

**Submitted To: Submitted By:**

IBM Roshan Gulgulia

UID:15BCS7051

CSE(Hons) Cloud

**DEPARTMENT OF APPLIED SCIENCE & ENGINEERING**

**CHANDIGARH UNIVERSITY**

**GHARUAN, MOHALI, PUNJAB, INDIA-140413**

**2015**

TABLE OF CONTENT

Acknowledgement 3

Brief Discussion of Project: 4

Header Files Used 7

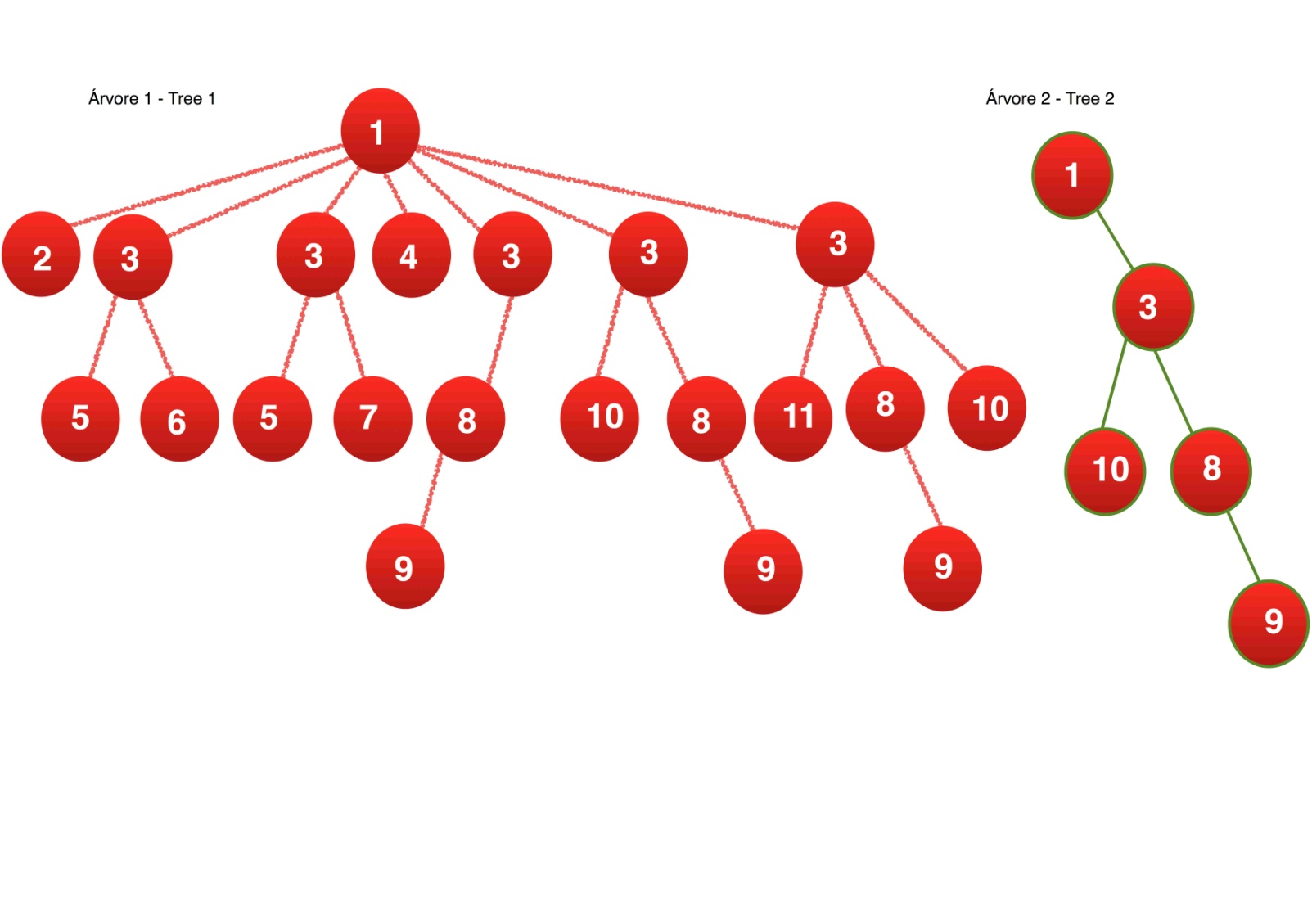
Pre Defined Function 8

User Defined Function 9

Coding 10

Screen Short 33

Conclusion 37

[](https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjQ1ruTupjMAhUP72MKHbZnD_0QjRwIBw&url=http://stackoverflow.com/questions/27239335/how-to-check-if-a-non-binary-tree-is-a-subtree-of-another&bvm=bv.119745492,d.cGc&psig=AFQjCNHm0cSU7mc4YiX0ttmPaFP3_0Aoqw&ust=1461078054988590)

ACKNOWLEDGEMENT

I am using this opportunity to show my project to everyone. who supported me throughout the project. I am thankful for their aspiring guidance, invaluably constructive criticism and friendy advice during the project work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

I Really thankful to NPTEL(online IIT Professors) they teaches me the data structure and algorithm by those concepts i made this program.

I express my warm thanks to our IBM teachers and MY SF Teacher for their support and guidance in my Project

**Brief Discussion of project**

This is the fastest searching words project. In this project i am using data structures a topic is their name BINARY TREE this is called as fastest searching inserting and deleting algorithm.

Why this is fastest searching Dictonary?

This is because i have used a head node and 26 alphabet a,b,c..z

as sub node and again in this 26 subnode i have again made 26 subnode as per my requirement this concept goes on.

so when we we will search a word name Roshan then its time complexity will be as pper the no of letters and if 1000 or 5000 words are save then aslo it will take only 6 steps to search word Roshan.

And other then that if we use array or simple linked list without the concept of trees then its searching time complexity will be as per no of words are saved in dictonary.

It Goes on as per our Requirement

Sub Node and it goes upto Z

Head Node

Now For example i have to save ROSHAN.

Then it will goto headnode and checks for R then in r it will goto s then it will goto h then a then n and in N it will save Roshan. with its meaning. so inserting is also the big O(n) where n is the no of letters.

Here Word Roshan

and its Meaning will be save

I have used LINKED LIST concept for binary tree and and as per requirement it creates the new new nodes so as to saves the memory it creates only the memory where we have to save the word and meaning.

I have use 5 switch cases 1. Adding Words

2. See All Words

1. All Words

2. Search By Letter

3. Search a Word

4. Delete A Word

5. Exit

All inserting deleting searching have the time complexity O(n).

I have made this Project in DEV C++.

