**K L E F deemed to be University**

**Green Fields, Vaddeswaram, Guntur (Dt) :: 522502**

**Department of Computer Science Engineering**

**Project Based Lab Report**

**SKILLING FOR ENGINEERS**

**17 TS 201**

****

**Submitted by**

K. SUDHARSAN REDDY- 170030689

B. GANESH BABU - 170030120

P. SOBHANA - 170031015

U. MAHESH - 170031326

**Under Guidance of**

Mr. G. Soma Sekhar

**II Year Engineering Course Work**

**2018-19**



**DECLARATION**

We declare that the project work entitled “**WORD FREQUENCY COUNT APPLICATION** “was carried out by us during 2018-19, and this work is not the same as that of any other and has not been submitted for award of any other degree/diploma

Place: Vaddeswaram Signature of the Student

Date: 14/11/2018

****

**K L E F deemed to be UNIVERSITY**

**Green fields, Vaddeswaram, Guntur Dist**.

**CERTIFICATE**

This is to certify that this project work entitled “**WORD FREQUENCY COUNT APPLICATION**” by is a bonafide work carried out by them in Department of Computer Science and Engineering.

Project supervisor Head of the Department

Mr. G. Soma Sekhar Dr. V. Hari Kiran

****

**ACKNOWLEDGEMENT**

We express my sincere gratitude to **Mr. G. Soma Sekhar** sir for encouraging and guiding us to undertake this project work. We express my deep sense of gratitude to **Dr. V. Hari Kiran** sir and our beloved course lecturers of department for their encouragement.

Place: Vaddeswaram

Date: 14/11/2018

K. SUDHARSAN REDDY- 170030689

B. GANESH BABU - 170030120

P. SOBHANA - 170031015

U. MAHESH - 170031326

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Topic** | **Page No** |
| 1 | ABSTRACT | 6 |
| 2 | INTRODUCTION | 7 |
| 3 | FUNCTIONAL REQUIREMENTS | 8 |
| 4 | NON-FUNCTIONAL REQUIREMENTS | 9 |
| 5 | SOURCE CODE | 10-18 |
| 6 | OUTPUT SCREENS | 19-21 |
| 7 | CONCLUSION | 22 |

**ABSTRACT**

The aim of this project is meant for creating an AVL tree application that uses a tree structure containing all of the words in a document, with a count of the number of times each word is used. Each entry in the AVL tree contains a word from the document and a pointer to an integer that contains a count of the number of times the word appears in the document.

The project mainly consists of the different functions mentioned below, they are:

Counts the words in a file, reads file and creates AVL tree containing list of all words used in the file with count of the number of times each word is found in the file, Reads one word from file, compares two integers identified by pointers to integers, Prints the list with the count for each word, Prints one word from the list with its count.

**INTRODUCTION**

In computer science, an AVL tree (named after inventors Adelson-Velsky and Landis) is a self-balancing binary search tree. It was the first such data structure to be invented. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion, and deletion all take O(log n) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

The AVL tree is named after its two Soviet inventors, Georgy Adelson-Velsky and Evgenii Landis, who published it in their 1962 paper "An algorithm for the organization of information".

AVL trees are often compared with red–black trees because both support the same set of operations and take O(log n) time for the basic operations. For lookup-intensive applications, AVL trees are faster than red–black trees because they are more strictly balanced. Similar to red–black trees, AVL trees are height-balanced. Both are, in general, neither weight-balanced nor μ-balanced for any μ ≤ 1/2 that is, sibling nodes can have hugely differing numbers of descendants..

**FUNCTIONAL REQUIREMENTS**

* **SOFTWARE REQUIREMENTS:**

The major software requirements of the project are as follows:

Language : C Language

Operating system **:**  windows 10

C Compiler : Dev C++

* **HARDWARE REQUIREMENTS:**

The hardware requirements that map towards the software are as follows:

RAM : 8 GB

Processor : Intel Core i7

Hard Disk : 1 TB

**Module Identification:**

Module 1: -

Counts the words in a file.

Module 2: -

Reads file and creates AVL tree containing list of all words used in the file with count of the number of times each word is found in the file.

Module 3: -

Reads one word from file.

Module 4: -

Compares two integers identified by pointers to integers.

Module 5: -

Prints the list with the count for each word.

Module 6: -

Prints one word from the list with its count.

