulty);
}

HakExpressionMaker::~HakExpressionMaker() {}

int HakExpressionMaker::getDifficultyLevel() {
    return this->difficultyLevel;
}

void HakExpressionMaker::setDifficultyLevel(int difficultyLevel) {
    this->difficultyLevel = difficultyLevel;
    genRange();
}

int HakExpressionMaker::getFinalResult() {
    return finalResult;
}

int HakExpressionMaker::getTimeLimit() {
    return timeLimit;
}

string HakExpressionMaker::getExpression() {
    return expression;
}
```



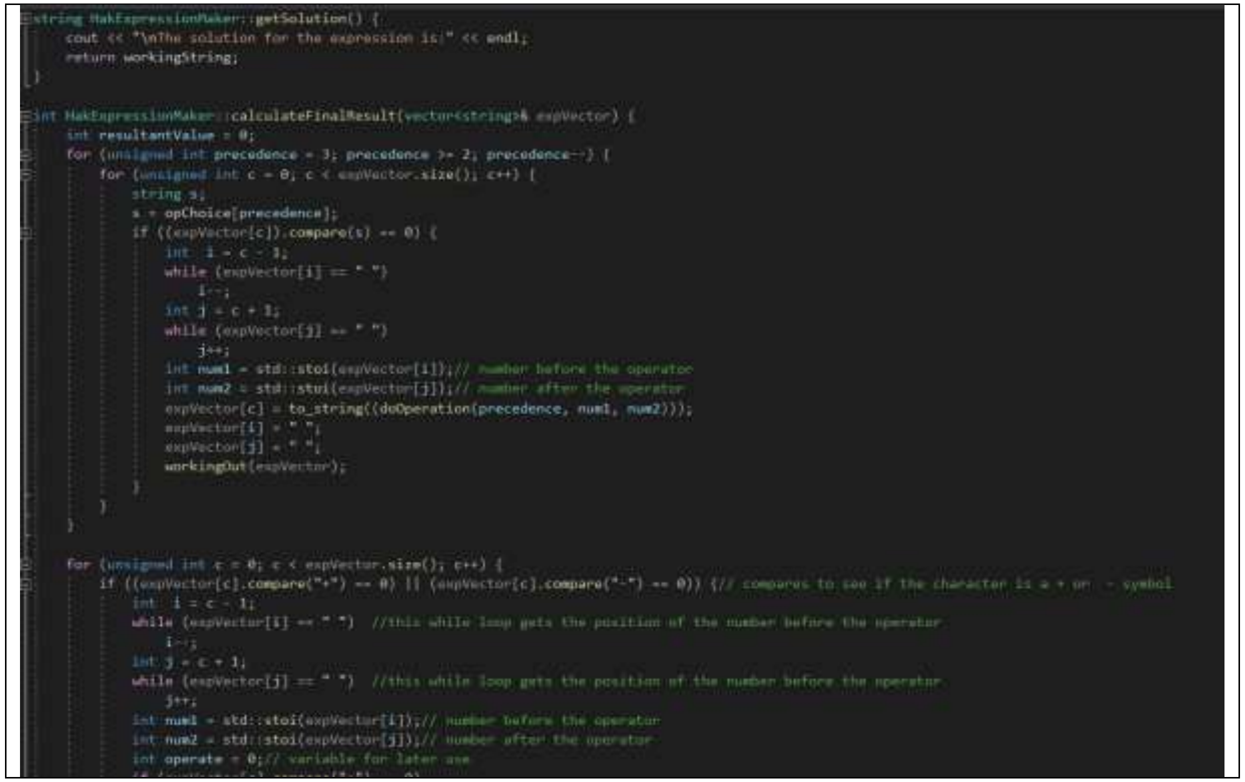
```cpp
void HakExpressionMaker::genRange() {
    switch (difficultyLevel) {
    case 0:
        this->lowBound = 2;
        this->upBound = 9;
        this->expLength = 2;
        this->timeLimit = 10;
        break;
    case 1:
        this->lowBound = 10;
        this->upBound = 20;
        this->expLength = 4;
        this->timeLimit = 20;
        break;
    case 2:
        this->lowBound = 20;
        this->upBound = 50;
        this->expLength = 6;
        this->timeLimit = 15;
        break;
    default:
        break;
    }
}

void HakExpressionMaker::genDivisors() {
    divisors.clear();
    divisors.push_back(1);
    if (randNum > 1) {
        for (int i = 2; i <= randNum / 2; i++) {
            if (randNum % i == 0)
                divisors.push_back(i);
        }
    }
    divisors.push_back(randNum);
```



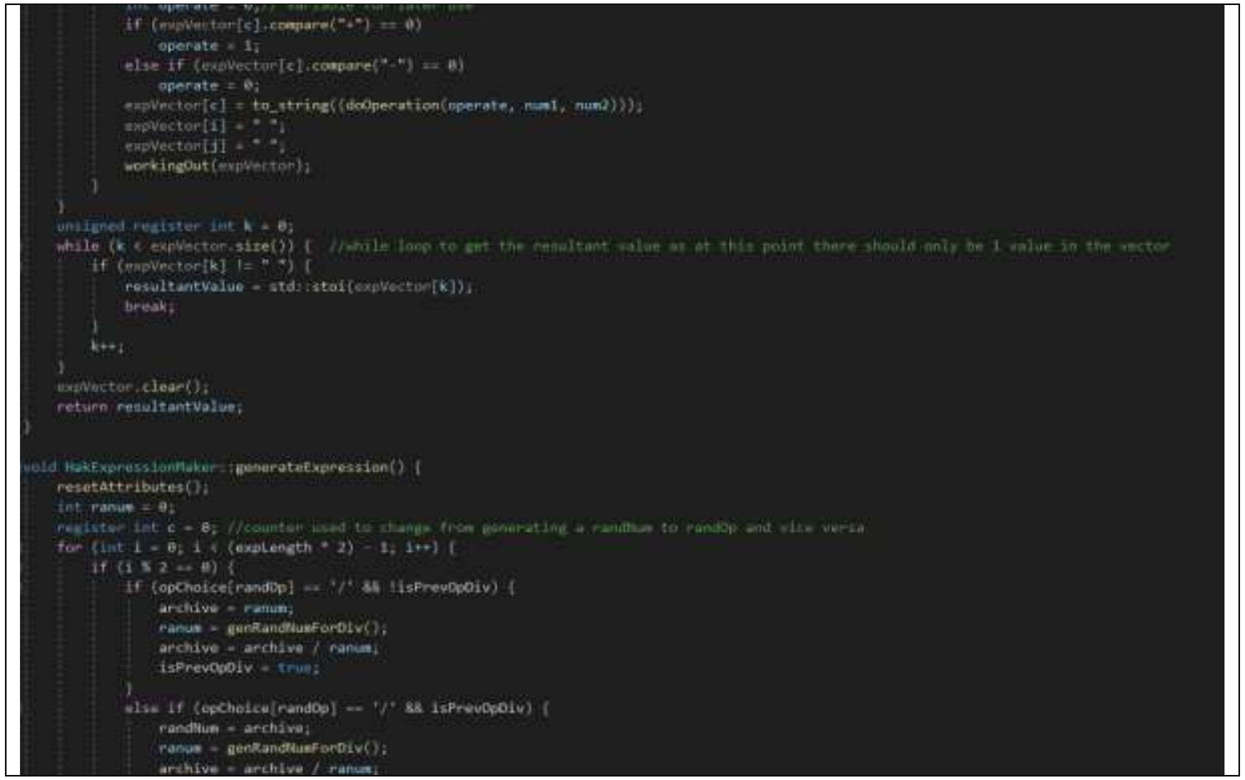
```cpp
int HakExpressionMaker::genRandNumForDiv() {
    genDivisors();
    if (randNum != 1) {
        if (divisors.size() == 3)
            randNum = divisors.at(1);
        if (divisors.size() > 3) {
            int r = 0;
            while (r == 0)
                r = rand() % (divisors.size() - 2);
            r++;
            randNum = divisors.at(r);
        }
    }
    return randNum;
}
```



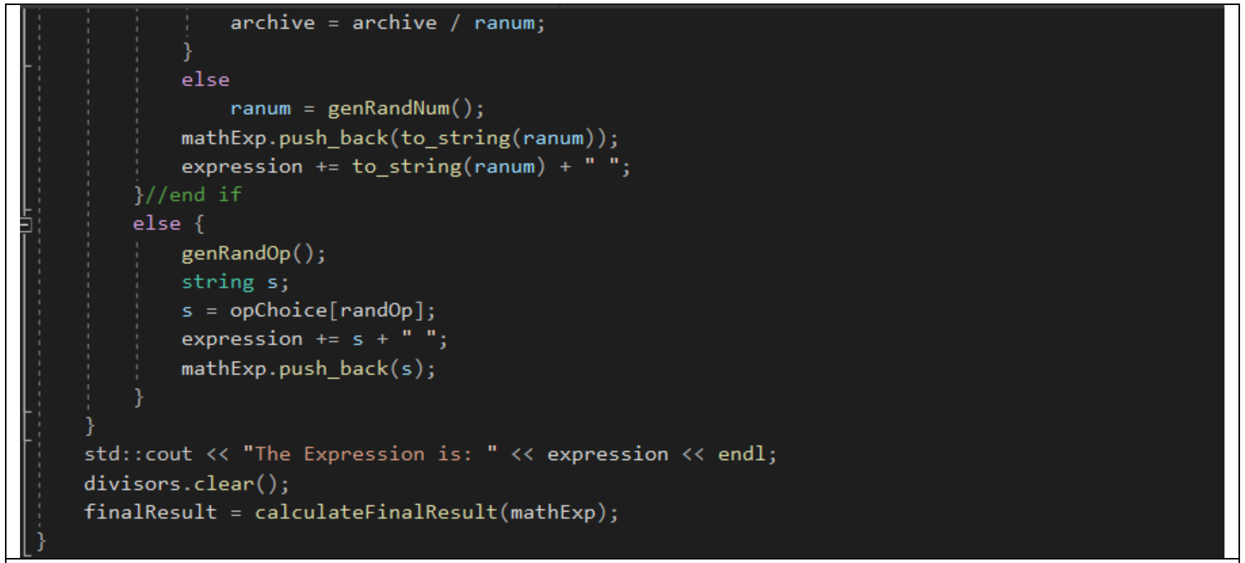
```
void MakExpressionMaker::genRandOp() {
    int numfactors = 1;
    if (archive != 0)
        randNum = archive;
    for (int j = 2; j < randNum / 2; j++) {
        if (randNum % j == 0) {
            numfactors++;
            break;
        }
    }
    if (numfactors != 1 && !isPrevOpMult) // if it isnt prime and the previous thing wasnt multiplication do what you want
        randOp = rand() % 4;
    else if (numfactors != 1 && isPrevOpMult) {
        randOp = rand() % 4;
        while (randOp == 2)
            randOp = rand() % 4;
    }
    else if (numfactors == 1 && !isPrevOpMult)
        randOp = rand() % 3;
    else
        randOp = rand() % 2;
    if (randOp == 2)
        isPrevOpMult = true;
    else
        isPrevOpMult = false;
    if (randOp != 3) {
        archive = 0;
        isPrevOpDiv = false;
    }
}

void MakExpressionMaker::workingOut(vector<string>& v) {
    workingString += "= ";
    for (auto const& i : v) {
        if (i != " ")
            workingString += i + " ";
    }
    workingString += "\n";
}
```