**HEADER FILES**

**iostream:** This is a header file used in my project for cin i.e input and cout i.e. output and any other functions are also their where the function defination is stored in iostream.

stdio.h: This header file i have use for file handlingfor file streaming i.e. my words and meaning are save in text files.

**strings.h**: The string.h header file contanins one variable type,one macro type and several functions for manipulating the char array.

variable type: **size\_t.**

Macro type: NULL.

function type: strncpy,strlen,strcat etc.

**windows.h:** The **windows.h** is a [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows)-specific header file which contains declarations for all of the functions in the [Windows API](https://en.wikipedia.org/wiki/Windows_API), all the common macros used by Windows programmers, and all the data types used by the various functions and subsystems. Here i am using for system clear screen.

**Pre Defined Functions**

**cout<<:** to print in the console screen.

**cin>>:** to read stdin stream.

**fopen():** to open the file.

**fclose():** to close the file.

**fread():** to read the data from the file.

**fwrite():** to write the data into the file.

**exit():** to exit the program.

**system("cls"):** to clear the screen.

**fflush():** to flush the stdio.

**getch():** to get a character from the user.

**strcpy():** to copy the strings.

**strcat():** to Concatenate strings.

**strlen():** to get the string length.

**gets():** to get the string.

**User Defined Function**

**void menu(dictonary \*head):** The Main Menu of my Programwhere it will show the different category like to add word to update word or to search or delete etc.

**dictonary\* Insert(dictonary \*root,char word[],char meaning[]):** this function takes 3 ardument one is head node and other is word and meaning and it also returns the node where the word and meaning is save.

**void search(char name[],dictonary \*root):** it takes two argument one searching name and the root address.

**void delty(char name[],dictonary \*root):** it takes two argument one deleting name and other is head node.

**void write(dictonary \*root):** it takes one argument i.e. head node and this function saves the file in txt format.

**void display(dictonary \*root):** it takes one argument i.e. head node and this function display the words with their meaning.

**void gotoxy():** this function is used to move the curser from one place to other.

**void exits():** To Exit The Program.

**void group():** To View The Groupsbelonging to same state and the same age and the same qualification same company etc.

**CODING**

#include <iostream>

#include<stdio.h>

using namespace std;

#include<string.h>

#include<windows.h>

void gotoxy(short x, short y)

{

COORD pos = {x, y};

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), pos);

}

FILE \*fp;

struct data

{

int letter;

char word[20];

char meaning[100];

}dat;

struct dictonary

{

data dat;

dictonary \*a,\*b,\*c,\*d,\*e,\*f,\*g,\*h,\*i,\*j,\*k,\*l,\*m;

dictonary \*n,\*o,\*p,\*q,\*r,\*s,\*t,\*u,\*v,\*w,\*x,\*y,\*z;

}plo;

int j=0;

dictonary\* newnode(int c)

{

dictonary\* node=new dictonary;

node->a=node->b=node->c=node->d=node->e=node->f=NULL;

node->g=node->h=node->i=node->j=node->k=node->l=NULL;

node->m=node->n=node->o=node->p=node->q=node->r=NULL;

node->s=node->t=node->u=node->v=node->w=node->x=NULL;

node->y=node->z=NULL;

strcpy(node->dat.word,"!@#$");

node->dat.letter=c;

return node;

}

int i=-1;

dictonary\* Insert(dictonary \*root,char word[],char meaning[])

{

i++;

if(i==strlen(word))

{

if(strcmp(word,root->dat.word)==0)

{

cout<<" Word Already Exist"<<endl;

}

else

{

strcpy(root->dat.word,word);

strcpy(root->dat.meaning,meaning);

}

}

else

{

switch(word[i])

{

case 'a':

case 'A':

if(root->a==NULL)

{

root->a=newnode(1);

root->a=Insert(root->a,word,meaning);

}

else

{

root->a=Insert(root->a,word,meaning);

}

break;

case 'b':

case 'B':

if(root->b==NULL)

{

root->b=newnode(2);

root->b=Insert(root->b,word,meaning);

}

else

{

root->b=Insert(root->b,word,meaning);

}

break;

case 'c':

case 'C':

if(root->c==NULL)

{

root->c=newnode(3);

root->c=Insert(root->c,word,meaning);

}

else

{

root->c=Insert(root->c,word,meaning);

}

break;

case 'd':

case 'D':

if(root->d==NULL)

{

root->d=newnode(4);

root->d=Insert(root->d,word,meaning);

}

else

{

root->d=Insert(root->d,word,meaning);

}

break;

case 'e':

case 'E':

if(root->e==NULL)

{

root->e=newnode(5);

root->e=Insert(root->e,word,meaning);

}

else

{

root->e=Insert(root->e,word,meaning);

}

break;

case 'f':

case 'F':

if(root->f==NULL)

{

root->f=newnode(6);

root->f=Insert(root->f,word,meaning);

}

else

{

root->f=Insert(root->f,word,meaning);

}

break;

case 'g':

case 'G':

if(root->g==NULL)

{

root->g=newnode(7);

root->g=Insert(root->g,word,meaning);

}

else

{

root->g=Insert(root->g,word,meaning);

}

break;

case 'h':

case 'H':

if(root->h==NULL)

{

root->h=newnode(8);

root->h=Insert(root->h,word,meaning);

}

else

{

root->h=Insert(root->h,word,meaning);

}

break;

case 'i':

case 'I':

if(root->i==NULL)

{

root->i=newnode(9);

root->i=Insert(root->i,word,meaning);

}

else

{

root->i=Insert(root->i,word,meaning);

}

break;

case 'j':

case 'J':

if(root->j==NULL)

{

root->j=newnode(10);

root->j=Insert(root->j,word,meaning);

}

else

{

root->j=Insert(root->j,word,meaning);

}

break;

case 'k':

case 'K':

if(root->k==NULL)

{

root->k=newnode(11);

root->k=Insert(root->k,word,meaning);

}

else

{

root->k=Insert(root->k,word,meaning);

}

break;

case 'l':

case 'L':

if(root->l==NULL)

{

root->l=newnode(12);

root->l=Insert(root->l,word,meaning);

}

else

{

root->l=Insert(root->l,word,meaning);

}

break;

case 'm':

case 'M':

if(root->m==NULL)

{

root->m=newnode(13);

root->m=Insert(root->m,word,meaning);

}

else

{

root->m=Insert(root->m,word,meaning);

}

break;

case 'n':

case 'N':

if(root->n==NULL)

{

root->n=newnode(14);

root->n=Insert(root->n,word,meaning);

}

else

{

root->n=Insert(root->n,word,meaning);

}

break;

case 'o':

case 'O':

if(root->o==NULL)

{

root->o=newnode(15);

root->o=Insert(root->o,word,meaning);

}

else

{

root->o=Insert(root->o,word,meaning);