**NON-FUNCTIONAL REQUIREMENTS**

**NESTED IF:**

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

When using if...else if..else statements, there are few points to keep in mind

• An if can have zero or one else's and it must come after any else if's.

• An if can have zero to many else if's and they must come before the else.

• Once an else if succeeds, none of the remaining else if's or else's will be tested.

**WHILE LOOP :**

Syntax

The syntax of a while loop in C programming language is −

while(condition) {

statement(s);

}

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.

**SWITCH CASE:**

In computer programming languages, a switch statement is a type of selection control mechanism used to allow the value of a variable or expression to change the control flow of program execution via a multiway branch.

**SOURCE CODE**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct Node {

char \*s;

struct Node \*left;

struct Node \*right;

int height;

int count;

};

struct node{

char \*s;

};

int count(char s[],char word[]) {

char temp[1000]="";

int j=0,c=0;

int i;

for(i=0;s[i]!='\0';i++) {

if(s[i]==' '||s[i]=='\0'){

temp[i]='\0';

if(strcmp(word,temp)==0)

c++;

j=0;

}

else{

temp[j]=s[i];

j++;

}

}

return c;

}

int height(struct Node \*N) {

if(N==NULL) return 0;

return N->height;

}

int max(int a,int b) {

return (a>b)?a:b;

}

struct Node\* newNode(char \*s,int count) {

struct Node\*node=(struct Node\*)malloc(sizeof(struct Node));

node->s=s;

node->count=count;

node->left= NULL;

node->right=NULL;

node->height=1;

return(node);

}

struct Node \*rightRotate(struct Node \*y) {

struct Node \*x=y->left;

struct Node \*T2=x->right;

x->right=y;

y->left=T2;

y->height=max(height(y->left),height(y->right))+1;

x->height=max(height(x->left),height(x->right))+1;

return x;

}

struct Node \*leftRotate(struct Node \*x) {

struct Node \*y=x->right;

struct Node \*T2=y->left;

y->left=x; x->right=T2;

x->height=max(height(x->left),height(x->right))+1;

y->height=max(height(y->left),height(y->right))+1;

return y;

}

int getBalance(struct Node \*N) {

if (N==NULL) return 0;

return height(N->left)-height(N->right);

}

struct Node\* insert(struct Node\* node, char \*s,int count) {

if (node==NULL)

return(newNode(s,count));

if (strcmp(s,node->s)<0)

node->left=insert(node->left,s,count);

else if (strcmp(s,node->s)>0)

node->right=insert(node->right,s,count);

else return node;

node->height=1+ max(height(node->left),height(node->right));

int balance=getBalance(node);

if(balance>1&&strcmp(s,node->left->s)<0)

return rightRotate(node);

if(balance<-1&&strcmp(s,node->right->s)>0)

return leftRotate(node);

if(balance>1&&strcmp(s,node->left->s)>0) {

node->left=leftRotate(node->left);

return rightRotate(node);

}

if(balance<-1&&strcmp(s,node->right->s)<0) {

node->right=rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void inOrder(struct Node \*root) {

if(root != NULL) {

inOrder(root->left);

printf("%s : %d\n", root->s,root->count);

inOrder(root->right);

}

}

int main(){

{

struct Node \*root=NULL;

int func,cc=0;

FILE \*fptr;

char s[1000],word[100];

char temp[100][100];

while(1){

printf("\n------------------------------------------------------\n");

printf("1.Reads file and creates AVL tree containing list of all words used in the file with count of the number of times each word is found in the file. \n");

printf("2.Reads one word from file.\n");

printf("3.Compares two integers identified by pointers to integers.\n");

printf("4.Display top element\n");

printf("5.Prints the list with the count for each word.\n");

printf("6:Prints one word from the list with its count.\n");

printf("7.Exit function\n");

printf("\n------------------------------------------------------\n");

printf("Enter function number:");

scanf("%d",&func);

printf("\n");

if(func==7) break;

switch(func){

case 1:{

fptr=fopen("skills33.txt","r");

while(!feof(fptr)){

fgets(s,1000,fptr);

}

int j=0,ctr=0;

int i;

for(i=0;s[i]!='\0';i++) {

if(s[i]==' ') {

temp[ctr][j]='\0';

ctr++;

j=0;

}

else {

temp[ctr][j]=s[i];

j++;

}

}

for(i=0;i<ctr;i++) {

int c=count(s,temp[i]);

root=insert(root,temp[i],c);

