

Data Structures

Lab Project (Semester-2)

**DICTIONARY SEARCHING AND SORTING**

**Members:**

**<Priyanka Mohanty> (B120047)**

**< Raj Aryan> (B120048)**

**International Institute of Information Technology**

**Bhubaneswar, India**

Contents

[Title of Project 3](#_Toc81067646)

[Description of Project 3](#_Toc81067647)

[Code 3](#_Toc81067648)

[Sample Outputs 3](#_Toc81067649)

[Contribution by group members 3](#_Toc81067650)

[References 3](#_Toc81067651)

# **Title of Project**

Dictionary searching and sorting

# **Description of Project**

o A Dictionary stores keywords & their meanings. Create a notepad file which stores 100 keywords and their meanings.

o Provide facility for adding new keywords, deleting keywords, updating values of any entry.

o Provide facility to display whole data sorted in ascending/descending order.

o Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.

# Code

# #include <stdio.h>

# #include <stdlib.h>

# #include<string.h>

# struct dict {

# struct dict \*left,\*right;

# char word[20],meaning[20];

# }\*Root=NULL;

# typedef struct dict dictionary;

# int check(char[],char[]);

# void insert(dictionary \*);

# // void Search();

# // void view(dictionary \*);

# /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# int check(char a[],char b[]){

# int i,j,c;

# for(i=0,j=0 ; a[i]!='\0'&&b[j]!='\0' ; i++,j++){

# if(a[i]>b[j]){

# c=1;

# break;

# }

# else if(b[j]>a[i]){

# c=-1;

# break;

# }

# else

# c=0;

# }

# if(c==1)

# return 1;

# else if(c==-1)

# return -1;

# else

# return 0;

# }

# /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# // ADD A WORD

# void insert(dictionary \*temp){

# int flag=0;

# dictionary \*ptr,\*par;

# ptr=Root;

# if(Root==NULL)

# Root=temp;

# else{

# while(ptr!=NULL ){

# if(check(temp->word,ptr->word)>0){

# par=ptr;

# ptr=ptr->right;

# }

# else if(check(temp->word,ptr->word)<0)

# {

# par=ptr;

# ptr=ptr->left;

# }

# else if(check(temp->word,ptr->word)==0){

# flag=1;

# // printf("\nWord already exists!\n");

# break;

# }

# }

# if(flag==0 && ptr==NULL){

# if(check(par->word,temp->word)==1)

# par->left=temp;

# else if(check(par->word,temp->word)==-1)

# par->right=temp;

# }

# }

# }

# /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# // DISPLAY THE WORDS

# void viewTree(dictionary \*ptr) { // INORDER TRAVERSAL

# if(Root==NULL)

# printf("\nEmpty tree\n");

# else if(ptr !=NULL) {

# viewTree(ptr->left);

# printf("\n%s - %s", ptr->word, ptr->meaning);

# viewTree(ptr->right);

# }

# }

# // 3.VIEW FROM FILE

# void view(){

# FILE \*fp;

# char str[20]={'\0'};

# fp = fopen("dictionary.txt", "r");

# while(fgets(str, 60, fp) != NULL) {

# if(str == '\0')

# printf("\nEmpty dictionary\n");

# else {

# printf("%s", str);

# // CODE TO SEPARATE WORD AND MEANING FROM A SINGLE STRING

# char w[60]={'\0'}, m[60]={'\0'}; int i=0;

# while (str[i] != ' ') {

# ++i;

# }

# for (size\_t j = 0; j < i; j++) {

# w[j] = str[j];

# }

# // printf("\n");

# for (size\_t j = i+3; j < strlen(str)-1; j++) {

# m[j-i-3] = str[j];

# }

# // CODE TO SEPARATE WORD AND MEANING FROM A SINGLE STRING

# // CODE TO RECREATE BINARY SEARCH TREE

# dictionary \*temp = (dictionary\*)malloc(sizeof(dictionary));

# temp->left=NULL;

# temp->right=NULL;

# strcpy(temp->word, w);

# strcpy(temp->meaning, m);

# // printf("%s ---> %s", temp->word, temp->meaning);

# insert(temp);

# // CODE TO RECREATE BINARY SEARCH TREE

# }

# }

# fclose(fp);

# fp = NULL;

# }

# // VIEW FILE IN REVERSE ORDER

# void viewReverse() {

# FILE \*fp;

# fp=fopen("dictionary.txt","r");

# char arr[60][60]={'\0'};

# int count=0;

# if(fp==NULL) {

# printf("File does not exist..");

# }

# else {

# int i=0;

# char str[60];

# while (fgets(str, 60, fp) != NULL) {

# // printf("%s", strrev(str));

# strcpy(arr[i], str);

# // printf("%s", arr[i]);

# ++i; ++count;

# }

# }

# // printf("\n%d", count);

# for (size\_t j = count; j > 0; j--) {

# printf("%s", arr[j]);

# }

# printf("%s", arr[0]);

# }

# /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# // UPDATE A WORD

# void update() {

# // CODE GOES HERE

# int flag=0;

# dictionary \*ptr;

# ptr=Root;

# char w[20]; // ENTER WORD TO SEARCH

# printf("\nEnter word: ");

# scanf("%s",w);

# while(ptr!