```
string MakExpressionMaker::getSolution() {
    cout << "\nThe solution for the expression is:" << endl;
    return workingString;
}

int MakExpressionMaker::calculateFinalResult(vector<string>& expVector) {
    int resultantValue = 0;
    for (unsigned int precedence = 3; precedence >= 2; precedence--) {
        for (unsigned int c = 0; c < expVector.size(); c++) {
            string s;
            s = opChoice[precedence];
            if ((expVector[c]).compare(s) == 0) {
                int  i = c - 1;
                while (expVector[i] == " ")
                    i--;
                int j = c + 1;
                while (expVector[j] == " ")
                    j++;
                int num1 = std::stoi(expVector[i]);// number before the operator
                int num2 = std::stoi(expVector[j]);// number after the operator
                expVector[c] = to_string((doOperation(precedence, num1, num2)));
                expVector[i] = " ";
                expVector[j] = " ";
                workingOut(expVector);
            }
        }
    }

    for (unsigned int c = 0; c < expVector.size(); c++) {
        if ((expVector[c].compare("+") == 0) || (expVector[c].compare("-") == 0)) {// compares to see if the character is a + or  - symbol
            int  i = c - 1;
            while (expVector[i] == " ")  //this while loop gets the position of the number before the operator
                i--;
            int j = c + 1;
            while (expVector[j] == " ")  //this while loop gets the position of the number before the operator
                j++;
            int num1 = std::stoi(expVector[i]);// number before the operator
            int num2 = std::stoi(expVector[j]);// number after the operator
            int operate = 0;// variable for later use
```



```cpp
            int operate = 0;// variable for later use
            if (expVector[c].compare("+") == 0)
                operate = 1;
            else if (expVector[c].compare("-") == 0)
                operate = 0;
            expVector[c] = to_string((doOperation(operate, num1, num2)));
            expVector[i] = " ";
            expVector[j] = " ";
            workingOut(expVector);
        }
    }
    unsigned register int k = 0;
    while (k < expVector.size()) {  //while loop to get the resultant value as at this point there should only be 1 value in the vector
        if (expVector[k] != " ") {
            resultantValue = std::stoi(expVector[k]);
            break;
        }
        k++;
    }
    expVector.clear();
    return resultantValue;
}

void MakExpressionMaker::generateExpression() {
    resetAttributes();
    int ranum = 0;
    register int c = 0; //counter used to change from generating a randNum to randOp and vice versa
    for (int i = 0; i < (expLength * 2) - 1; i++) {
        if (i % 2 == 0) {
            if (opChoice[randOp] == '/' && !isPrevOpDiv) {
                archive = ranum;
                ranum = genRandNumForDiv();
                archive = archive / ranum;
                isPrevOpDiv = true;
            }
            else if (opChoice[randOp] == '/' && isPrevOpDiv) {
                randNum = archive;
                ranum = genRandNumForDiv();
                archive = archive / ranum;
```



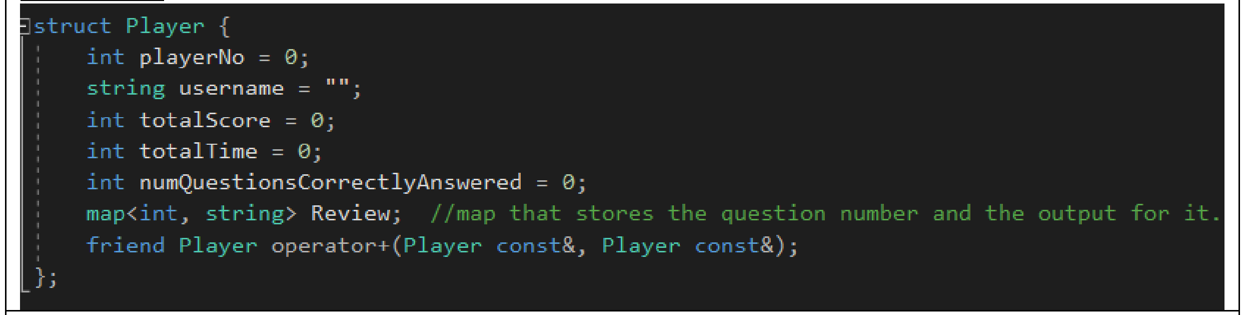
```
                archive = archive / ranum;
            }
            else
                ranum = genRandNum();
            mathExp.push_back(to_string(ranum));
            expression += to_string(ranum) + " ";
        }//end if
        else {
            genRandOp();
            string s;
            s = opChoice[randOp];
            expression += s + " ";
            mathExp.push_back(s);
        }
    }
    std::cout << "The Expression is: " << expression << endl;
    divisors.clear();
    finalResult = calculateFinalResult(mathExp);
}
```

Motivation:

Our focus for creating a class for generating expression was to have all the functions that pertain to the generation of an expression in a single file, making error analysis and debugging easier. The header file declares all the functions and variable that the class would need while the cpp file contains the actual implementation of the class. We are displaying and calculating different expressions for each user. While one would say why not just create a bunch of functions for the class containing the main game. As aforementioned, it makes debugging easier. It also helps in the scalability of code as newer and faster pieces of technology are developed every year. Choosing to replace a certain algorithm which a much more efficient one deems it perfect to use classes. It is also poor programming practice to be creating a program under a single file as this leads the way for sensitive data becoming exposed to users. i.e. Classes encapsulate data. We created a class for generating expression as well as a class to validate the various inputs of users. The use of classes also meant we could destroy the object after being used (using destructor) allowing for more memory to be accessible to other parts of the program.

3. Struct

Screenshot:



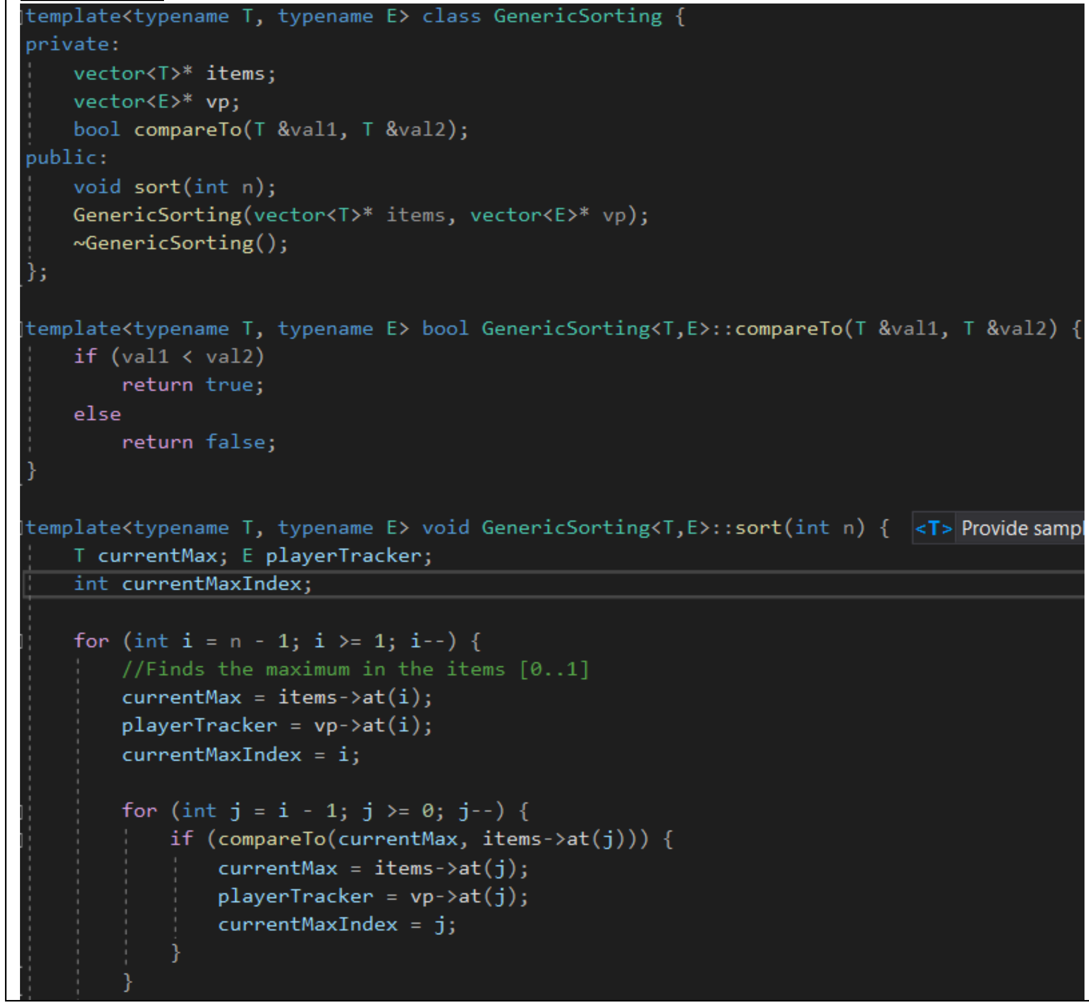
```
struct Player {
    int playerNo = 0;
    string username = "";
    int totalScore = 0;
    int totalTime = 0;
    int numQuestionsCorrectlyAnswered = 0;
    map<int, string> Review;  //map that stores the question number and the output for it.
    friend Player operator+(Player const&, Player const&);
};
```

Motivation:

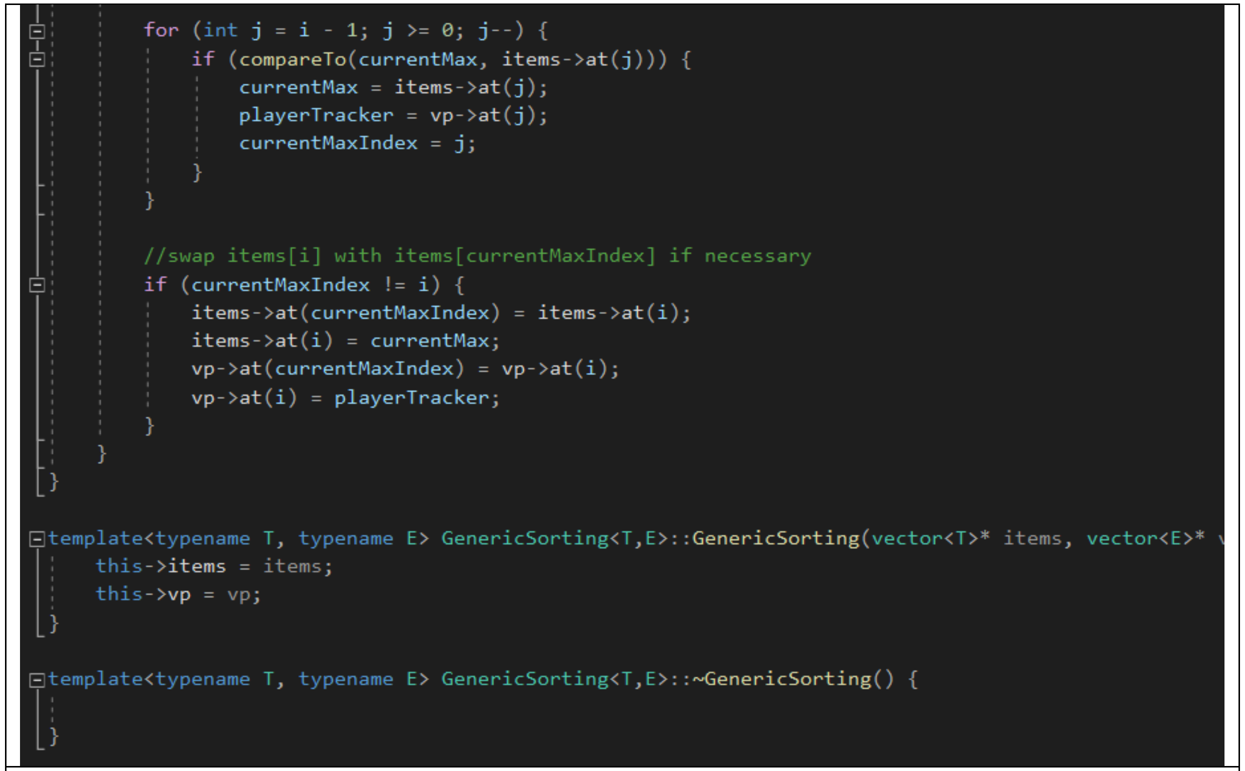
Our main objective for using the struct was to store details about the different players, because of this it was deemed highly unlikely to use a class, since we would only be retrieving and modifying the details about each player, as they continue playing the game. For instance, the players score is updated after each question they answer. To continuously be calling a method when we could just have direct access to the variable, meant that we would decrease the amount of CPU cycles it had taken. The player stats would also not be inherited nor would there be any subclasses of players. All members of the struct do not need to be initialized, for example. The map review which holds the details about the players questions do not need to be initialized when creating a player struct, since it is only called at the end of each question to store the players choice of answer and question type. Another example is our team struct, which would only hold values at the end of the quiz to combine player stats into an overall team stats, initialising the team structs at the beginning of the game would be a waste of memory space.

4. Pointer

Screenshot:



```
template<typename T, typename E> class GenericSorting {
private:
    vector<T>* items;
    vector<E>* vp;
    bool compareTo(T &val1, T &val2);
public:
    void sort(int n);
    GenericSorting(vector<T>* items, vector<E>* vp);
    ~GenericSorting();
};

template<typename T, typename E> bool GenericSorting<T,E>::compareTo(T &val1, T &val2) {
    if (val1 < val2)
        return true;
    else
        return false;
}

template<typename T, typename E> void GenericSorting<T,E>::sort(int n) {  <T> Provide sampl
    T currentMax; E playerTracker;
    int currentMaxIndex;

    for (int i = n - 1; i >= 1; i--) {
        //Finds the maximum in the items [0..1]
        currentMax = items->at(i);
        playerTracker = vp->at(i);
        currentMaxIndex = i;

        for (int j = i - 1; j >= 0; j--) {
            if (compareTo(currentMax, items->at(j))) {
                currentMax = items->at(j);
                playerTracker = vp->at(j);
                currentMaxIndex = j;
            }
        }
```



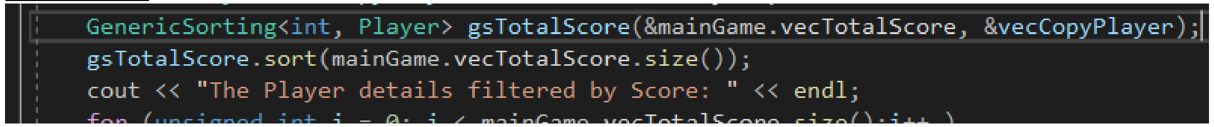
```
        for (int j = i - 1; j >= 0; j--) {
            if (compareTo(currentMax, items->at(j))) {
                currentMax = items->at(j);
                playerTracker = vp->at(j);
                currentMaxIndex = j;
            }
        }

        //swap items[i] with items[currentMaxIndex] if necessary
        if (currentMaxIndex != i) {
            items->at(currentMaxIndex) = items->at(i);
            items->at(i) = currentMax;
            vp->at(currentMaxIndex) = vp->at(i);
            vp->at(i) = playerTracker;
        }
    }
}

template<typename T, typename E> GenericSorting<T,E>::GenericSorting(vector<T>* items, vector<E>*
    this->items = items;
    this->vp = vp;
}

template<typename T, typename E> GenericSorting<T,E>::~GenericSorting() {

}
```

Motivation:

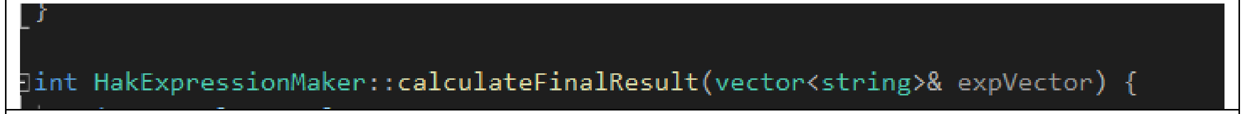
Pointers store the address of the memory location. This is useful mainly when working with data structures as sending a copy of the whole data structure or even a reference to it, would take up large amounts of space. Pointers also make a call by reference sufficiently easier than having to use references. In the case of the above screenshot. In the GenericSorting class we sort vectors in ascending order, we would like the changes to be made directly to the vector that was sent to the class. The use of pointers also improves performance for repetitive operations. In our case we are traversing through a vector and sorting it based on what the user decides to sort. The user can choose to sort out various attributes and is not restricted to only sorting out a single attribute during each run

5. Reference

Screenshot:



```
    GenericSorting<int, Player> gsTotalScore(&mainGame.vecTotalScore, &vecCopyPlayer);
    gsTotalScore.sort(mainGame.vecTotalScore.size());
    cout << "The Player details filtered by Score: " << endl;
```



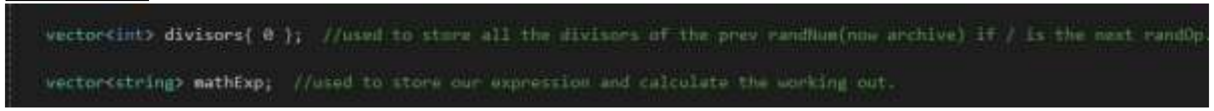
```
}

int HakExpressionMaker::calculateFinalResult(vector<string>& expVector) {
```

Motivation:

Referencing helps mainly when you send a function some parameter and want to keep changes you do to the object. We have a function which updates the score for player. With referencing whatever changes we make towards the score variable inside the function the same changes happen to the variable being sent, since with referencing we send the location in memory of the object. With the use of pointers comes the use of referencing. Since a pointer points to an address in memory and a reference returns the location of the object in memory.

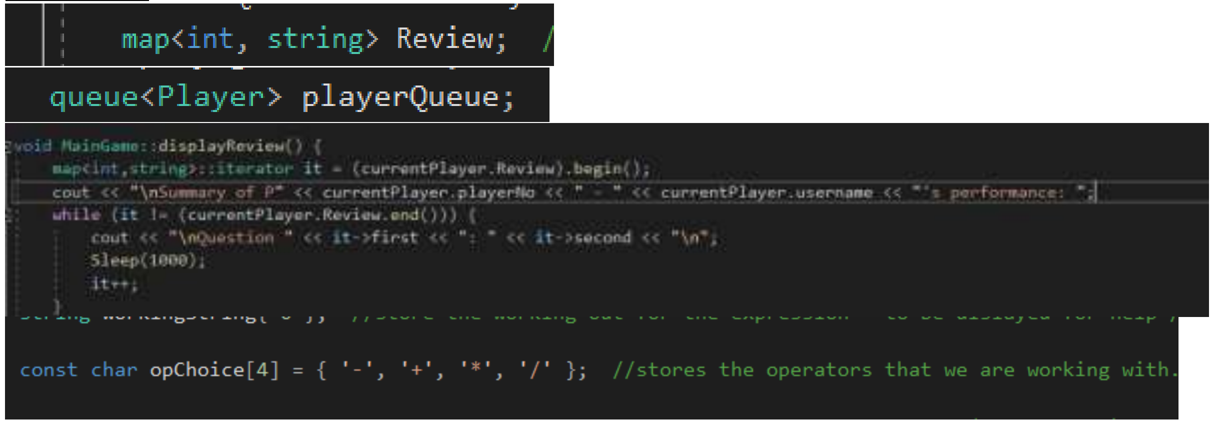
In the calculateFinalresult method, we pass by value a vector containing the expression for calculation. In this function, we delete and append new values to the expVector as well as display the contents of the vector after each calculation (used as a help feature when players do not know how to go about solving the expression). It would be a waste of memory to be sending a copy of the vector, since it will be overridden, for the next question.

6. Vector

Screenshot:
 `vector<int> divisors{ 0 };  //used to store all the divisors of the prev randNum(now archive) if / is the next randOp.` `vector<string> mathExp;  //used to store our expression and calculate the working out.`
Motivation: We decided to stick with vectors over arrays since the size of the array has to be hard coded. And is fixed at run time. We cannot specify the size we want from the divisors vector. Which is used to hold all numbers than can divide a random number generated. What if a prime number gets generate at one point and a composite number at another point in time? It would be inefficient to be using an array for something that grows and shrinks all the time. Our mathExp vector is used to store an expression. The vector is primarily used in the calculateFinalResults method, where we iterate through the vector carrying out the rules of BODMAS. Since the expression size is dependent on the difficulty the Player enters, it is not feasible to be using an array as mentioned before that the array size is fixed at run time.

7. Data Structure

Screenshot:



```
map<int, string> Review;
```

```
queue<Player> playerQueue;
```

```
void MainGame::displayReview() {
    map<int,string>::iterator it = (currentPlayer.Review).begin();
    cout << "\nSummary of P" << currentPlayer.playerNo << " - " << currentPlayer.username << "'s performance: ";
    while (it != (currentPlayer.Review.end())) {
        cout << "\nQuestion " << it->first << ": " << it->second << "\n";
        Sleep(1000);
        it++;
    }
```

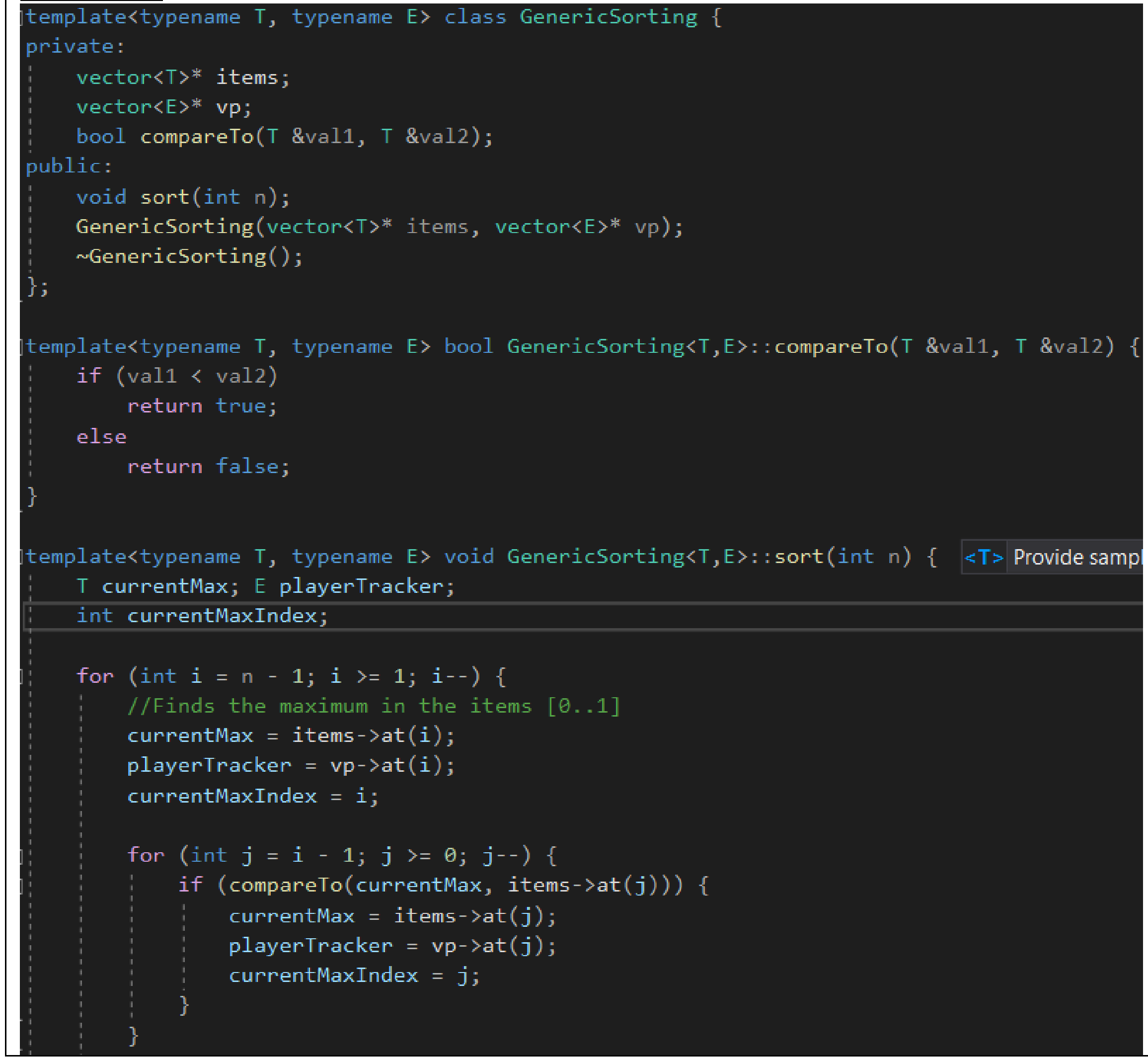
```
const char opChoice[4] = { '-', '+', '*', '/' };  //stores the operators that we are working with.
```

Motivation:

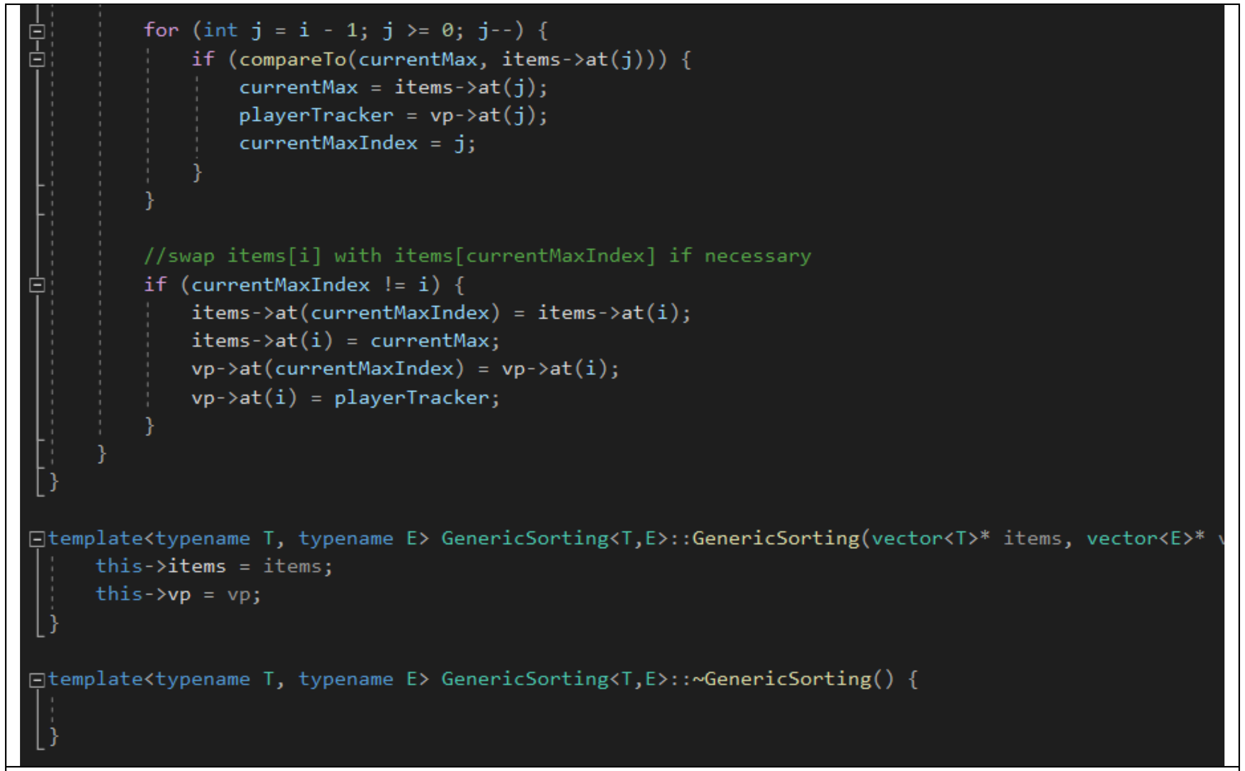
We chose to use a queue for the players because it keeps in comparison to the FiFo format we follow for asking users to answer questions, again we see that the no. of players is dependent on what the user enters. As such it would be poor programming to use an array to store the players. Since arrays are fixed at run time. We chose to use maps, since maps have better iteration methods and get values quicker, saving up CPU cycles and memory. These data structures are more efficient as opposed to use of arrays. Most of our data structures where based on memory management, since the game would be implemented in school computer laboratories, where the system specifications are low standard. We would not want a program that takes up 1Gig of RAM to be running on computers that only have 2Gigs of RAM available.

8. Class Template

Screenshot:



```
template<typename T, typename E> class GenericSorting {
private:
    vector<T>* items;
    vector<E>* vp;
    bool compareTo(T &val1, T &val2);
public:
    void sort(int n);
    GenericSorting(vector<T>* items, vector<E>* vp);
    ~GenericSorting();
};

template<typename T, typename E> bool GenericSorting<T,E>::compareTo(T &val1, T &val2) {
    if (val1 < val2)
        return true;
    else
        return false;
}

template<typename T, typename E> void GenericSorting<T,E>::sort(int n) {  <T> Provide sampl
    T currentMax; E playerTracker;
    int currentMaxIndex;

    for (int i = n - 1; i >= 1; i--) {
        //Finds the maximum in the items [0..1]
        currentMax = items->at(i);
        playerTracker = vp->at(i);
        currentMaxIndex = i;