}

break;

case 'p':

case 'P':

if(root->p==NULL)

{

root->p=newnode(16);

root->p=Insert(root->p,word,meaning);

}

else

{

root->p=Insert(root->p,word,meaning);

}

break;

case 'q':

case 'Q':

if(root->q==NULL)

{

root->q=newnode(16);

root->q=Insert(root->q,word,meaning);

}

else

{

root->q=Insert(root->q,word,meaning);

}

break;

case 'r':

case 'R':

if(root->r==NULL)

{

root->r=newnode(17);

root->r=Insert(root->r,word,meaning);

}

else

{

root->r=Insert(root->r,word,meaning);

}

break;

case 's':

case 'S':

if(root->s==NULL)

{

root->s=newnode(18);

root->s=Insert(root->s,word,meaning);

}

else

{

root->s=Insert(root->s,word,meaning);

}

break;

case 't':

case 'T':

if(root->t==NULL)

{

root->t=newnode(19);

root->t=Insert(root->t,word,meaning);

}

else

{

root->t=Insert(root->t,word,meaning);

}

break;

case 'u':

case 'U':

if(root->u==NULL)

{

root->u=newnode(20);

root->u=Insert(root->u,word,meaning);

}

else

{

root->u=Insert(root->u,word,meaning);

}

break;

case 'v':

case 'V':

if(root->v==NULL)

{

root->v=newnode(21);

root->v=Insert(root->v,word,meaning);

}

else

{

root->v=Insert(root->v,word,meaning);

}

break;

case 'w':

case 'W':

if(root->w==NULL)

{

root->w=newnode(22);

root->w=Insert(root->w,word,meaning);

}

else

root->w=Insert(root->w,word,meaning);

break;

case 'x':

case 'X':

if(root->x==NULL)

{

root->x=newnode(24);

root->x=Insert(root->x,word,meaning);

}

else

{

root->x=Insert(root->x,word,meaning);

}

break;

case 'y':

case 'Y':

if(root->y==NULL)

{

root->y=newnode(25);

root->y=Insert(root->y,word,meaning);

}

else

{

root->y=Insert(root->y,word,meaning);

}

break;

case 'z':

case 'Z':

if(root->z==NULL)

{

root->z=newnode(26);

root->z=Insert(root->z,word,meaning);

}

else

{

root->z=Insert(root->z,word,meaning);

}

break;

default:

return root;

}

}

return root;

}

void display(dictonary \*root)

{

static int t=10,r=3,f=3,d=30;

j=0;

if(root==NULL)

{

return;

}

if(0!=strcmp(root->dat.word,"!@#$"))

{

r+=2;

f+=2;

gotoxy(t,r);

cout<<root->dat.word;

gotoxy(d,f);

cout<<root->dat.meaning;

j++;

}

display(root->a);

display(root->b);

display(root->c);

display(root->d);

display(root->e);

display(root->f);

display(root->g);

display(root->h);

display(root->i);

display(root->j);

display(root->k);

display(root->l);

display(root->m);

display(root->n);

display(root->o);

display(root->p);

display(root->q);

display(root->r);

display(root->s);

display(root->t);

display(root->u);

display(root->v);

display(root->w);

display(root->x);

display(root->y);

display(root->z);

//if(j==0)

//cout<<"Empty Dictonary";

}

void write(dictonary \*root)

{

j=0;

//cout<<"789";

if(root==NULL)

{

return;

}

if(0!=strcmp(root->dat.word,"!@#$"))

{

fprintf(fp,"%s\n%s\n",root->dat.word,root->dat.meaning);

j++;

}

write(root->a);

write(root->b);

write(root->c);

write(root->d);

write(root->e);

write(root->f);

write(root->g);

write(root->h);

write(root->i);

write(root->j);

write(root->k);

write(root->l);

write(root->m);

write(root->n);

write(root->o);

write(root->p);

write(root->q);

write(root->r);

write(root->s);

write(root->t);

write(root->u);

write(root->v);

write(root->w);

write(root->x);

write(root->y);

write(root->z);

//if(j==0)

//cout<<"Empty Dictonary";

}

void search(char name[],dictonary \*root)

{

i++;

if(i==strlen(name))

{

if(root==NULL)

{

cout<<"\n Word Not Found";

return;

}

else if(0!=strcmp(root->dat.word,"!@#$"))

{

cout<<endl<<"\t"<<root->dat.word<<"::";

cout<<root->dat.meaning<<endl;

}

else

{

cout<<"4";

cout<<"\n Word Not Found";

}

}

else if(root==NULL)

{

cout<<"\n Word Not Found";

return;

}

else

{

switch(name[i])

{

case 'a':

case 'A':

search(name,root->a);

break;

case 'b':

case 'B':

search(name,root->b);

break;

case 'c':

case 'C':

search(name,root->c);

break;

case 'd':

case 'D':

search(name,root->d);

break;

case 'e':

case 'E':

search(name,root->e);

break;

case 'f':

case 'F':

search(name,root->f);

break;

case 'g':

case 'G':

search(name,root->g);

break;

case 'h':

case 'H':

search(name,root->h);

break;

case 'i':

case 'I':

search(name,root->i);

break;

case 'j':

case 'J':

search(name,root->j);

break;

case 'k':

case 'K':

search(name,root->k);

break;

case 'l':

case 'L':

search(name,root->l);

break;

case 'm':

case 'M':

search(name,root->m);

break;

case 'n':

case 'N':

search(name,root->n);

break;

case 'o':

case 'O':

search(name,root->o);

break;

case 'p':

case 'P':

search(name,root->p);

break;

case 'q':

case 'Q':

search(name,root->q);

break;

case 'r':

case 'R':

search(name,root->r);

break;

case 's':

case 'S':

search(name,root->s);

break;

case 't':

case 'T':

search(name,root->t);

break;

case 'u':

case 'U':

search(name,root->u);

break;

case 'v':

case 'V':

search(name,root->v);

break;

case 'w':

case 'W':

search(name,root->w);

break;

case 'x':

case 'X':

search(name,root->x);

break;

case 'y':

case 'Y':

search(name,root->y);

break;

case 'z':

case 'Z':

search(name,root->z);

break;

default:

return;

}

}

}

void delty(char name[],dictonary \*root)

{

char z;

i++;

if(i==strlen(name))

{

if(root==NULL)

{

cout<<"\n Word Not Found";

return;

}

else if(0!=strcmp(root->dat.word,"!@#$"))

{

cout<<endl<<"\t"<<root->dat.word<<"::";

cout<<root->dat.meaning<<endl;

cout<<"\n\tAre You sure You want to delete\n";

cin>>z;

if(z=='y')

{

strcpy(root->dat.word,"!@#$");

}

}

else

{

cout<<"\n Word Not Found";