}

printf("Data in file is \"%s\"\n",s);

printf("AVL tree is created\n");

printf("Go to function 5 to check whether AVL tree is created or not\n");

break;

}

case 2:{

fptr=fopen("skills33.txt","r");

while(!feof(fptr)){

fgets(s,1000,fptr);

}

printf("Enter the word to find :");

scanf("%s",word);

int j=0,ctr=0;

cc=count(s,word);

if(cc==0){

printf("Word is present in the FILE \n");

printf("Go to function 6 to check how many times the word is repeated in the file\n");

}

else{

printf("Word is not found in the FILE\n");

printf("If you are not yet satisfied with our project then proceed with another function:\n");

}

printf("\n");

break;

}

case 3:

{

int m=0;

char w[100];

fptr=fopen("skills33.txt","r");

while(!feof(fptr)){

fgets(s,1000,fptr);

}

int i,j=0,ctr=0;

for(i=0;s[i]!='\0';i++) {

if(s[i]==' ')

{

temp[ctr][j]='\0';

ctr++;

j=0;

}

else

{

temp[ctr][j]=s[i];

j++;

}

}

for(i=0;i<ctr;i++)

{

int c=count(s,temp[i]);

if(c>m){

m=c;

strcpy(w,temp[i]);

}

}

printf("The most frequent word in the file is \'%s\' and it is repeated \'%d\' times\n\n",w,m);

printf("If you are not yet satisfied with our project then proceed with another function:\n");

break;

}

case 4:

{

if(root==NULL)

printf("AVL tree is not constructed\n");

else

printf("The top element is: \"%s\" and its count is \'%d\' \n",root->s,root->count);

printf("\n");

printf("If you are not yet satisfied with our project then proceed with another function:\n");

break;

}

case 5:

{

if(root==NULL)

printf("AVL tree is not constructed\n");

else

{

printf("Words in the file and their count is :\n");

inOrder(root);

}

printf("\n");

printf("If you are not yet satisfied with our project then proceed with another function:\n");

break;

}

case 6:

{

printf("Word \'%s\' is found \'%d\' times in the file\n\n",word,cc);

printf("If you are not yet satisfied with our project then proceed with another function:\n");

break;

