EXAMTEX

Brief

This project creates a markup language for writing exam files

The user can feed two files
1. A question bank file which contains various
questions of a particular type such as MCQ, numerical, TrueFalse, multiple MCQ, single word along with their difficulties and scores.
2. A question paper file which is used to prepare the question paper.

Working

1. We initially allocate vectors to store data i.e. ,difficulty, id and score associated with each question.

2. We then call the readQuestionBank function from readFile.c to read the question bank and store the data in vectors, we all also check for syntax errors in this step.

3. The vectors are sorted based on the difficulty.

4. The sample question paper is read and validated and the range in which difficulty of the questions are to be picked is noted .

5. We then apply binary search on sorted vector arrays to get the questions in range from which we can choose random questions based on the requirement of the user.

6. For each selected question we make use of the id number stored in the vector associated with it and use the fseek and print functions to print the question paper.

Rules for language are written in the rules for language.txt file.

Data Structures Used

Vectors (dynamic arrays):-

1)String - dynamic strings
2)Node - a struct containing the difficulty, score and the id for each question.
3)Character arrays.

Vec.c

```
vector allocate(int elementType, unsigned int dim);
	-allocates memory for new vector of given elementType and initial size as dim
	-O(1) time complexity

void push_back(vector *v, parameterUnion val);
	-pushes  an element in the end of the vector v
	-O(1) time complexity

parameterUnion back(vector v);
	-returns the element in the end
	-O(1) time complexity

parameterUnion front(vector v);
	-returns the element in the front
	-O(1) time complexity

vector *reallocate(vector *v, int x);
	-reallocates memory from old dimensions to new(x)
	-O(1) time complexity

void pop_back(vector *v);
	-pops the last element
	-O(1) time complexity

unsigned int size(vector *v);
	-returns number of elements in the vector
	-O(1) time complexity

void add_string(vector *str, char c);
	- Adds char at the end
	-O(1) time complexity

void cpy_string(vector *s, char* c);
	- copies string to vector s
	-O(1) time complexity

char *return_string(vector *str);
	- returns string in vector to char* type
	-O(1) time complexity

void deletevector(vector * v);
	- Deletes vector, frees space allocated
	-O(1) time complexity
```

Binary_Search

In order to find questions within the given difficulty range we need to use binary search.

-Time complexity is O(log n)

-Best case time complexity is O(1)

Int binary_ub_search(vector *v, double upper_limit);

-This code returns the position of the largest element just smaller than the upper bound

Int binary_lb_search(vector *v, double upper_limit);

-This code returns the position of the smallest element just greater than the lower bound

Select random Question

void select_rand_q(int ub, int lb, int num_q, char *type, FILE *qb, FILE *op, FILE *oa) ;

-This function selects random questions from question bank file and read them

-Time complexity is O(n)

- It also checks if there are num_q number of unique questions or not.

void b_search(vector *vec, float diff_ub, float diff_lb, int num_q, char type[], FILE *fp, FILE *op, FILE *oa);

-This function first calls the selectrand_q_() function to find some random question and print them on the questionPaper output file.

-Time complexity is O(length_of_file)^2

Merge_sort.c:

void merge(vector *vec, int l, int m, int r);

- O(n) time complexity
- Merges two vectors by by comparing to get sorted single vector out

void mergeSort(vector *vec, int l, int r);

-O(nlogn) time complexity

-Repeatedly calls itself and merge function to split into smaller pieces and merge back vector by sorting repeatedly

Printers:

These functions produce the question text, score, options etc. in the correct output format to the questionPaper file and also prints the correct answer to the answer file.

-Time complexity is O(length_Of_Argument)

Validators:

This explanation stands for validators of every type(mcq, mulmcq, oneWord, truefalse, numerical)

-Checks if each type of the parameter in the given question bank is valid according to the data type and also checks if they are repeated or not present at all.

- O(length_Of_argument) time complexity

void remove_Spaces(char **c);

-Removes extra spaces from the start and end of the text and options

Utils:

readFile.c:

void readQuestionBank(FILE *qb);

-Time complexity is O(length_Of_File);

-It reads the type of question and calls the appropriate validator functions.

```
Void readSamplePaper(FILE *qb, FILE *sp, FILE *op, FILE* oa);
      -Time complexity is O(length_Of_File);
      -It reads the question paper file and calls the appropriate functions to select a
question and print it.
```

Parsers.c

```
int isSyntax(char c);
      -O(1) time complexity
      -
void raiseSyntaxError(char c);
      -O(1) time complexity

int parseType(FILE *fp, char **des);
      -O(length_of_argument) time complexity
      -It checks the type of question and calls the validator(called from the
      readQuestionBank())

int parseArgument(FILE *fp, char **param, char **val);
      -O(length_of_argument) time complexity
      -It stores the tag and the value in two separate char arrays.
      -Used for validators and printer functions.
```

cleanText.c

```
void remove_spaces(char **c);
_____-It remove spaces in the text
      -O(len(text))
void clean(char text[])
      -cleans code
      -O(len(text))
```

Bracket_balancing.c:

```
int IsLeft(char ch);
      -checks if the bracket is a left bracket, ({,[,(,<)
      -O(1)
int IsRight(char ch);
      -checks if the bracket is a right bracket, (},],),>)
      -O(1)
int main();
      -checks if the brackets are balanced in correctly and in order
      -O(length_of_file)
```

Work Division:

Abhijnan Vegi :

a. Files for parsers
b. Validators for numerical type
c. Printer functions for numerical type
d. Functions for reading the question bank and question paper.
e. CMake files and other prerequisite files.
f. Verification of codes
g. Finding and fixing bugs
h. main file
i . testing of the final files
j. examples for numerical type questions

Shreyansh Agarwal:

a. Vector files
b. Validators for multiple MCQ questions.
c. Printer functions for multiple MCQ
d. Verification of codes
e. Finding and fixing errors.
f. main file.
g. testing of the final files
h. Examples for multiple mcq type questions

Keerthi Pothalaraju:
a. Validators for true false functions
b. Printer functions for single correct MCQ questions.
c. Binary search functions
d. Vector sort function.
e. bsearch and print functions.
f. Examples for True False type questions
g. Testing and bug fixes.

Sumanth V.N.M:
a. Validator for single word questions
b. Printer functions for single word
c. Parenthesis checker.
d. Examples for single word questions.
e. Testing

Pratyush Mohanty:
a. Validator for single correct MCQ type questions
b. Printer functions for single correct MCQ
c. Quick sort function.
d. Code of Basic data structures of dynamic array.
e. Testing
f. Examples for single correct MCQ.
g. Bug fixes

LINK FOR GITHUB REPO:

https://github.com/AbhijnanVegi/DSA-Mini-project.git