=NULL && flag==0){

# if(check(w,ptr->word)>0)

# ptr=ptr->right;

# else if(check(w,ptr->word)<0)

# ptr=ptr->left;

# else if(check(w,ptr->word)==0){

# flag=1;

# // printf("\n%s",ptr->meaning);

# char newMeaning[60];

# printf("Enter new meaning: ");

# scanf("%s", newMeaning);

# strcpy(ptr->meaning, newMeaning);

# printf("\nWord updated successfully!");

# }

# }

# if(flag==0)

# printf("\nWord not found");

# }

# /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# // CODE TO SYNC FILE AND BST

# void updateFile(dictionary \*ptr) {

# if(Root==NULL)

# printf("\nEmpty dictionary\n");

# else if(ptr !=NULL) {

# updateFile(ptr->left);

# FILE \*fp;

# fp = fopen("dictionary.txt", "a");

# if(fp == NULL) {

# printf("file cannot be opened!");

# // return (-1);

# }

# if(fp != NULL) {

# char str[60], str2[60];

# int isThere=0;

# strcpy(str, ptr->word);

# strcat(str, " - ");

# strcat(str, ptr->meaning);

# while(fgets(str2, 60, fp) != NULL) {

# if(strcmp(str, str2) == 0) {

# isThere = -1; // ALREADY THERE

# }

# }

# if(isThere == 0) {

# fputs(str, fp);

# }

# fputs("\n", fp);

# }

# fclose(fp);

# fp = NULL;

# updateFile(ptr->right);

# }

# }

# /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# // DELETE A WORD

# dictionary \*minValue(dictionary \*ptr) {

# dictionary \*current = ptr;

# while (current && current->left != NULL) {

# current = current->left;

# }

# return current;

# }

# dictionary \*deleteNode(dictionary \*root, char key[]) {

# // printf("Word found! - %s\n", key);

# if(root == NULL) {

# return root;

# }

# if(root->left == NULL && root->right == NULL) {

# Root = NULL;

# return root;

# }

# if(check(key, root->word) < 0) {

# root->left = deleteNode(root->left, key);

# } else if(check(key, root->word) > 0) {

# root->right = deleteNode(root->right, key);

# }

# // FOUND WORD TO DELETE

# else {

# if(root->left == NULL) {

# dictionary \*temp = root->right;

# free(root);

# return temp;

# } else if(root->right == NULL) {

# dictionary \*temp = root->left;

# free(root);

# return temp;

# }

# dictionary \*temp = minValue(root->right);

# strcpy(root->word, temp->word);

# root->right = deleteNode(root->right, temp->word);

# }

# return root;

# }

# void deleteWord() {

# FILE \*fp;

# fp = fopen("dictionary.txt", "r");

# if(fp == NULL) {

# printf("file cannot be opened!");

# return;

# }

# char str[60], w[60], helpstr[60];

# printf("\nEnter word to delete: ");

# scanf("%s", w);

# while (fgets(str, 60, fp) != NULL) {

# // printf("%s", str);

# // CODE TO SEPARATE WORD

# for (size\_t i = 0; i < strlen(str)-1; i++) {

# if(str[i] == ' ') {

# break;

# }

# helpstr[i] = str[i];

# }

# // CODE TO CHECK IN FILE IF PRESENT

# if(check(helpstr, w) == 0) {

# // printf("Word found! - %s", w);

# deleteNode(Root, w);

# // updateFile(Root);

# break;

# }

# }

# fclose(fp);

# fp = NULL;

# }

# /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# int main(int argc, char const \*argv[]) {

# int ch;

# dictionary \*temp;

# while(ch!=8){

# printf("\n1.Update list\n2.Insert\n3.View from file\n4.Update word\n5.View Tree\n6.Delete a word\n7.View reverse\n8.Exit\nEnter choice: ");