}

}

else if(root==NULL)

{

cout<<"\n Word Not Found";

return;

}

else

{

switch(name[i])

{

case 'a':

case 'A':

delty(name,root->a);

break;

case 'b':

case 'B':

delty(name,root->b);

break;

case 'c':

case 'C':

delty(name,root->c);

break;

case 'd':

case 'D':

delty(name,root->d);

break;

case 'e':

case 'E':

delty(name,root->e);

break;

case 'f':

case 'F':

delty(name,root->f);

break;

case 'g':

case 'G':

delty(name,root->g);

break;

case 'h':

case 'H':

delty(name,root->h);

break;

case 'i':

case 'I':

delty(name,root->i);

break;

case 'j':

case 'J':

delty(name,root->j);

break;

case 'k':

case 'K':

delty(name,root->k);

break;

case 'l':

case 'L':

delty(name,root->l);

break;

case 'm':

case 'M':

delty(name,root->m);

break;

case 'n':

case 'N':

delty(name,root->n);

break;

case 'o':

case 'O':

delty(name,root->o);

break;

case 'p':

case 'P':

delty(name,root->p);

break;

case 'q':

case 'Q':

delty(name,root->q);

break;

case 'r':

case 'R':

delty(name,root->r);

break;

case 's':

case 'S':

delty(name,root->s);

break;

case 't':

case 'T':

delty(name,root->t);

break;

case 'u':

case 'U':

delty(name,root->u);

break;

case 'v':

case 'V':

delty(name,root->v);

break;

case 'w':

case 'W':

delty(name,root->w);

break;

case 'x':

case 'X':

delty(name,root->x);

break;

case 'y':

case 'Y':

delty(name,root->y);

break;

case 'z':

case 'Z':

delty(name,root->z);

break;

}

}

}

void menu(dictonary \*head)

{

system("cls");

int m;

char word[20];

char meaning[100];

char c='y';

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Welcome to Dictonary\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

cout<<" Choose Your Option\n\n 1.Add Words To Dictonary\n 2.See All Words\n 3.Search a Word\n 4.Delete a word\n 5.Exit\n";

cin>>m;

system("cls");

switch(m)

{

case 1:

c='y';

while(c=='y')

{

i=-1;

cout<<"\n Enter Word::";

cin>>word;

fflush(stdin);

cout<<"\n Meaning::";

gets(meaning);

head=Insert(head,word,meaning);

cout<<"\n\n Want To Continue y/n ";

cin>>c;

}

menu(head);

break;

case 2:

cout<<" Type 1 to see all words\n\n Type any letter to see the word starting with that letter\n";

cin>>c;

system("cls");

gotoxy(10,2);

cout<<"Word";

gotoxy(30,2);

cout<<"Meaning\n\n";

switch(c)

{

case '1':

display(head);

break;

case 'a':

case 'A':

display(head->a);

break;

case 'b':

case 'B':

display(head->b);

break;

case 'c':

case 'C':

display(head->c);

break;

case 'd':

case 'D':

display(head->d);

break;

case 'e':

case 'E':

display(head->e);

break;

case 'f':

case 'F':

display(head->f);

break;

case 'g':

case 'G':

display(head->g);

break;

case 'h':

case 'H':

display(head->h);

break;

case 'i':

case 'I':

display(head->i);

break;

case 'j':

case 'J':

display(head->j);

break;

case 'k':

case 'K':

display(head->k);

break;

case 'l':

case 'L':

display(head->l);

break;

case 'm':

case 'M':

display(head->m);

break;

case 'n':

case 'N':

display(head->n);

break;

case 'o':

case 'O':

display(head->o);

break;

case 'p':

case 'P':

display(head->p);

break;

case 'q':

case 'Q':

display(head->q);

break;

case 'r':

case 'R':

display(head->r);

break;

case 's':

case 'S':

display(head->s);

break;

case 't':

case 'T':

display(head->t);

break;

case 'u':

case 'U':

display(head->u);

break;

case 'v':

case 'V':

display(head->v);

break;

case 'w':

case 'W':

display(head->w);

break;

case 'x':

case 'X':

display(head->x);

break;

case 'y':

case 'Y':

display(head->y);

break;

case 'z':

case 'Z':

display(head->z);

break;

default:

return;

}

cin>>c;

menu(head);

break;

case 3:

c='y';

while(c=='y'||c=='Y')

{

cout<<"\n Type a name to search::";

cin>>word;

i=-1;

search(word,head);

cout<<"\n\tWant To Continue y/n ";

cin>>c;

}

menu(head);

break;

case 4:

c='y';

while(c=='y'||c=='Y')

{

cout<<" Type a name to delete::";

cin>>word;

i=-1;

delty(word,head);

cout<<"\n\tWant To Continue y/n ";

cin>>c;

}

menu(head);

break;

case 5:

system("cls");

cout<<"\n\n\n\n\n\n\n\n\n\n Thank You Have a Nice Day";

fp=fopen("G:\\roshan.txt","w+");

write(head);

fclose(fp);

exit(0);

default:

cout<<" \*\*\*\*\*\*\*\*\*\*\*Incorrect Selection\*\*\*\*\*\*\*\*\*\*\n";

}

}

int main()

{

char word[20];

char meaning[100];

dictonary \*head=newnode(100);

int f;

char ch;

fp=fopen("G:\\roshan.txt","a+");

if(fp==NULL)

{

cout<<"File Founding Error";

}

else

{

while(1)

{

ch=fgetc(fp);

//cout<<ch;

if(ch==EOF)

break;

f=0;

while(ch!='\n')

{

word[f]=ch;

f++;

ch=fgetc(fp);

}

word[f]='\0';

f=0;

ch=fgetc(fp);

while(ch!='\n')

{

meaning[f]=ch;

f++;

ch=fgetc(fp);

}

meaning[f]='\0';

i=-1;

//cout<<word<<"::"<<meaning<<endl;

head=Insert(head,word,meaning);

}

}

//cin>>f;

fclose(fp);

menu(head);

return 0;

