BMS COLLEGE OF ENGINEERING
TELECOMMUNICATION ENGINEERING
(Autonomous Institution under VTU)
Bull Temple Road, Bangalore- 560019

Subject: C++ and Data Structures
Subject Code: 16TE6DE2OP

Presented By:
Abhash Kumar Jha(1BM17TE002)

Aim : To demonstrate a boggle board game and how it functions through c++.

Explanation :

t	h	i	s	i	s	a
s	i	m	p	l	e	x
b	x	x	x	x	e	b
x	o	g	g	l	x	o
x	x	x	D	T	r	a
R	E	P	E	A	d	x
x	x	x	x	x	x	x
N	O	T	R	E	-	P
x	x	D	E	T	A	E

This is a boggle board.
We are given a board set as like as shown in the above figure(sample). We have to find the words contained in this boggle board which are also in the list of words we have….

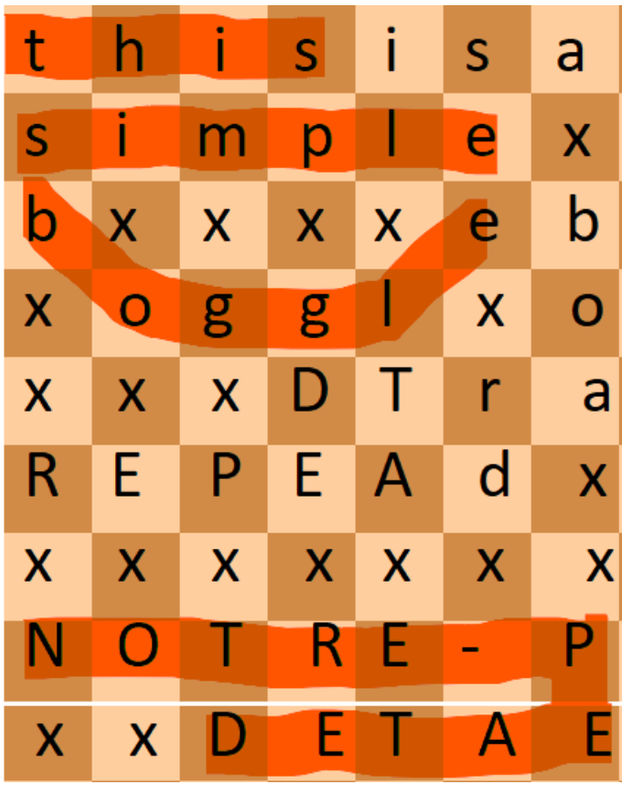
Here is the sample list of word pertinent to above sample figure:

```
[
  "this",
  "is",
  "not",
  "a",
  "simple",
  "boggle",
  "board",
  "test",
  "REPEATED",
```

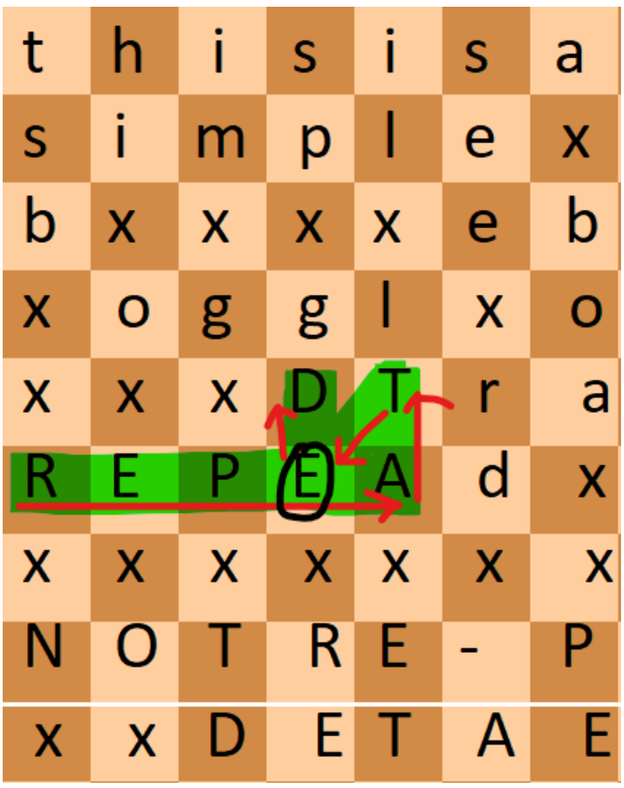
```
  "NOTRE-PEATED"
]
```

Rules:
To find a word in the board, we can traverse from the board at any direction (horizontally, vertically or diagonally) but the same letter cannot be repeated.