}

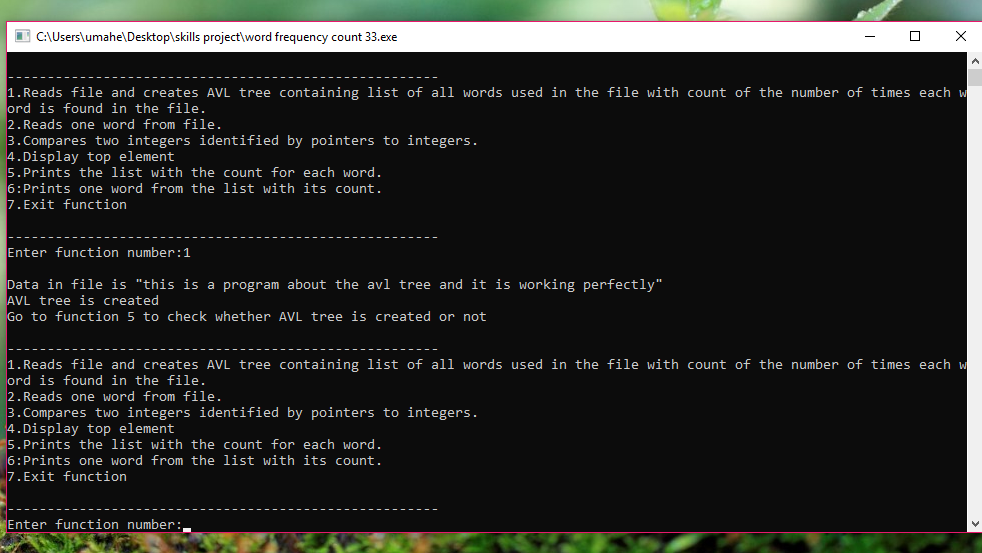
}

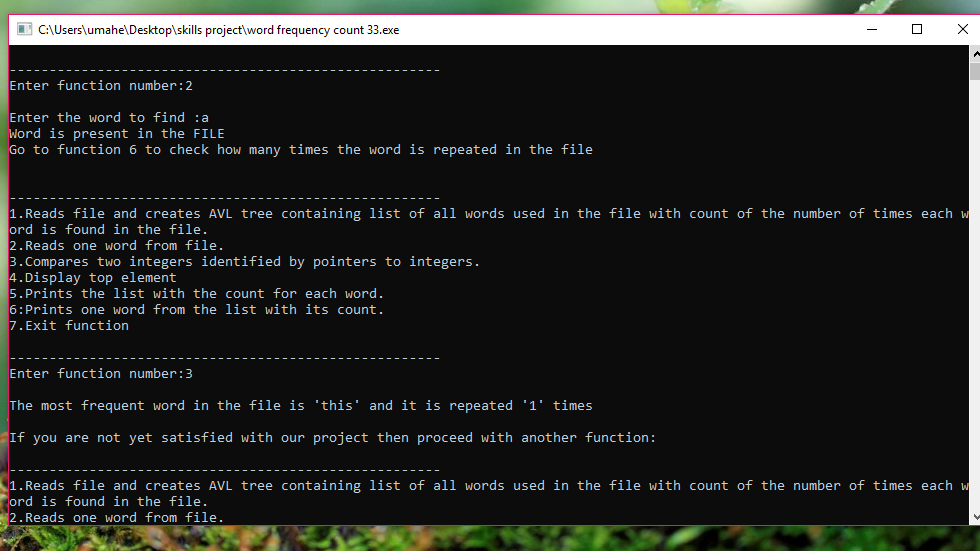
}

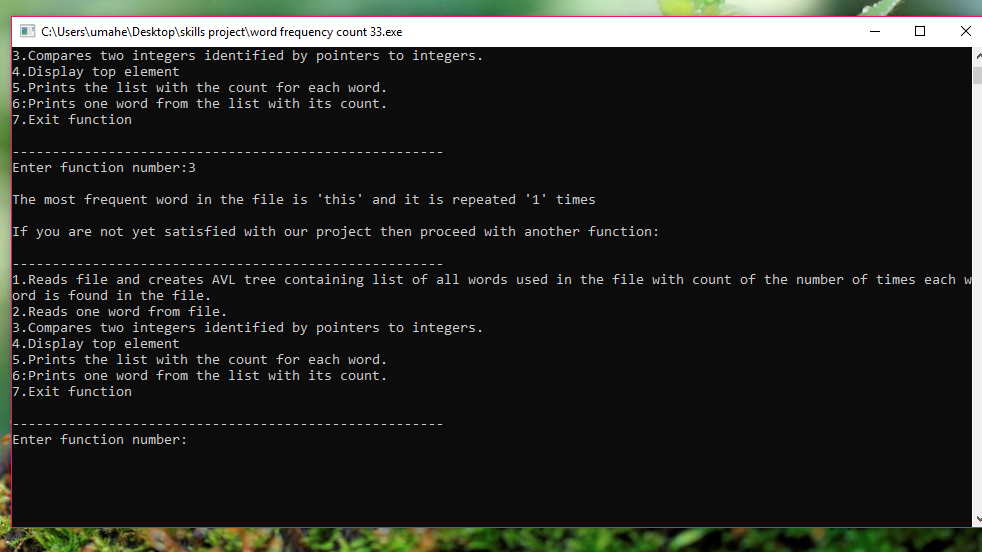
}

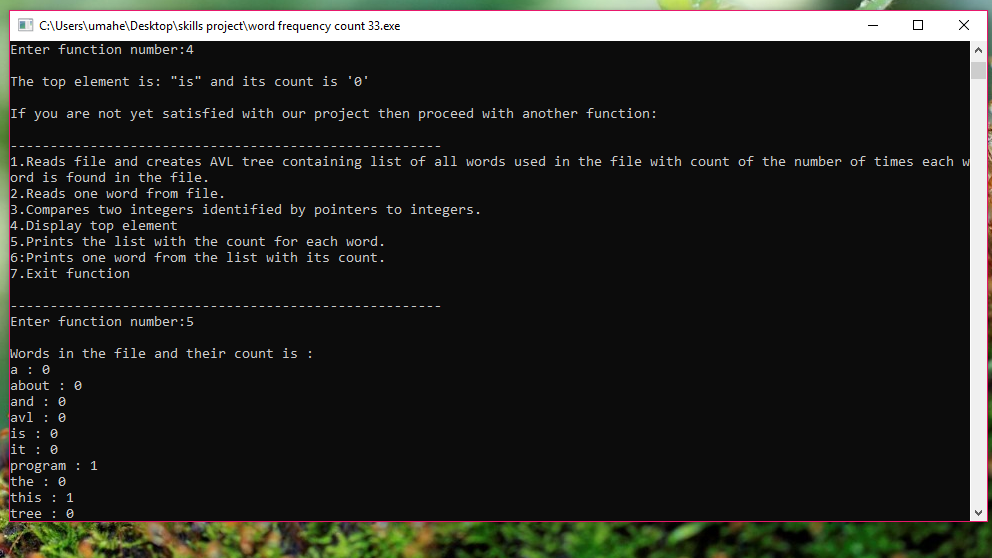
}

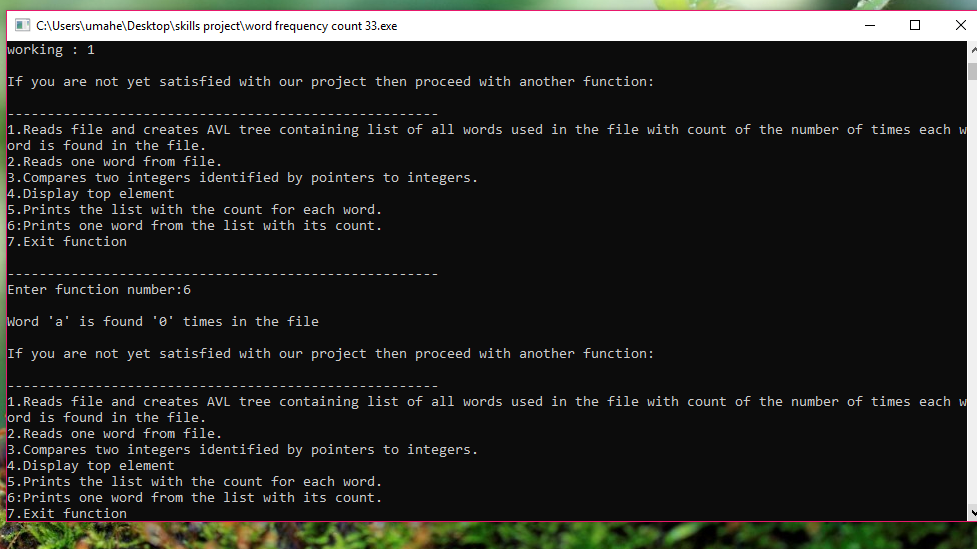
**OUTPUT SCREENS**

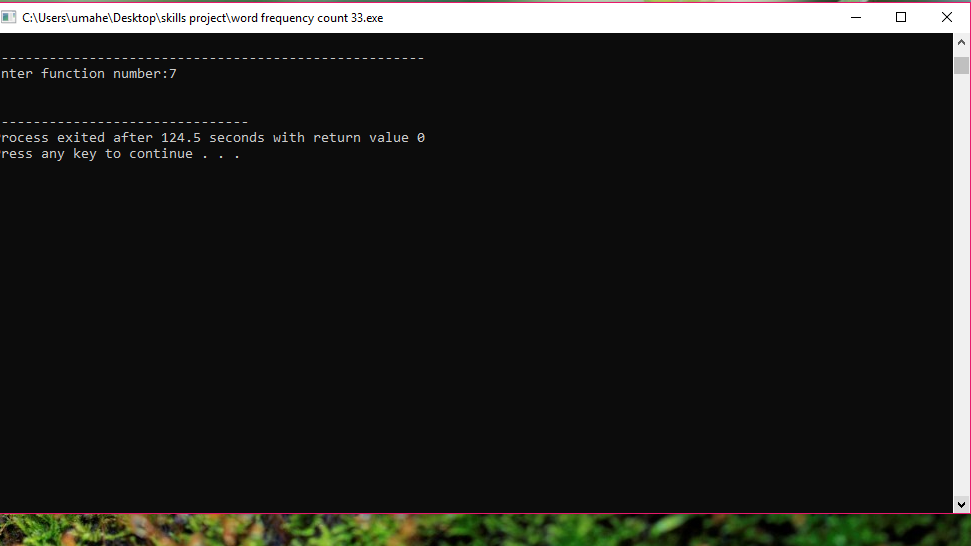
****

****

****

****

****

****

**CONCLUSION**

It is always prudent to opt for a student information system that is designed using modern system architecture to cope with changing requirements. This system should encompass very solid information coding and distinctly outlined business applications, separating the presentation of details and methods of support.