}

**OUTPUT**

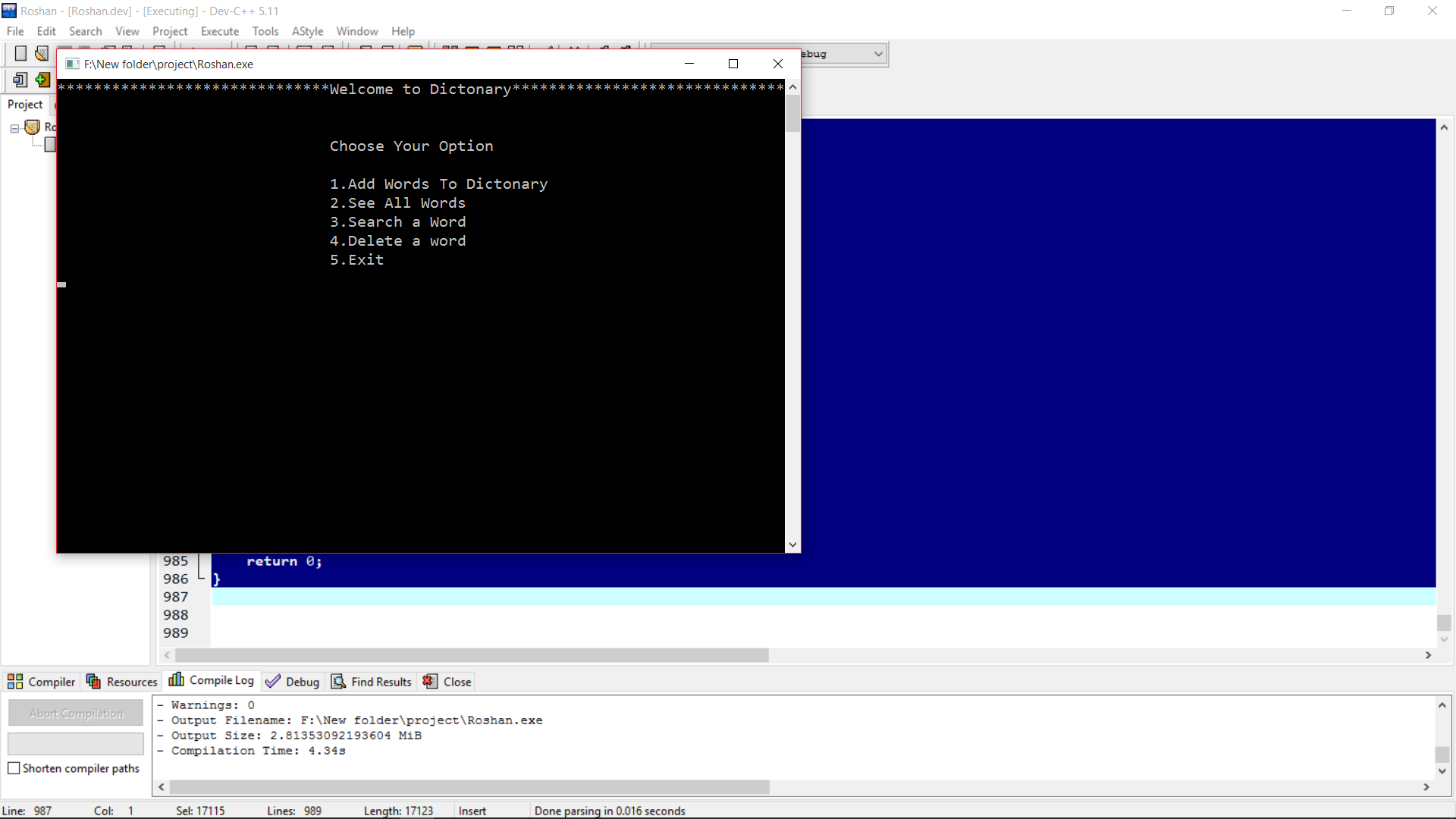


Fig:1.1 Main Menu

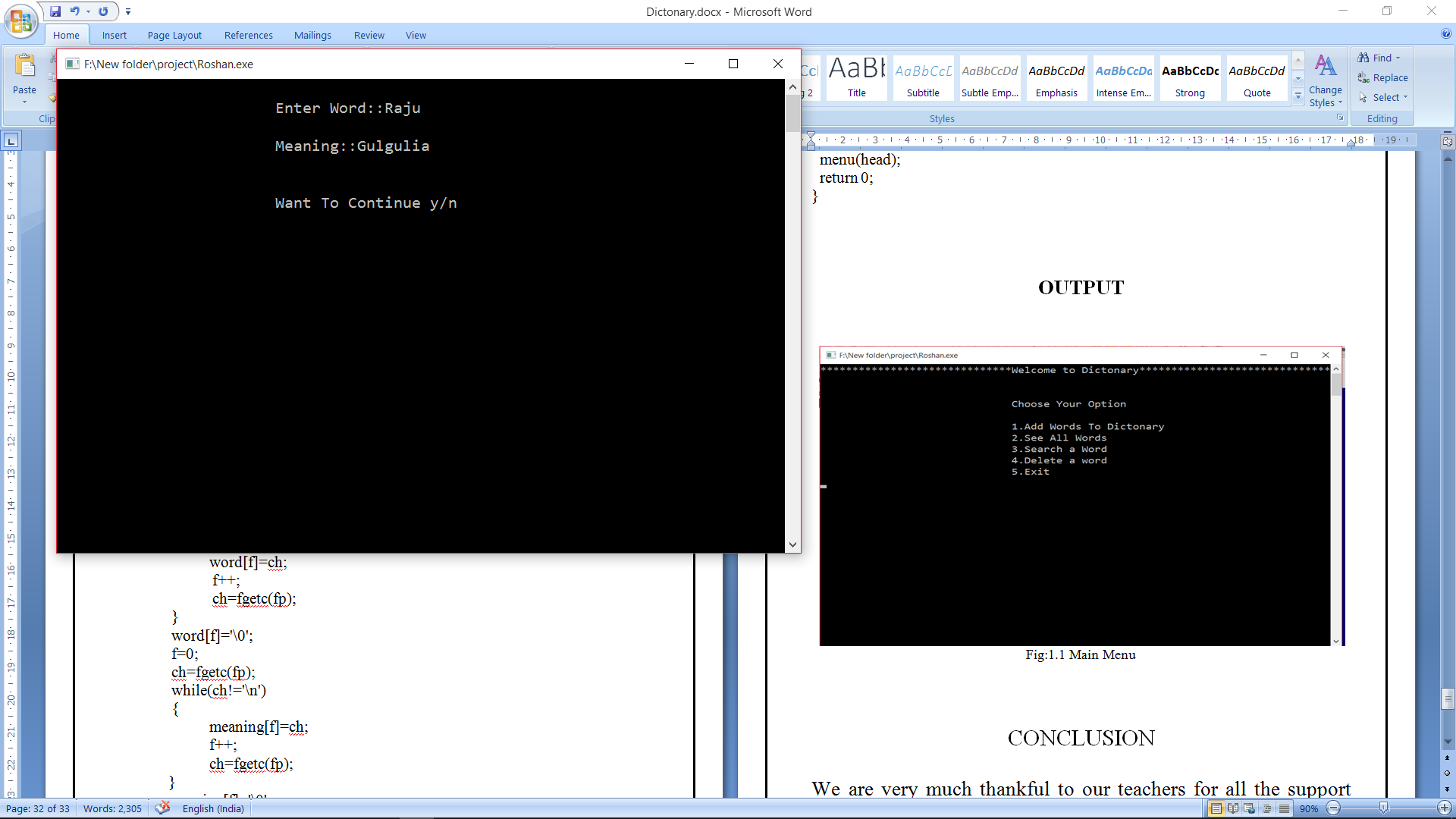