For Example:



These are some valid words:
-> this
-> simple
-> boggle
-> NOTRE-PEATED



This is not a valid word selection as circled E is repeating and hence ‘repeated’ is not the valid selection.

Code :

```
      /*given a 2-d array of potentially unequal height and width
containing letters; this matrix is called boggle board

We are also given a list of words

We have to extract all the words contained in the boggle board and
which are present in the list

A word is constructed in the boggle board(horizontally,vertically and diagonally)
without using any single letter at a given position more than once;

2 or more words can overlap and use the same letters in the boggle board
*/

//.....
.....
.............



#include <bits/stdc++.h>
#include <string>
using namespace std;


//.............................suffix tree.....................................
..........

class TrieNode{
    public:
        unordered_map<char,TrieNode *> children;
```

```
        string word = "";
};



class Trie{
    public:
        TrieNode *root;
        char endSymbol;

        Trie();
        void add(string str);
};

Trie::Trie(){
    this->root = new TrieNode();
    this->endSymbol = '*';
}

void Trie::add(string str){
    TrieNode *node = this->root;
    for(char letter : str){
        if(node->children.find(letter) == node->children.end()){
            TrieNode *newNode = new TrieNode();
            node->children.insert({letter,newNode});
        }
        node = node->children[letter];
    }
    node->children.insert({this->endSymbol,NULL});
    node->word = str;
}


//.....................................................................
...........







//function for checking neigbour
vector<vector<int>> getNeighbours(int i,int j ,vector<vector<char>> board);

//function for exploring the neighbours
void explore(int i,int j,vector<vector<char>> board, TrieNode *trieNode,
                         vector<vector<bool>> *visited,
                        unordered_map<string,bool> *finalWords);



vector<string> boggleBoard(vector<vector<char>> board, vector<string> words) {
  Trie trie;
    for(string word: words){
        trie.add(word);
    }

    //hash table for reducing redundancy

    unordered_map<string,bool> finalWords;
```

```
    //initialzing visited to all false assuming symmetrical size throughout one s
ide
    //dimensions -> board.size() = height & board[0].size = width
    vector<vector<bool>> visited(board.size(),vector<bool>(board[0].size(),false)
);

    //exploring all nodes one by one (basically going trough all the nodes)
    for(int i=0;i<board.size();i++){
        for(int j=0;j<board[0].size();j++){
            explore(i,j,board,trie.root,&visited,&finalWords);
        }
    }

    //appending from hash table to the final vector answer
    vector<string> finalWordsArray;
    for(auto iter : finalWords){
        finalWordsArray.push_back(iter.first);
    }

    return finalWordsArray;
}



void explore(int i,int j,vector<vector<char>> board, TrieNode *trieNode,
                        vector<vector<bool>> *visited,
                       unordered_map<string,bool> *finalWords){

    //if visited = true return
    //if letter not found in the trie,return
    //since this is present , go to this letter i.e. mark it visited
    //go to next letter in suffix trie

    //if next children of the TrieNode is found in the hashmap,
    //insert it in the finalwords hashmap

    //else get neighbours

    if(visited->at(i)[j]){
        return;
    }

    char letter = board[i][j];
    if(trieNode->children.find(letter) == trieNode->children.end()){
        return;
    }

    visited->at(i)[j]= true;
    trieNode = trieNode->children[letter];

    if(trieNode->children.find('*') != trieNode->children.end()){
        finalWords->insert({trieNode->word,true});
    }

    vector<vector<int>> neighbours = getNeighbours(i,j,board);
    for(vector<int> neighbour : neighbours){
        //going clockwise
        explore(neighbour[0],neighbour[1],board,trieNode,visited,finalWords);
    }
```

```
    //now back-track
    visited->at(i)[j] = false;

}


//.....................................................................................
.........................



vector<vector<int>> getNeighbours(int i,int j ,vector<vector<char>> board){

    vector<vector<int>> neighbours;

    if(i>0 && j>0){
        //not the first row and first column
        neighbours.push_back({i-1,j-1});
    }

    if(i>0 && j < board[0].size()-1){
        //not the first row and not the last column
        neighbours.push_back({i-1,j+1});
    }

    if(i < board.size() -1 && j<board[0].size() -1){
        //not the last row and not the last column
        neighbours.push_back({i+1,j+1});
    }

    if(i < board.size()-1 && j>0){
        //not the last row and not the first column
        neighbours.push_back({i+1,j-1});
    }

    if(i > 0){
        //
        neighbours.push_back({i-1,j});
    }

    if(j > 0){
        //
        neighbours.push_back({i,j-1});
    }

    if(i < board.size()-1){
        //
        neighbours.push_back({i+1,j});
    }

    if(j < board[0].size()-1){
        //
        neighbours.push_back({i,j+1});
    }

    return neighbours;
}


vector<string> select_word(int set){
```

```
    vector<vector<string>> buffer_words{
                                        {
                                          "this",
                                          "is",
                                          "not",
                                          "a",