# scanf("%d",&ch);

# switch (ch) {

# case 1: {

# FILE \*fp;

# fp = fopen("dictionary.txt", "w");

# if(fp != NULL) {

# fputs("", fp);

# }

# fclose(fp);

# fp = NULL;

# updateFile(Root);

# printf("\nFile updated successfully!\n"); break;

# }

# case 2:

# temp=(dictionary\*)malloc(sizeof(dictionary));

# temp->left=NULL;

# temp->right=NULL;

# printf("\nInsert word: ");

# scanf("%s",temp->word);

# printf("Insert meaning: ");

# scanf("%s",temp->meaning);

# insert(temp);

# break;

# case 3: {

# printf("\n");

# view(); break;

# }

# case 4: update(); printf("\n"); break;

# case 5: viewTree(Root); printf("\n"); break;

# case 6: printf("\n"); deleteWord(); break;

# case 7: {

# printf("\n");

# viewReverse();

# printf("\n"); break;

# }

# case 8: exit(0);

# // default: printf("\nWrong choice!\n");

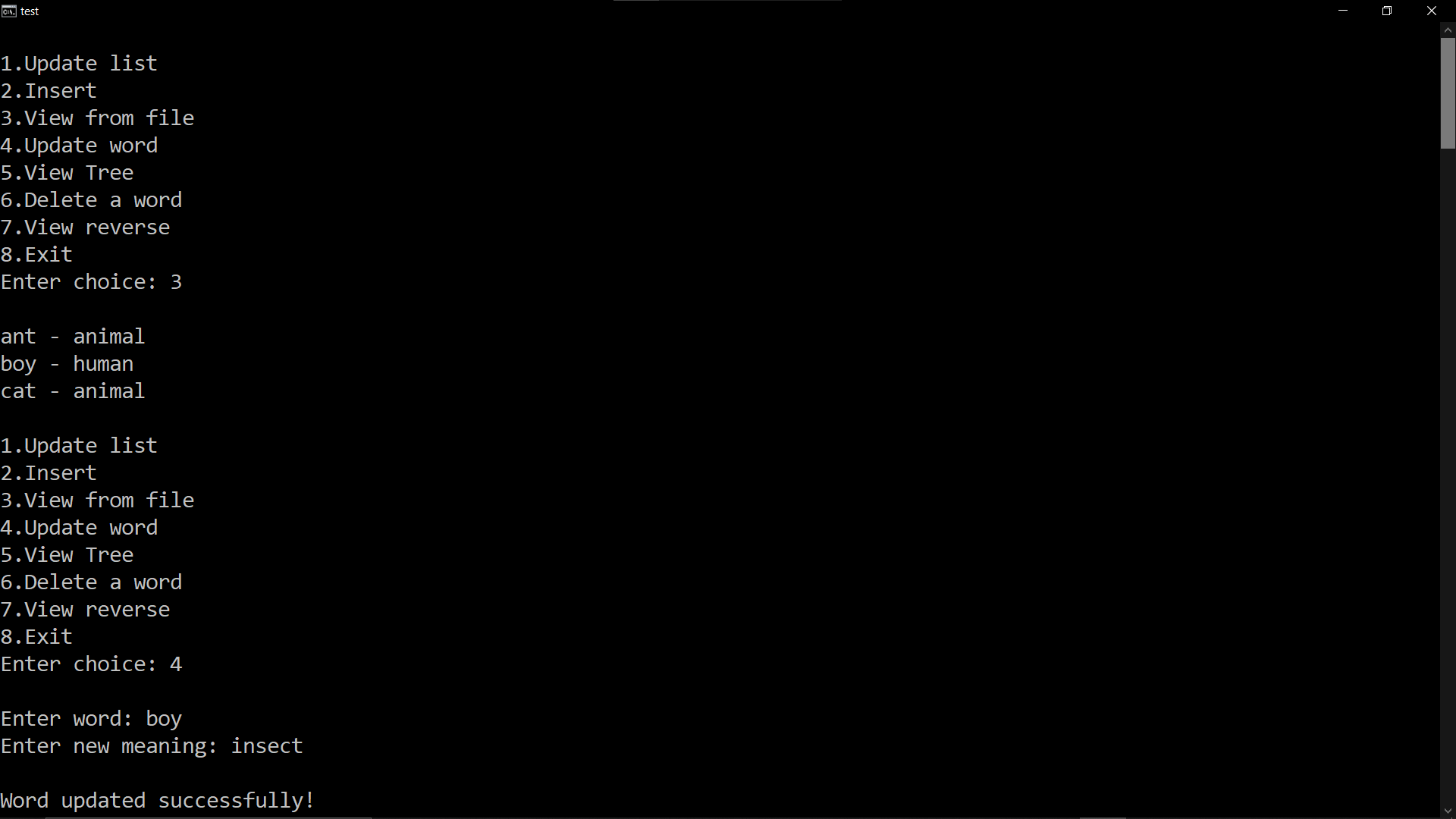
# }

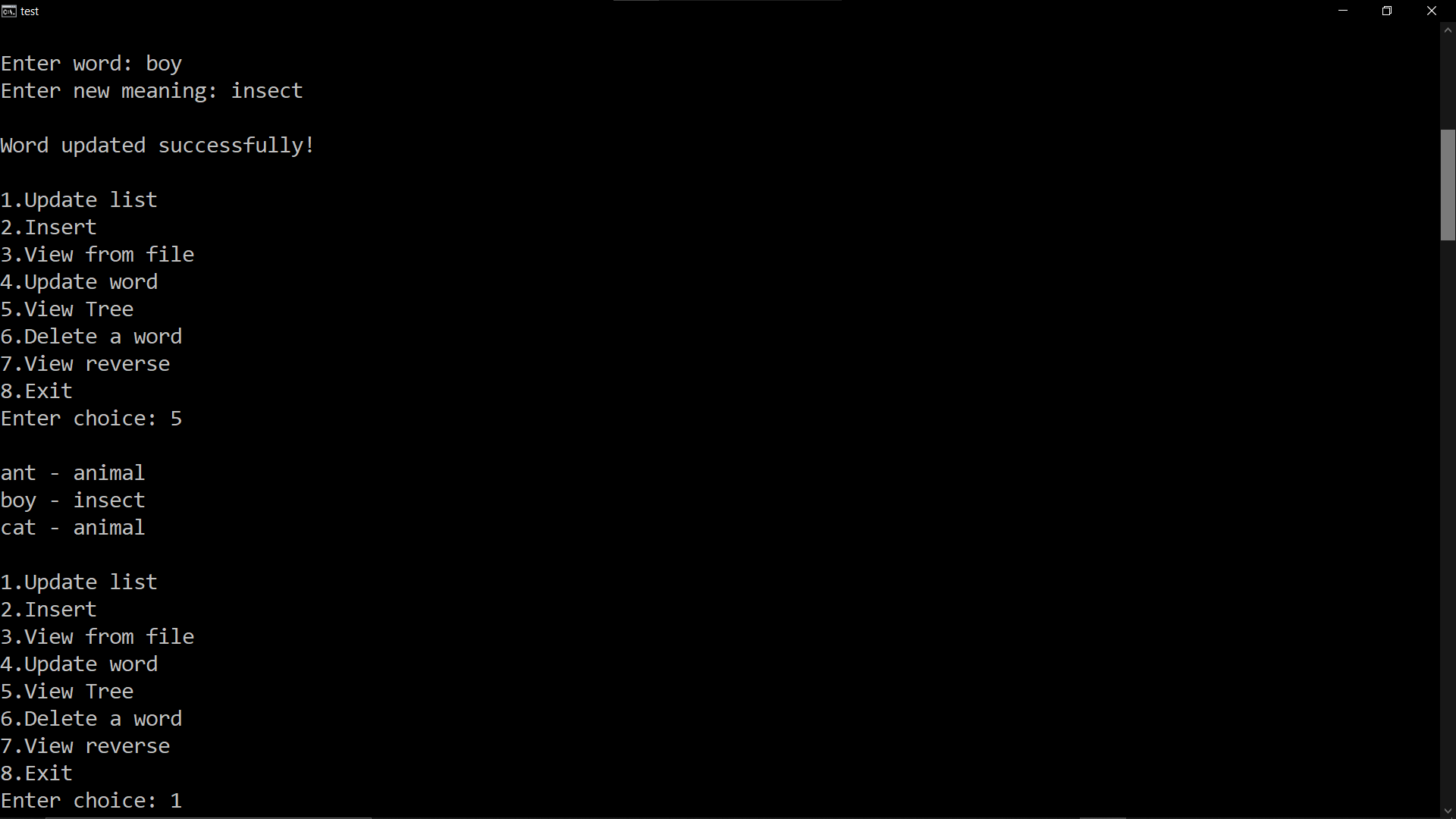
# }

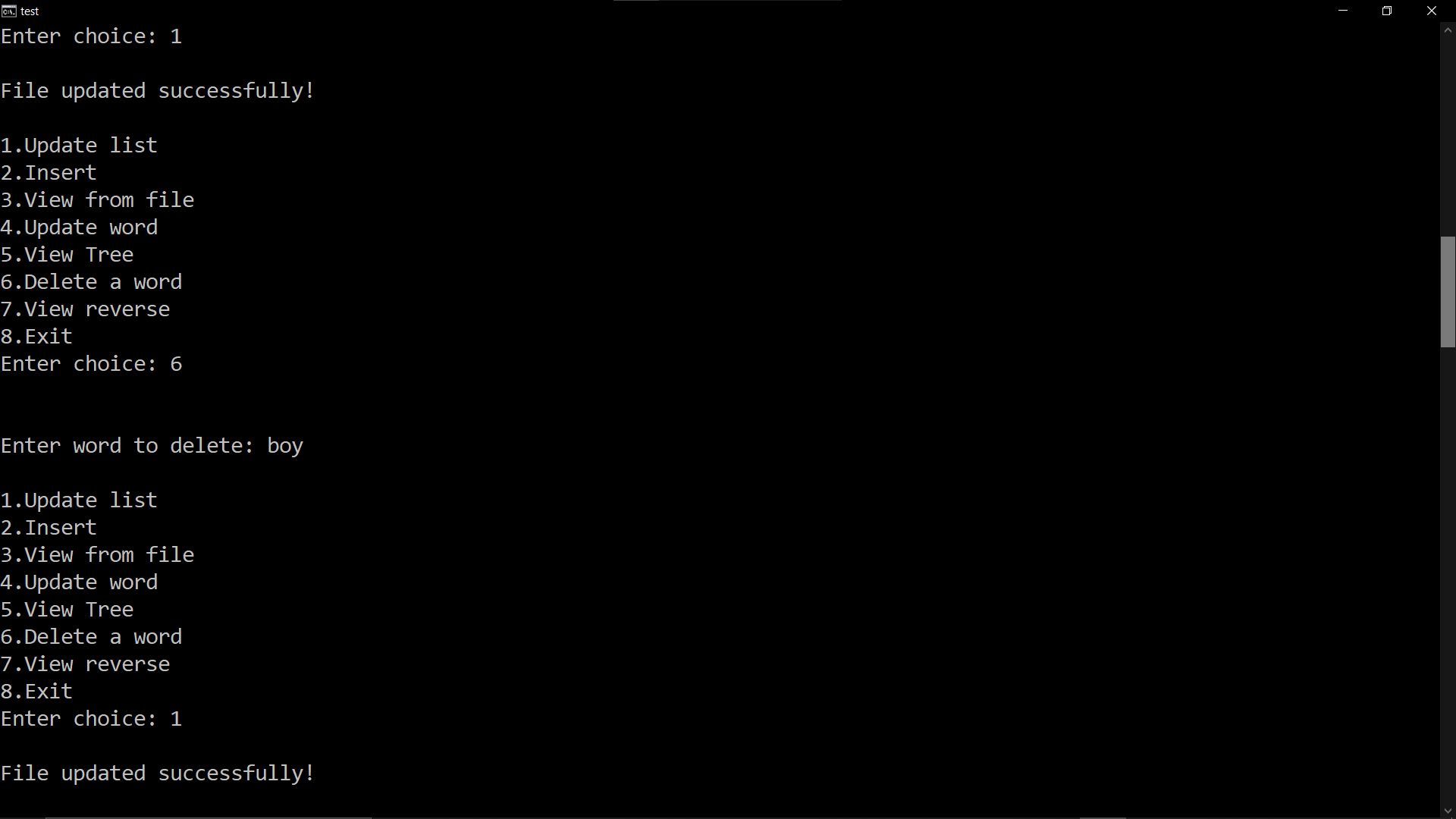
# return 0;