Fig:1.2 Adding Words

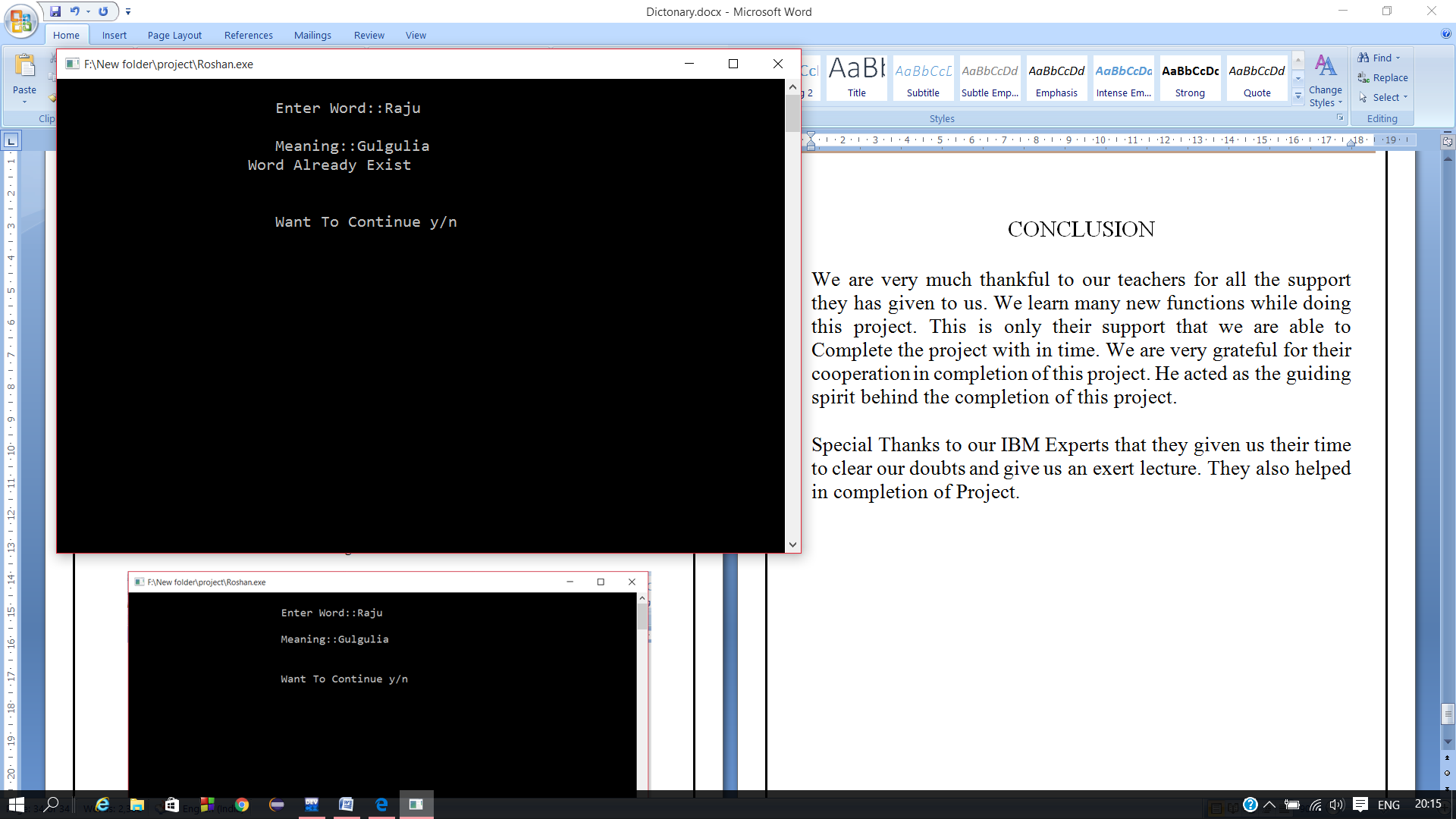


Fig:1.3 Duplicate Word is not possible

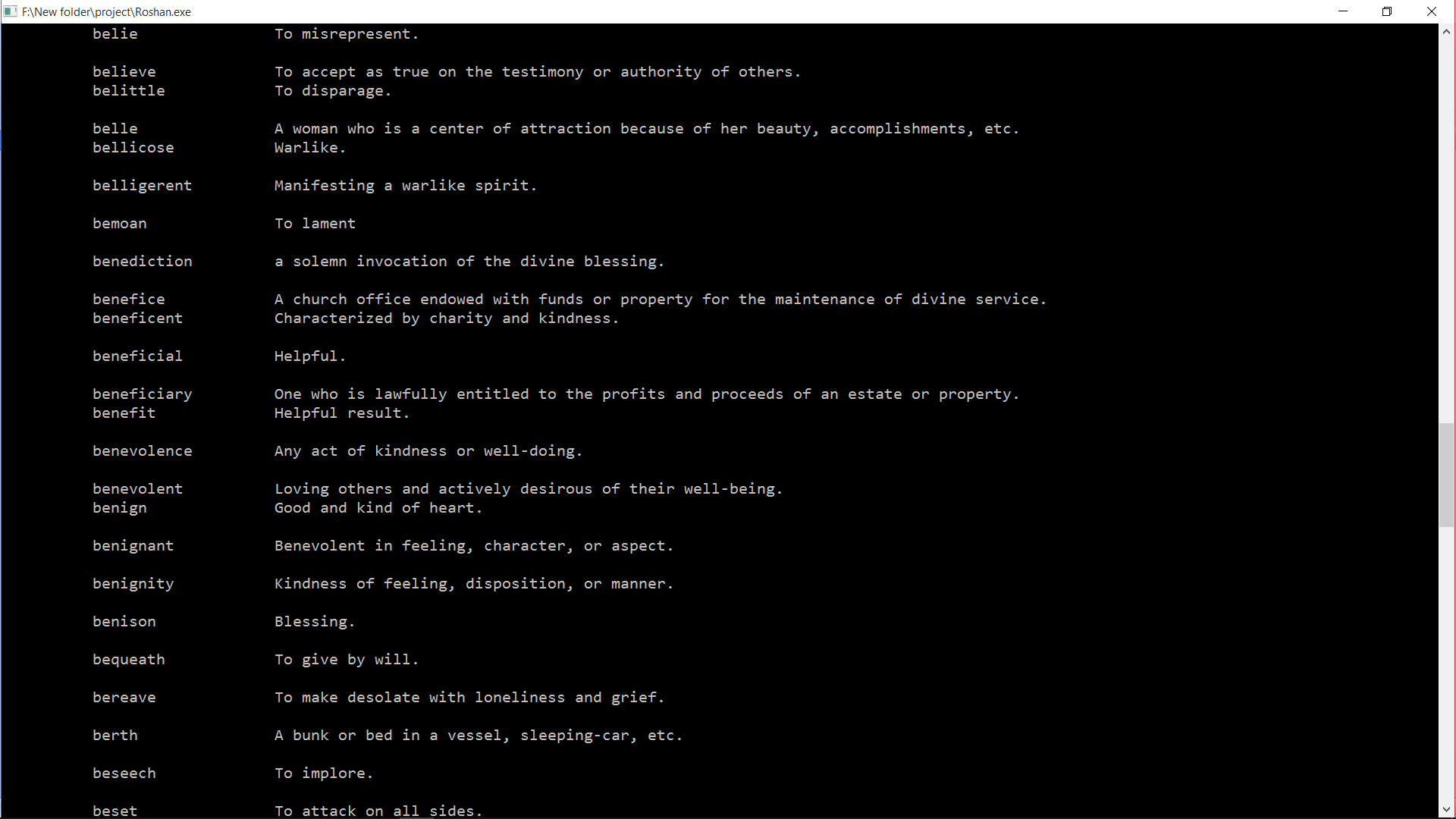