                                          "simple",
                                          "boggle",
                                          "board",
                                          "test",
                                          "REPEATED",
                                          "NOTRE-PEATED"
                                        },
                                        {
                                          "san",
                                          "sana",
                                          "at",
                                          "vomit",
                                          "yours",
                                          "help",
                                          "end",
                                          "been",
                                          "bed",
                                          "danger",
                                          "calm",
                                          "ok",
                                          "chaos",
                                          "complete",
                                          "rear",
                                          "going",
                                          "storm",
                                          "face",
                                          "epual",
                                          "dangerous"
                                        },
                                        {
                                          "commerce",
                                          "complicated",
                                          "twisted",
                                          "zigzag",
                                          "comma",
                                          "foobar",
                                          "baz",
                                          "there"
                                        },
                                        {
                                          "frozen",
                                          "rotten",
                                          "teleport",
                                          "city",
                                          "zutgatz",
                                          "kappa",
                                          "before",
                                          "rope",
                                          "obligate",
                                          "annoying"
                                        },
                                        {
                                          "abcd", "abdc", "acbd", "acdb", "adbc", "adcb
", "abca"
```

```
                                    }};
    vector<string> set_word = buffer_words[set];

    return set_word;

}

vector<vector<char>> select_set(int set){

    vector<vector<vector<char>>> buffer_group{
                                     {
                                      {'t', 'h', 'i', 's', 'i', 's', 'a'},
                                      {'s', 'i', 'm', 'p', 'l', 'e', 'x'},
                                      {'b', 'x', 'x', 'x', 'x', 'e', 'b'},
                                      {'x', 'o', 'g', 'g', 'l', 'x', 'o'},
                                      {'x', 'x', 'x', 'D', 'T', 'r', 'a'},
                                      {'R', 'E', 'P', 'E', 'A', 'd', 'x'},
                                      {'x', 'x', 'x', 'x', 'x', 'x', 'x'},
                                      {'N', 'O', 'T', 'R', 'E', '-', 'P'},
                                      {'x', 'x', 'D', 'E', 'T', 'A', 'E'}
                                    },
                                    {
                                      {'y', 'g', 'f', 'y', 'e', 'i'},
                                      {'c', 'o', 'r', 'p', 'o', 'u'},
                                      {'j', 'u', 'z', 's', 'e', 'l'},
                                      {'s', 'y', 'u', 'r', 'h', 'p'},
                                      {'e', 'a', 'e', 'g', 'n', 'd'},
                                      {'h', 'e', 'l', 's', 'a', 't'}
                                    },
                                    {
                                      {'c', 'o', 'm'},
                                      {'r', 'p', 'l'},
                                      {'c', 'i', 't'},
                                      {'o', 'a', 'e'},
                                      {'f', 'o', 'd'},
                                      {'z', 'r', 'b'},
                                      {'g', 'i', 'a'},
                                      {'o', 'a', 'g'},
                                      {'f', 's', 'z'},
                                      {'t', 'e', 'i'},
                                      {'t', 'w', 'd'}
                                    },
                                    {
                                      {'f', 't', 'r', 'o', 'p', 'i', 'k', 'b', 'o'}
,
                                      {'r', 'w', 'l', 'p', 'e', 'u', 'e', 'a', 'b'}
,
                                      {'j', 'o', 't', 's', 'e', 'l', 'f', 'l', 'p'}
,
                                      {'s', 'z', 'u', 't', 'h', 'u', 'o', 'p', 'i'}
,
                                      {'k', 'a', 'e', 'g', 'n', 'd', 'r', 'g', 'a'}
,
                                      {'h', 'n', 'l', 's', 'a', 't', 'e', 't', 'x'}
                                    },
                                    {
                                      {'a', 'b'}, {'c', 'd'}}};

    vector<vector<char>> set_group = buffer_group[set] ;
```