        for (int j = i - 1; j >= 0; j--) {
            if (compareTo(currentMax, items->at(j))) {
                currentMax = items->at(j);
                playerTracker = vp->at(j);
                currentMaxIndex = j;
            }
        }
```

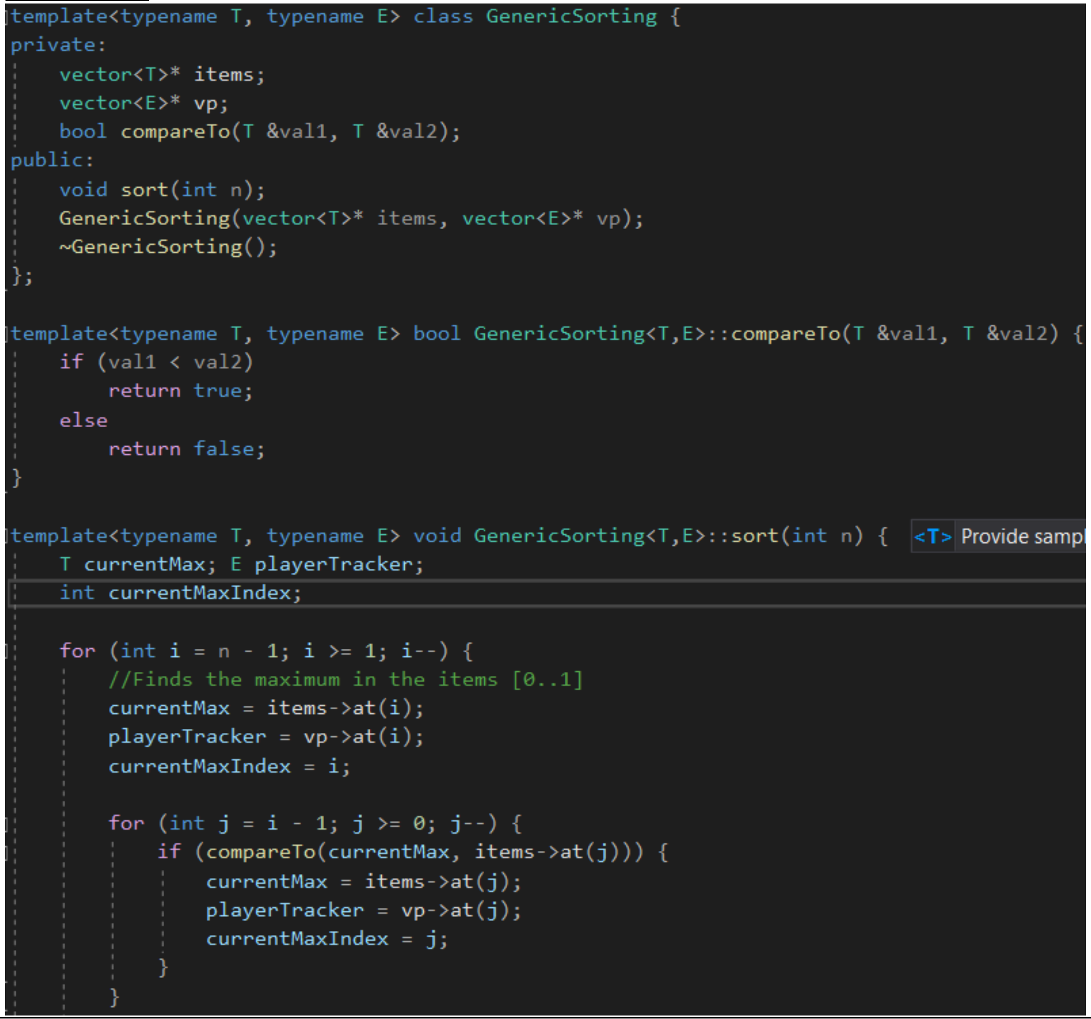


```
        for (int j = i - 1; j >= 0; j--) {
            if (compareTo(currentMax, items->at(j))) {
                currentMax = items->at(j);
                playerTracker = vp->at(j);
                currentMaxIndex = j;
            }
        }

        //swap items[i] with items[currentMaxIndex] if necessary
        if (currentMaxIndex != i) {
            items->at(currentMaxIndex) = items->at(i);
            items->at(i) = currentMax;
            vp->at(currentMaxIndex) = vp->at(i);
            vp->at(i) = playerTracker;
        }
    }
}

template<typename T, typename E> GenericSorting<T,E>::GenericSorting(vector<T>* items, vector<E>* v
    this->items = items;
    this->vp = vp;
}

template<typename T, typename E> GenericSorting<T,E>::~GenericSorting() {

}
```

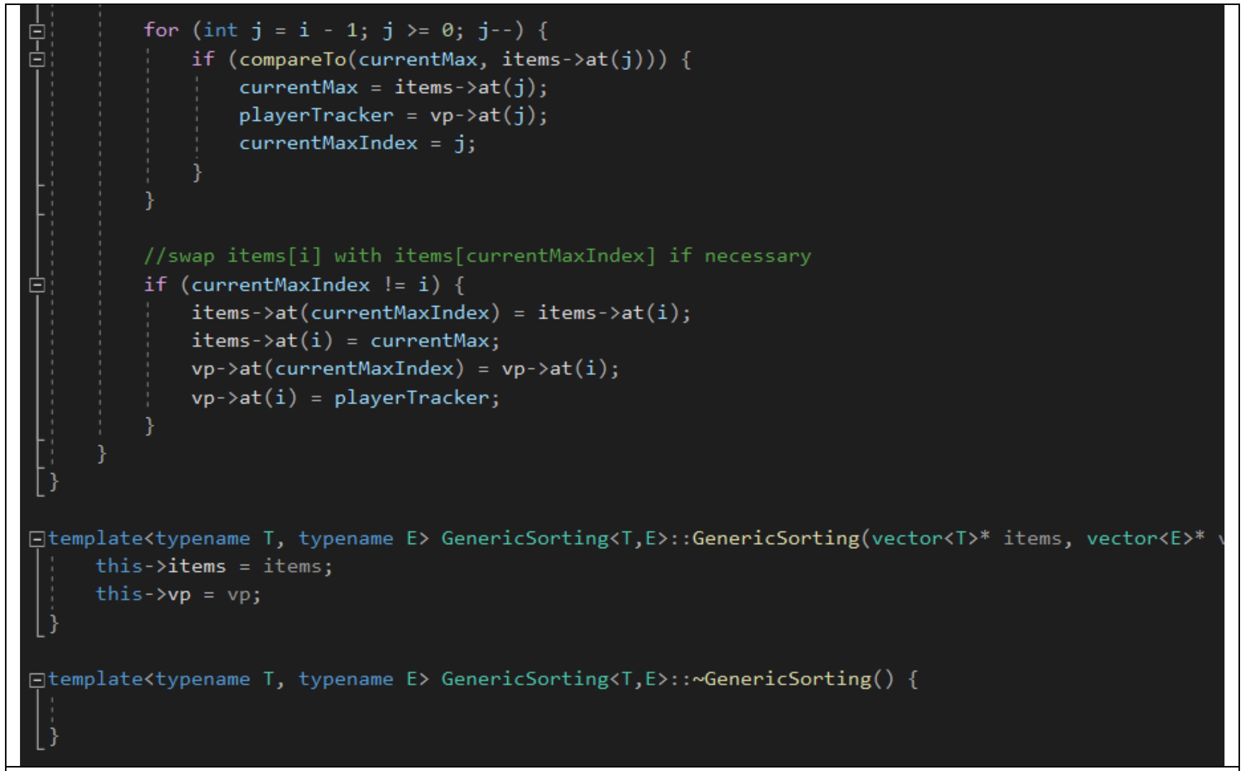
Motivation:
The template constructor is used to retrieve the two data types you would like to sort out and store an instance of them in the class. The template class contains a sort function which sorts out a vector in descending order. The use of template classes was to retrieve the two vectors we which to sort in the constructor rather than taking them as input in a function.

9. Function Template

Screenshot:



```
template<typename T, typename E> class GenericSorting {
private:
    vector<T>* items;
    vector<E>* vp;
    bool compareTo(T &val1, T &val2);
public:
    void sort(int n);
    GenericSorting(vector<T>* items, vector<E>* vp);
    ~GenericSorting();
};

template<typename T, typename E> bool GenericSorting<T,E>::compareTo(T &val1, T &val2) {
    if (val1 < val2)
        return true;
    else
        return false;
}

template<typename T, typename E> void GenericSorting<T,E>::sort(int n) {  <T> Provide sampl
    T currentMax; E playerTracker;
    int currentMaxIndex;

    for (int i = n - 1; i >= 1; i--) {
        //Finds the maximum in the items [0..1]
        currentMax = items->at(i);
        playerTracker = vp->at(i);
        currentMaxIndex = i;

        for (int j = i - 1; j >= 0; j--) {
            if (compareTo(currentMax, items->at(j))) {
                currentMax = items->at(j);
                playerTracker = vp->at(j);
                currentMaxIndex = j;
            }
        }
```

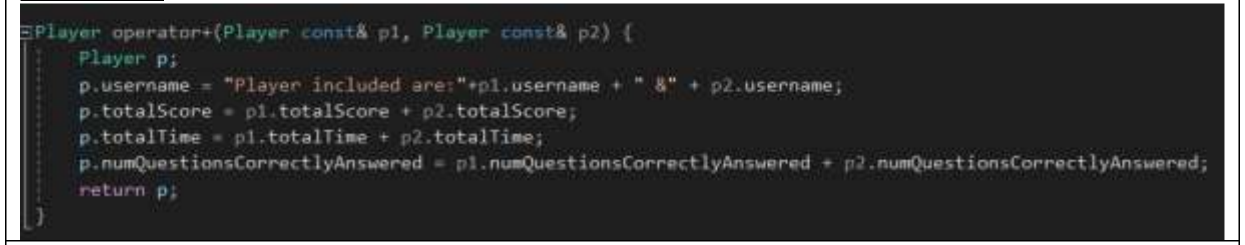


```
            for (int j = i - 1; j >= 0; j--) {
                if (compareTo(currentMax, items->at(j))) {
                    currentMax = items->at(j);
                    playerTracker = vp->at(j);
                    currentMaxIndex = j;
                }
            }

            //swap items[i] with items[currentMaxIndex] if necessary
            if (currentMaxIndex != i) {
                items->at(currentMaxIndex) = items->at(i);
                items->at(i) = currentMax;
                vp->at(currentMaxIndex) = vp->at(i);
                vp->at(i) = playerTracker;
            }
        }
}

template<typename T, typename E> GenericSorting<T,E>::GenericSorting(vector<T>* items, vector<E>* v
    this->items = items;
    this->vp = vp;
}

template<typename T, typename E> GenericSorting<T,E>::~GenericSorting() {

}
```

Motivation:
We had opted to use template function because we would be storing and sorting vectors of different primitive types. For example, we would like to sort out the players scores (an integer vector) and output the time taken for each player (an integer vector), on the other hand we would also like to display the score for each player (integer vector) and display the username of the player who obtained that score(a string vector). Instead of having to write the same function twice with the only changes being the type of vectors it contains. Creating a single template function meant we got rid of redundant code.

10. Operator Overloading

Screenshot:



```
Player operator+(Player const& p1, Player const& p2) {
    Player p;
    p.username = "Player included are:"+p1.username + " &" + p2.username;
    p.totalScore = p1.totalScore + p2.totalScore;
    p.totalTime = p1.totalTime + p2.totalTime;
    p.numQuestionsCorrectlyAnswered = p1.numQuestionsCorrectlyAnswered + p2.numQuestionsCorrectlyAnswered;
    return p;
}
```

Motivation:

We used operator overloading to add two player objects together. Combining their scores and times into a team total score and time. Exceptionally helpful because it did not mean having to write a separate function, we would just overload the + operator to add two player structs.

Section	How where the objectives met	Explanation
Functions	Fully	We fully understand the structure. How a function is used and for what aspects do we generally use functions for. Functions had helped us allot during the debugging phase where it was genuinely easier to find points which contained errors. Functions help in reducing the redundancy of code and allows for recursion. Primarily used in debugging and in modular programming. They stop the code from looking squashed and create an overall neatness to the project.
Classes	Fully	There are 3 different classes used and 1 class template. Class was used to group functions which correspond to what group or what object a function relates to. For example, the validation class contains all the functions needed to perform validation on user inputs. since objects are created with classes and all validation is performed at the beginning of the quiz. The validation class object can be deleted once the quiz commences, freeing up space and improving memory management.
Structs	Fully	Our struct does not need behaviour. Our struct is only used to store data. The only functions we would be using if the player struct was to be implemented as a class would be get and set methods. Since all variables are defaulted to private in a class and all our attributes needed to be public, we chose a struct over a class.

Pointer	Fully	Pointers exist to save space, and save memory in c++. Since it points to the actual value and a hardcopy is not stored. Pointers help by only accessing the region in memory when it is needed. We can thus manipulate the values directly instead of having to manipulate a hard copy and storing the copy of the value in another spot in memory. The use of pointers also improves performance for repetitive operations.
Reference	Fully	With the use of pointers comes the aid of references, as references get the memory address of a value and pointers use the memory address to access the values in memory. If one were to use pointers there would have to use references. References also allow use to pass by reference instead of pass by copy. This approach allows us to directly manipulate the data instead of storing the same value in another block in memory and manipulating that block.
Vector	Fully	Vectors are resizable. They can grow and shrink throughout the life span of the program. This aided us when we want to find the divisors of a number. Since the program randomly generates numbers for the expression. The size of the divisors vector would be manipulated after every instance of the expression generator object being called. Arrays in c++ however do not allow for this feature and as such vectors are more helpful in reducing the time complexity of the code.

Data Structures	Fully	A queue was used since it has faster insertion and deletion at both the beginning and end. Since our approach to the players is the FiFo method, inserting the player object back into the queue and retrieving the player object at the start of the queue allowed us to save time reducing the stime complexity of the code. the same applies to the map. A map allows for the fast retrieval of values using the key. Since our maps are dependant on the number of questions the player chose, the time complexity of retrieving items in a map is far less than the use of using a vector or array. The main advantage of an array is that values can be accessed randomly by using the index number. Hence the middle values can be retrieved faster than as opposed to other data structures. All the data structures have one attribute in common they are all used for some ability to reduce the time complexity of the program
Function Template	Fully	We used template functions for what they were intended for. To reduce redundancy of code. it would be unlikely to develop 3 different functions to take in 3 different primitive types but carry out the same procedure on these primitive types. It is because of the functionality of template functions, they can be set up to take a type of data and carry out the same operation on those data types, that we decided to use function templates in our program.

Class Template	Fully	As mentioned with the function templates. Class templates was used primarily for the function the contain, they can carry out the same operations across many different data types. It is redundant to be creating two or more different classes which carry out the same procedures.
Operator Overloading	Fully	Operator overloading is used because of the functionality it brings. It allows us to carry out procedures on different data types that you generally cannot perform. Operator overloading was used to overload the + operator so we where able to combine the stats of two players and create an overall team score, as well as many other attributes that are required in the Team struct.

Score Calculation

Scores are calculated based on the time taken to solve the problem. There is a time limit for each question is dependent on the difficulty you select. If a player were to obtain the correct answer but the time it took him to get that answer is over the stipulated time limit, they only get 50 points. If a player were to incorrectly answer the question, they will receive no points. Our score calculation algorithm is done as follows:

- The system time is recorded when the question is displayed to the user. Giving us the start time for the question. The moment the question is displayed it is a race against time.
- When a player enters a valid value, the system time is recorded again, giving us the end time for the question
- We then subtract the end time from the start time to end up with the duration it took the Player to answer.
- We then check if what the user entered was correct.
- If the answer is correct:
    - We check if the duration is longer than the stipulated time limit.
        - If it is then the player will only get 50 points.
    - If the player is well within the time limit:
        - The players time is subtracted from the time limit giving us a difference.
        - The difference is then divided by the time limit and multiplied by 90.
        - 10 is added to the equation to give us a mark out of 100.
- The players score for the question is then added to the players overall score.

$(\frac{timelimit - playertime}{timelimit} *90) +1$

Additional Item

What does your quiz include?	Cross (X) the appropriate box
Simplicity in expression	X
Filter function	X
Multiplayer	X
Sound	X
Support	X
About/Credits	X
Rollback feature	X
Avatars	X
Difficulty level	X
Progress bar	X
Story	X
Validation	X
MVP	X
Review	X

Simplicity in expression:

All expressions generated result in a whole number being produced. If a number is generated e.g. x and the division operator is chosen as random. The number following the division operator would be a factor of x. In the case of multiple division. If there is a subexpression like x / y / z.

x / y is computed and used to generate a z value such that the end answer is always an integer and there will never be decimal values in any expression generated.

This was primarily done so that players would not get confused when answering the question.

Filter function:

The filter function is used in conjunction to the MVP feature. Here the players enter the category for which the leader-board should display.

Multiplayer:

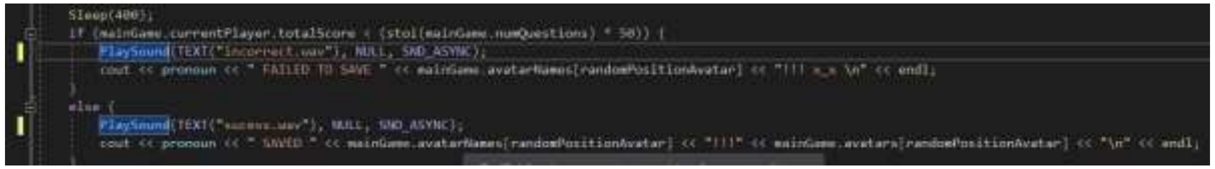
Multiplayer is accessible when a user enters a value greater than 1 for the no. of players prompt. Each user has to enter their own unique username to procced. Failure to do this will result in the player being asked to re-enter their username as the one they had chosen is unavailable.

There are two types of multiplayer categories:

1. (2/3 players) Versus – this sees each player fend for themselves and take on their friend to see who can end up with the highest score at the end. The difficulty and number of questions remain the same. Players follow a First In First Out format for answering question. For example, in the three-player format:
    - Player 1 answers question 1, Player 2 answers question 1, Player 3 answers question 1 then Player 1 moves to question 2.
2. (4 players) Teams – This sees Player 1 pair up with Player 3 and Player 2 team up with Player 4. While the format is the same (Players answer questions in a FiFo form), the key difference is that at the end. Players stats for each pair are combined to give an overall team stat for the players. The scores from both teams are compared and the statistics of the winning team are displayed. You can have the highest score but still be on the losing team.

Players can choose to view overall stats, choose to play again, or decide to end game. In each category players can view a leader board, Display either the players score, their time or the number of questions answered correctly in descending order, dependant on the players choice

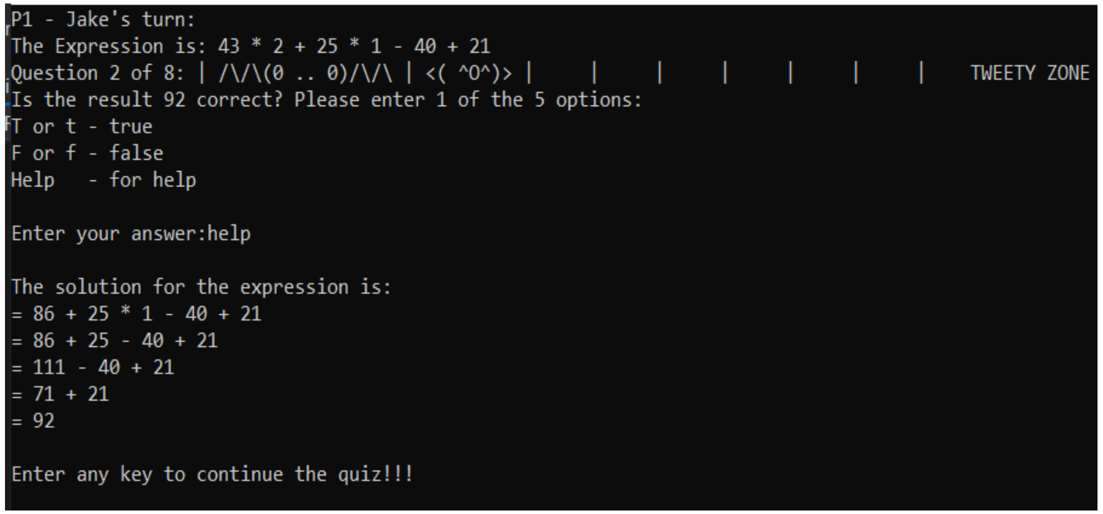
Sound:

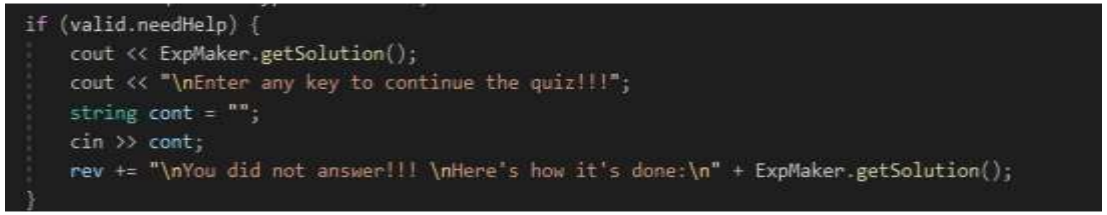


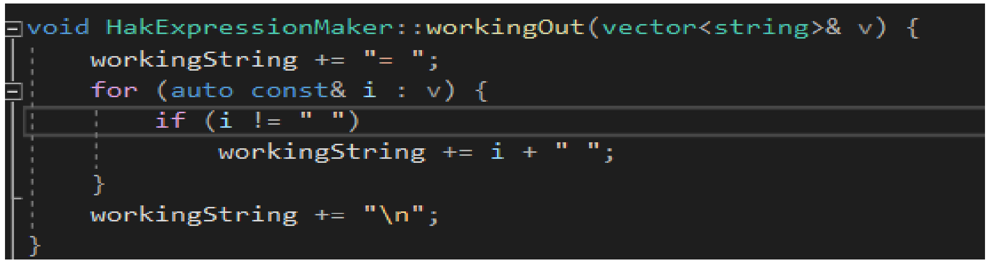
```
Sleep(400);
if (mainGame.currentPlayer.totalScore < (stol(mainGame.numQuestions) * 50)) {
    PlaySound(TEXT("incorrect.wav"), NULL, SND_ASYNC);
    cout << pronoun << " FAILED TO SAVE " << mainGame.avatarNames[randomPositionAvatar] << "!!! x_x \n" << endl;
}
else {
    PlaySound(TEXT("success.wav"), NULL, SND_ASYNC);
    cout << pronoun << " SAVED " << mainGame.avatarNames[randomPositionAvatar] << "!!!" << mainGame.avatars[randomPositionAvatar] << "\n" << endl;
```

If the player ends up with a score over 50% a supportive and cheerful song is played, signalling to the user that thy had saved their character, something similar along the lines of “HORRAAYYYY”. Else we have a sad crowd awhing for the player. We make use of the Windows.h file along with the Winmm.lib to give us access to the PlaySound() function.

Support:

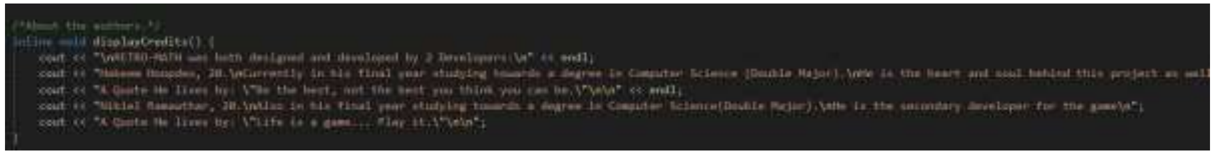