Fig: See All Words

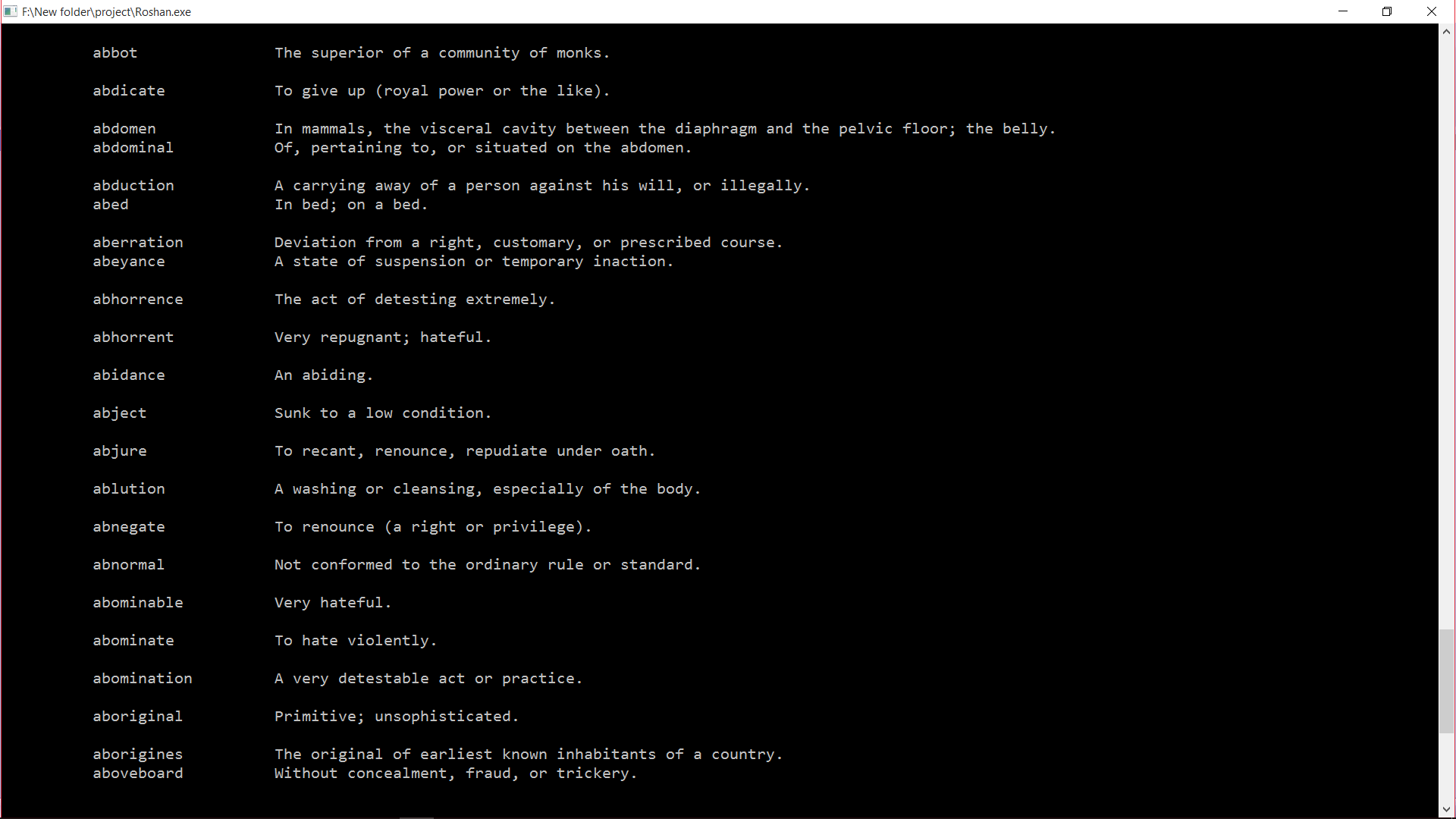


Fig:1.5 Words Starting with A