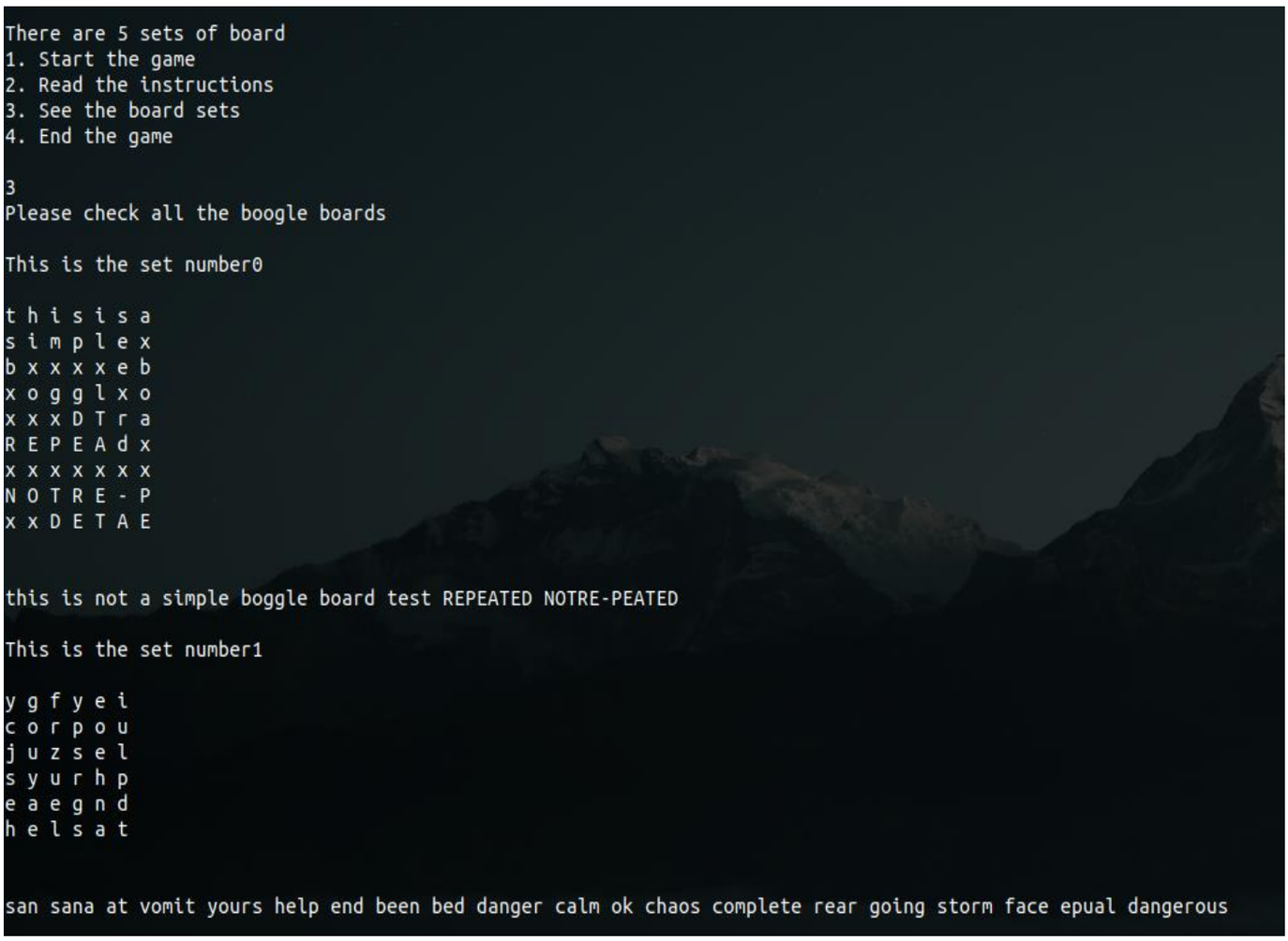
```
    return set_group;
}



int main(){

    cout << ".............................WELCOME TO THE BOGGLE BOARD.....
.............................." << endl;
    cout << endl << endl;


    bool end = false;
    while(!end){
        cout << "There are 5 sets of board" << endl;

        cout << "1. Start the game" << endl;
        cout << "2. Read the instructions" << endl;
        cout << "3. See the board sets" << endl;
        cout << "4. End the game" << endl;
        cout << endl;
        int option;
        cin >> option;



        if(option == 1){
            cout << endl << endl;
            cout << "Please select the set from set 0 to set 4" << endl;
            int set;
            cin >> set;

            if(set >= 5){
                cout << " Please Enter a valid set" << endl;
                break;
            }

            //display the set
            cout << endl << endl;
            vector<vector<char>> myboard = select_set(set);
            vector<string> mywords = select_word(set);

            for(vector<char> board_i : myboard){
                for(auto k : board_i){
                    cout << k << " " ;
                }
                cout << endl;

            }

            cout << endl << endl;
            //display the word set

            for(auto word : mywords){
                cout << word << " ";
            }

            cout << endl << endl;
```

```
            vector<string> answer_string =  boggleBoard(myboard, mywords);

            for(auto word: answer_string){
                cout << word << " ";
            }

            cout << endl << endl;

        }

        else if(option == 2){
            cout <<"Given a 2-
d array/matrix of potentially unequal height and width containing letters" << end
l;
            cout << "this matrix is called boggle board" << endl;
            cout << "And we are also given a list of words" << endl;
            cout << "We have to extract all the words contained in the boggle boa
rd and which are present in the list" << endl;

            cout << endl;
            cout << "A word is constructed in the boggle board by traversing from
 a letter horizontally,vertically or diagonally in any direction without using an
y single letter" << endl;
            cout << "at a given position more than once" << endl;
            cout << "Two or more words can overlap and use the same letters in th
e boggle board" << endl << endl << endl;

        }


        else if(option == 3){
            cout << "Please check all the boogle boards" << endl << endl;

            for(int s =0;s<5;s++){
                vector<vector<char>> myboard1 = select_set(s);
                vector<string> mywords1 = select_word(s);
                cout << "This is the set number" << s << endl << endl;

                for(vector<char> board_i1 : myboard1){
                    for(auto k : board_i1){
                        cout << k << " " ;
                    }
                    cout << endl;

                }

                cout << endl << endl;
                //display the word set

                for(auto word : mywords1){
                    cout << word << " ";
                }

                cout << endl << endl;
            }

            cout << endl;
        }

        else if(option == 4){
```

```
            cout << "Glad you Enjoyed :)" << endl;
            end = true;

        }

        else{
            cout << "You have entered an incorrect choice..Please Try Again" << e
ndl << endl;
        }




    }

}
```