```
P1 - Jake's turn:
The Expression is: 43 * 2 + 25 * 1 - 40 + 21
Question 2 of 8: | /\/\(0 .. 0)/\/\ | <( ^0^)> |     |     |     |     |     |     |    TWEETY ZONE
Is the result 92 correct? Please enter 1 of the 5 options:
T or t - true
F or f - false
Help   - for help

Enter your answer:help

The solution for the expression is:
= 86 + 25 * 1 - 40 + 21
= 86 + 25 - 40 + 21
= 111 - 40 + 21
= 71 + 21
= 92

Enter any key to continue the quiz!!!
```


```
if (valid.needHelp) {
    cout << ExpMaker.getSolution();
    cout << "\nEnter any key to continue the quiz!!!";
    string cont = "";
    cin >> cont;
    rev += "\nYou did not answer!!! \nHere's how it's done:\n" + ExpMaker.getSolution();
}
```


```
void HakExpressionMaker::workingOut(vector<string>& v) {
    workingString += "= ";
    for (auto const& i : v) {
        if (i != " ")
            workingString += i + " ";
    }
    workingString += "\n";
}
```

If a player enters the keyword help in any variation for any 1 of the questions. The working out for that expression is shown as output. The working out string uses the expression vector which shrinks after every calculation performed on it. In that way learners can follow the calculations going on in the problem and how it is being processed.

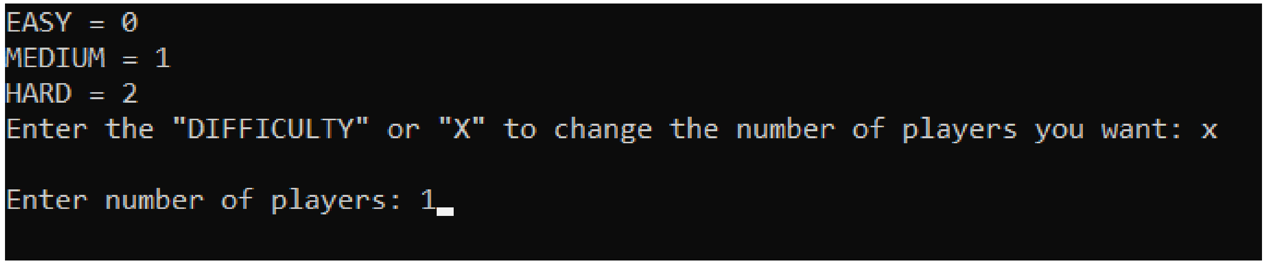
Credits:



```
/*About the authors.*/
inline void displayCredits() {
    cout << "\nRETRO-MATH was both designed and developed by 2 Developers:\n" << endl;
    cout << "Hakeem Hoopdeo, 20.\nCurrently in his final year studying towards a degree in Computer Science (Double Major).\nHe is the heart and soul behind this project as well
    cout << "A Quote He lives by: \"Be the best, not the best you think you can be.\"\n\n" << endl;
    cout << "Nikhil Ramauthar, 20.\nAlso in his final year studying towards a degree in Computer Science(Double Major).\nHe is the secondary developer for the game\n";
    cout << "A Quote He lives by: \"Life is a game... Play it.\"\n\n";
}
```

Once the player reaches the end of the game. They have the choice of either ending the game or playing again. If the user decides to end the game a short description about the developers are displayed along with their respective roles in the project

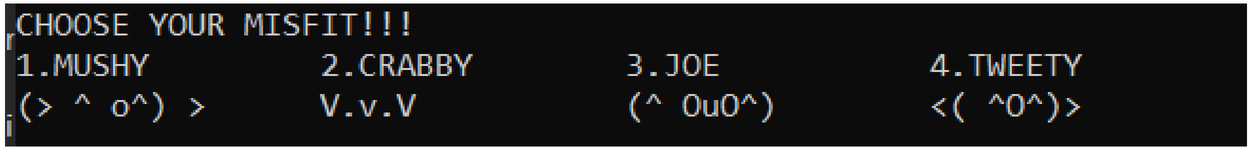
Rollback:



```
EASY = 0
MEDIUM = 1
HARD = 2
Enter the "DIFFICULTY" or "X" to change the number of players you want: x

Enter number of players: 1
```

If a player is unhappy with their choice of questions, difficulty level, username or mode selection. They have the option of changing those setting by entering the character X. Which will display the previous setting for the user to change. Settings are changeable before the commencement of the game but become static once the players enter the quiz.

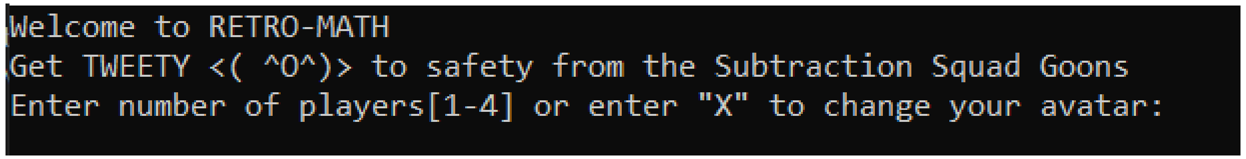
Avatars:



```
CHOOSE YOUR MISFIT!!!
1.MUSHY          2.CRABBY          3.JOE             4.TWEETY
(> ^ o^) >       V.v.V             (^ OuO^)          <( ^O^)>
```

Players have the choice of picking their own individual avatar at the beginning of the game. Each avatar corresponds to an individual from the band of misfits and play a vital role in the story telling of the game. There is also a villain avatar which is selected based on the difficulty level chosen by the player.

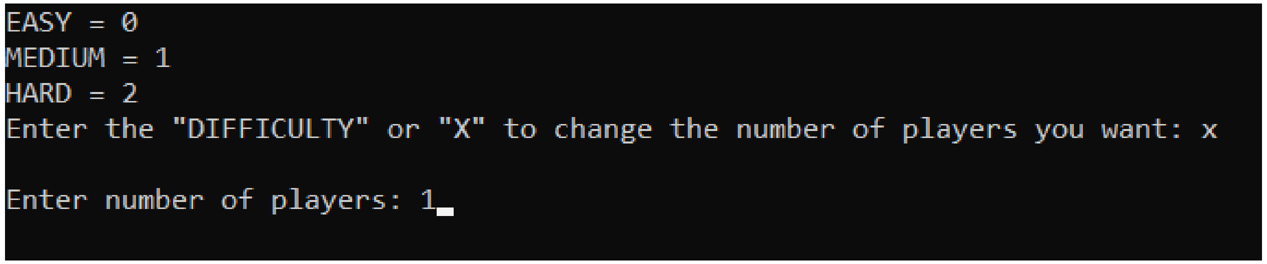
Story:



```
Welcome to RETRO-MATH
Get TWEETY <( ^O^)> to safety from the Subtraction Squad Goons
Enter number of players[1-4] or enter "X" to change your avatar:
```

Users are presented with a scenario at the start of the game. To save their character from the hands of some villain. The approach for this was to give the program a more video game experience. The difficulty chosen by the player allows us to cycle an array of villains each which correspond to a different difficulty and each with their own unique background story. A more in-depth analysis of the story is given in the user-manual.

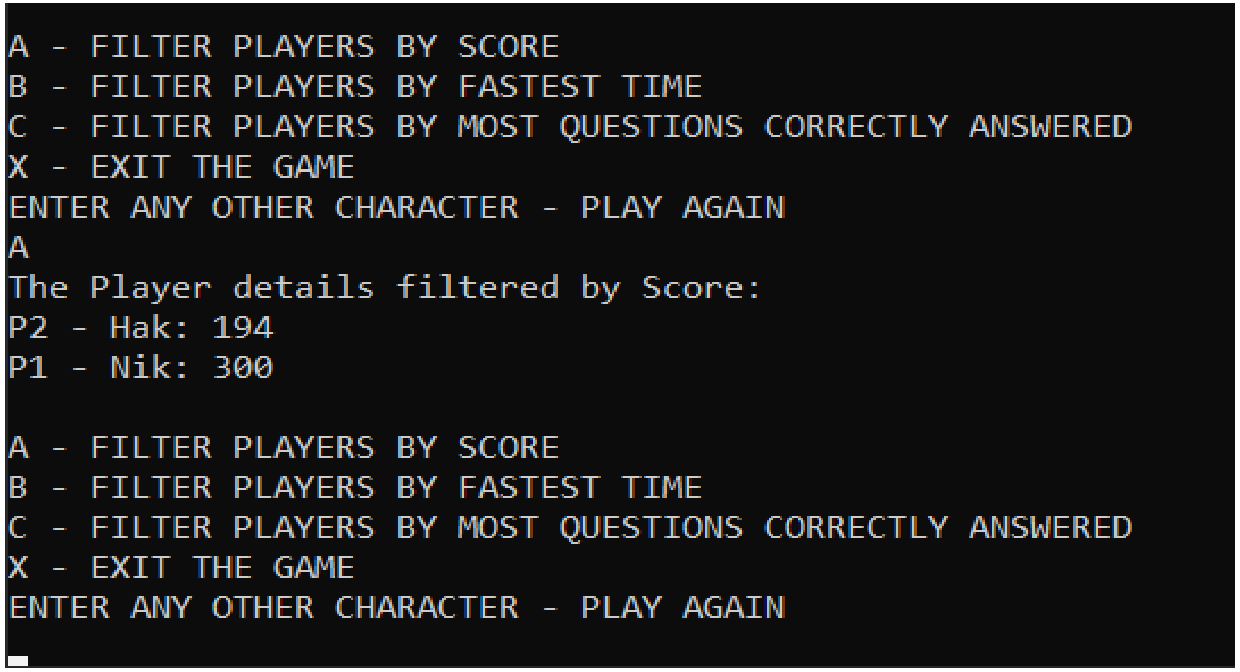
Difficulty level:



```
EASY = 0
MEDIUM = 1
HARD = 2
Enter the "DIFFICULTY" or "X" to change the number of players you want: x

Enter number of players: 1_
```

Players are given the choice of choosing different difficulty levels. Each difficulty levels corresponds to some length for our expression. Easy mode gives the user an expression containing only 1 operand while hard gives the user an expression containing 5 different operands. Having difficulty mode means the range for implementation is expanded. Easy mode can be used to teach children entering the foundation phase of their schooling career. Hard mode can be used to teach children entering high school or those who are beginning to learn bodmas. The difficulty level is used in conjunction to the progress bar as the villain character selected is unique to each difficulty.

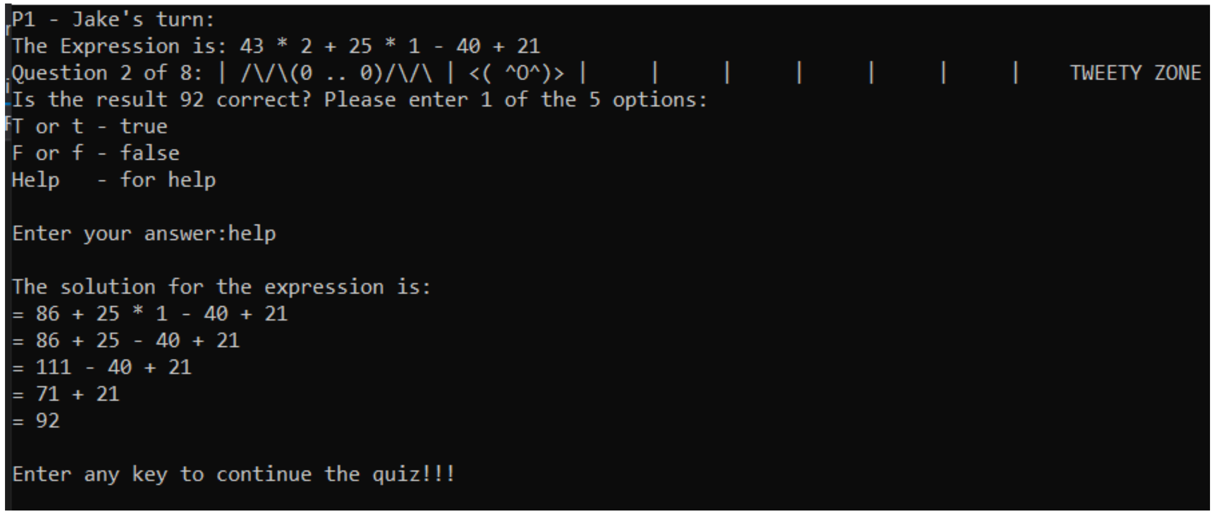
MVP:



```
A - FILTER PLAYERS BY SCORE
B - FILTER PLAYERS BY FASTEST TIME
C - FILTER PLAYERS BY MOST QUESTIONS CORRECTLY ANSWERED
X - EXIT THE GAME
ENTER ANY OTHER CHARACTER - PLAY AGAIN
A
The Player details filtered by Score:
P2 - Hak: 194
P1 - Nik: 300

A - FILTER PLAYERS BY SCORE
B - FILTER PLAYERS BY FASTEST TIME
C - FILTER PLAYERS BY MOST QUESTIONS CORRECTLY ANSWERED
X - EXIT THE GAME
ENTER ANY OTHER CHARACTER - PLAY AGAIN
```

In multiplayer once the game is over players have a choice to display the leader-board for each different category of the game. Players can view who had the highest score. Who had the fastest time and who had achieved the most number of questions correctly. A player in comparison might have achieved the highest score for the game but not necessarily the most number of questions correctly. This is since player scores are calculated based on the time taken for them to answer.

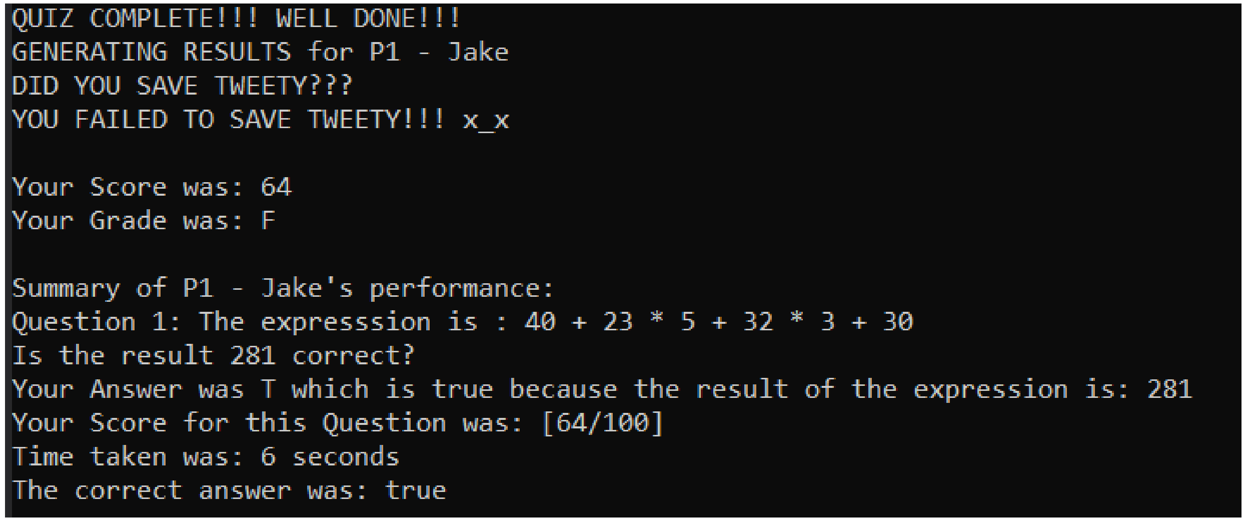
Progress bar:



```
P1 - Jake's turn:
The Expression is: 43 * 2 + 25 * 1 - 40 + 21
Question 2 of 8: | /\/\(0 .. 0)/\/\ | <( ^O^)> |     |     |     |     |     |     |    TWEETY ZONE
Is the result 92 correct? Please enter 1 of the 5 options:
T or t - true
F or f - false
Help   - for help

Enter your answer:help

The solution for the expression is:
= 86 + 25 * 1 - 40 + 21
= 86 + 25 - 40 + 21
= 111 - 40 + 21
= 71 + 21
= 92

Enter any key to continue the quiz!!!
```

The progress bar depicts how many levels are left before the player is done with the quiz. It contains the players chosen avatar and the villain avatar for each difficulty. The progress bar depicts our story. That the character from the band of misfits are being chased by a member of the Subtraction Squad. (Line 3 of the above screenshot shows the progress bar. It depicts the players character being chased by a villain. The position of the players avatar informs the player how far or how close they are to the end of the game.)

Review:

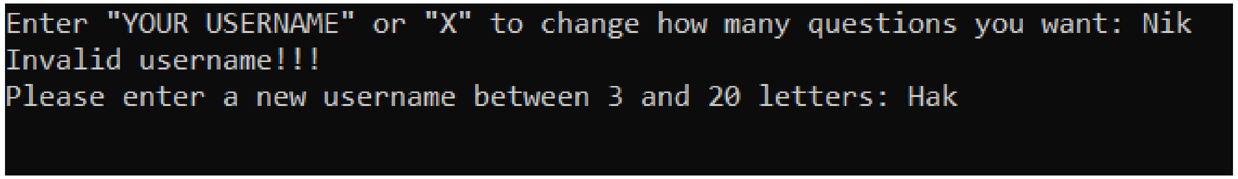


```
QUIZ COMPLETE!!! WELL DONE!!!
GENERATING RESULTS for P1 - Jake
DID YOU SAVE TWEETY???
YOU FAILED TO SAVE TWEETY!!! x_x

Your Score was: 64
Your Grade was: F

Summary of P1 - Jake's performance:
Question 1: The expresssion is : 40 + 23 * 5 + 32 * 3 + 30
Is the result 281 correct?
Your Answer was T which is true because the result of the expression is: 281
Your Score for this Question was: [64/100]
Time taken was: 6 seconds
The correct answer was: true
```

At the end of the quiz, players are presented with an overall look at their game attempt. Player reviews consist of:

- The question presented to the player.
- The players response to the question.
- The time taken to answer the question.
- If the player answered incorrect. The intended answer is displayed along with the steps taken to achieve the final answer.

Validation:



```
Enter "YOUR USERNAME" or "X" to change how many questions you want: Nik
Invalid username!!!
Please enter a new username between 3 and 20 letters: Hak
```

All player inputs are validated before they are committed to the program. This is done so that the players do not run into any problems whilst going about in the game. Players inputs are captured and are processed in the validation class. If an input does not match our criteria. For example, if a player enters a difficulty level of 5, when the range is from 0 to 2. The player will be asked to re-enter their desired difficulty level. This continues to happen until the player enters a valid input. Validation at the most part is important in the multiplayer format. We do not want 2 or more players to have the same username. Primarily because it then becomes confusing when player stats and the review for each player is displayed. Hence the need for validation of player inputs.

Appendix

[This is **OPTIONAL**: *Provide any additional information that you would like.*]