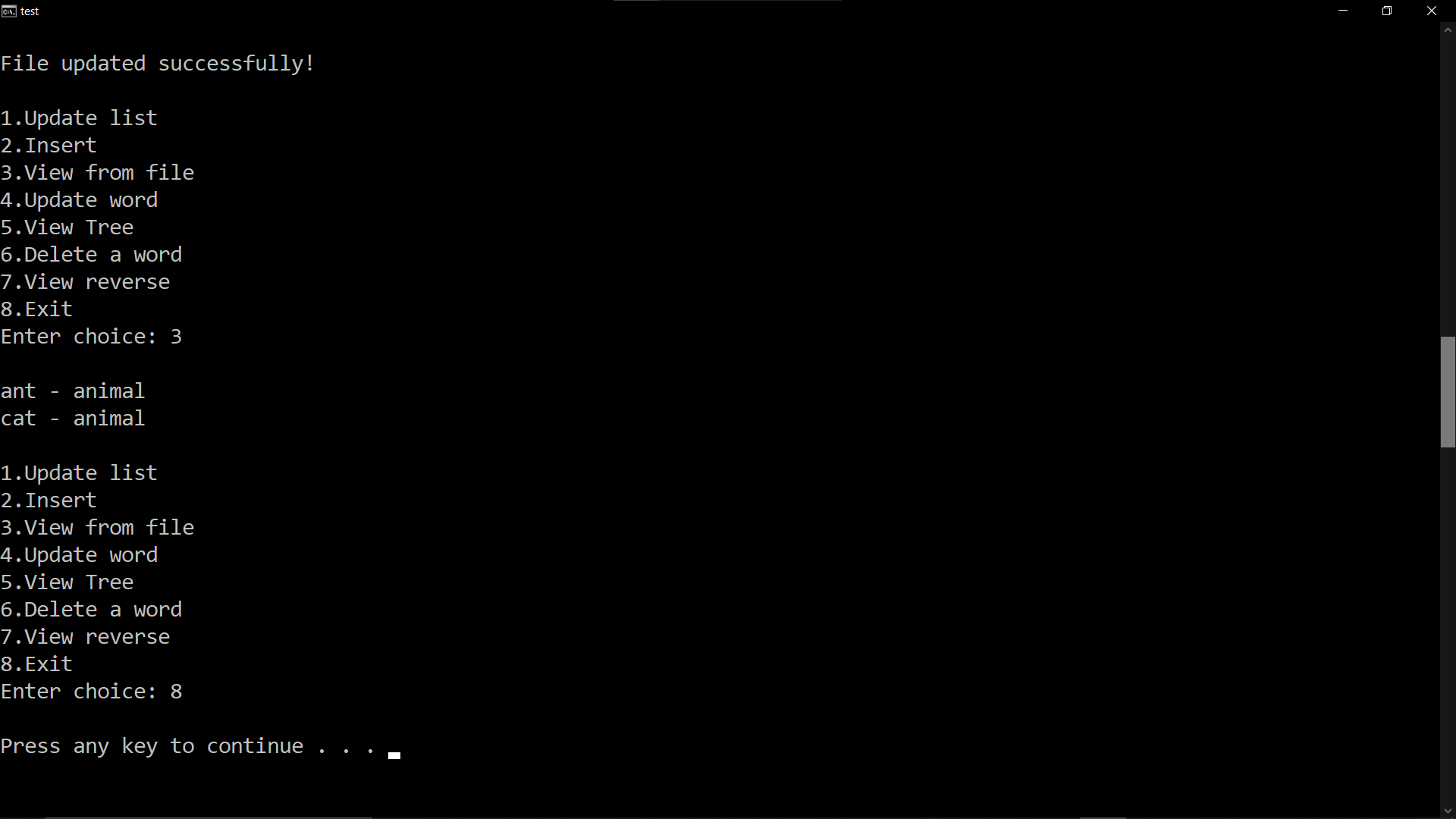
# }

# Sample Outputs









# Contribution by group members

<B120047>: In writing the code to make BST from words of text file.

<B120048>: For writing the different operations in the dictionary.

# References

YouTube

Geeksforgeeks

Github