Output :

```
.....WELCOME TO THE BOGGLE BOARD.....

There are 5 sets of board
1. Start the game
2. Read the instructions
3. See the board sets
4. End the game

1


Please select the set from set 0 to set 4
0


t h i s i s a
s i m p l e x
b x x x x e b
x o g g l x o
x x x D T r a
R E P E A d x
x x x x x x x
N O T R E - P
x x D E T A E


this is not a simple boggle board test REPEATED NOTRE-PEATED

NOTRE-PEATED is simple a this boggle board
```


There are 5 sets of board
1. Start the game
2. Read the instructions
3. See the board sets
4. End the game
3
Please check all the boogle boards
This is the set number0
t h i s i s a
s i m p l e x
b x x x x e b
x o g g l x o
x x x D T r a
R E P E A d x
x x x x x x x
N O T R E - P
x x D E T A E
this is not a simple boggle board test REPEATED NOTRE-PEATED
This is the set number1
y g f y e i
c o r p o u
j u z s e l
s y u r h p
e a e g n d
h e l s a t
san sana at vomit yours help end been bed danger calm ok chaos complete rear going storm face epual dangerous

```
This is the set number2

c o m
r p l
c i t
o a e
f o d
z r b
g i a
o a g
f s z
t e i
t w d


commerce complicated twisted zigzag comma foobar baz there

This is the set number3

f t r o p i k b o
r w l p e u e a b
j o t s e l f l p
s z u t h u o p i
k a e g n d r g a
h n l s a t e t x


frozen rotten teleport city zutgatz kappa before rope obligate annoying

This is the set number4

a b
c d


abcd abdc acbd acdb adbc adcb abca


There are 5 sets of board
1. Start the game
2. Read the instructions
3. See the board sets
4. End the game

4
Glad you Enjoyed :)
```