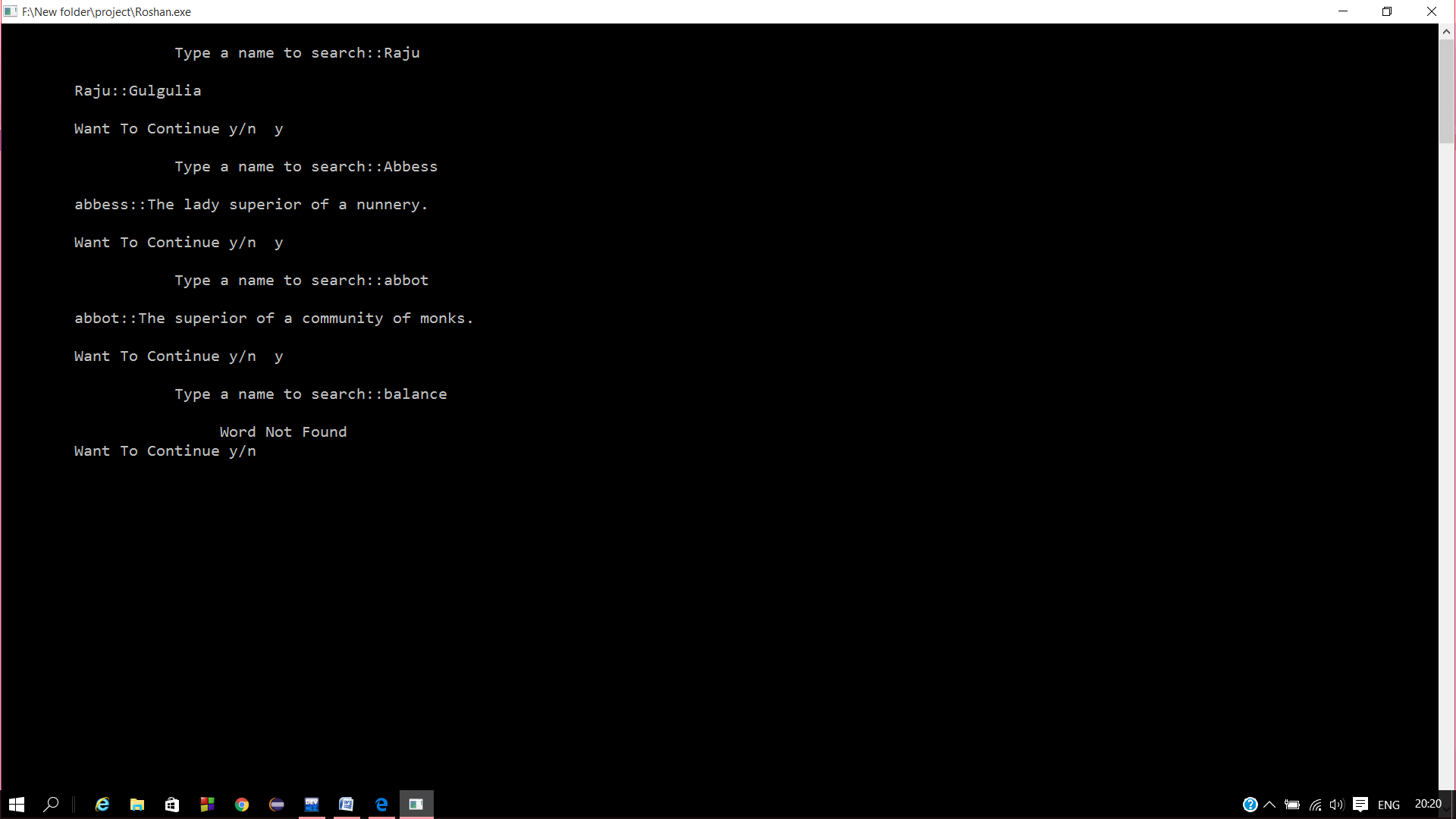
.

Fig:1.6 searching Words

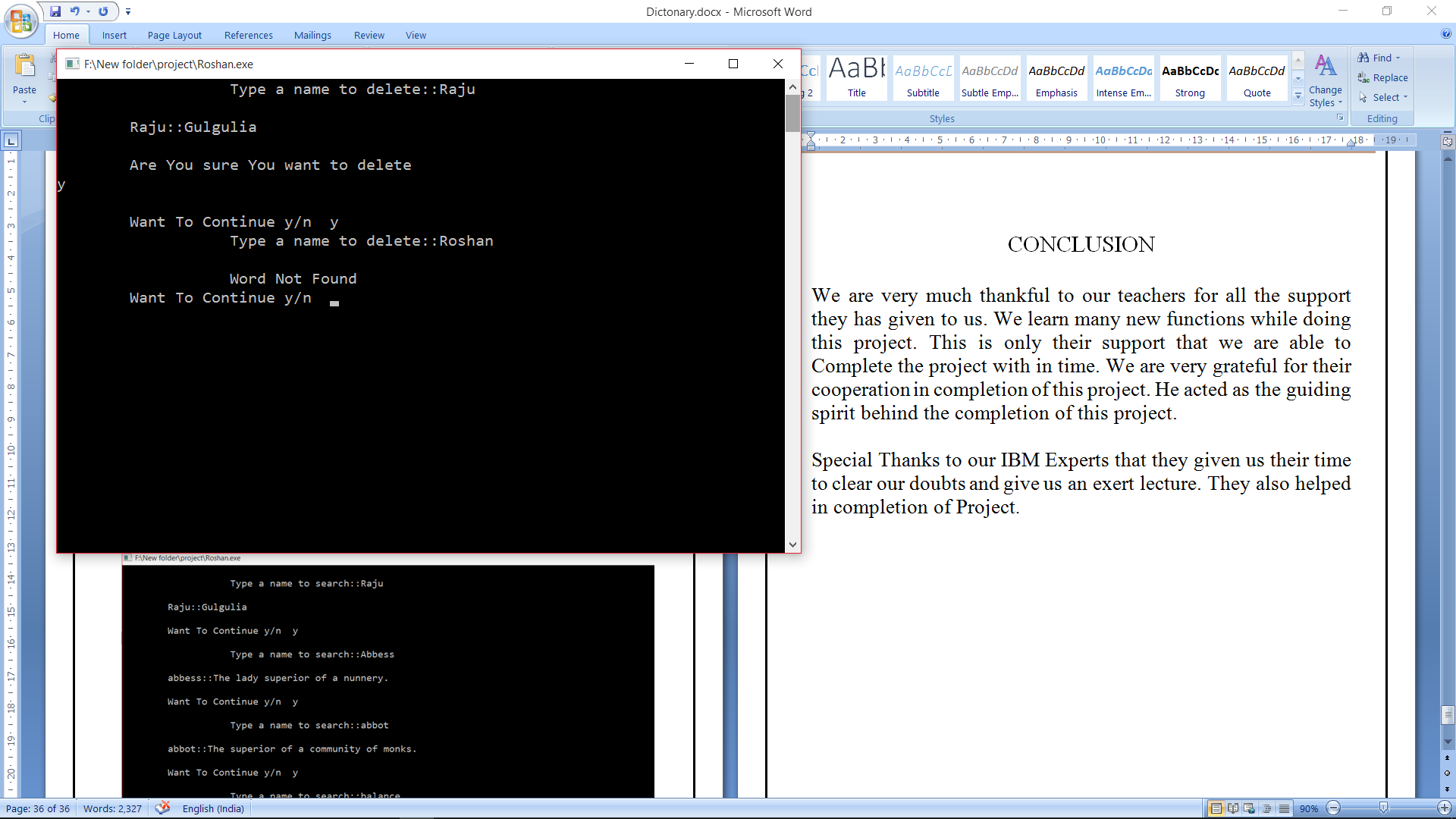


Fig:1.7 Delete a word

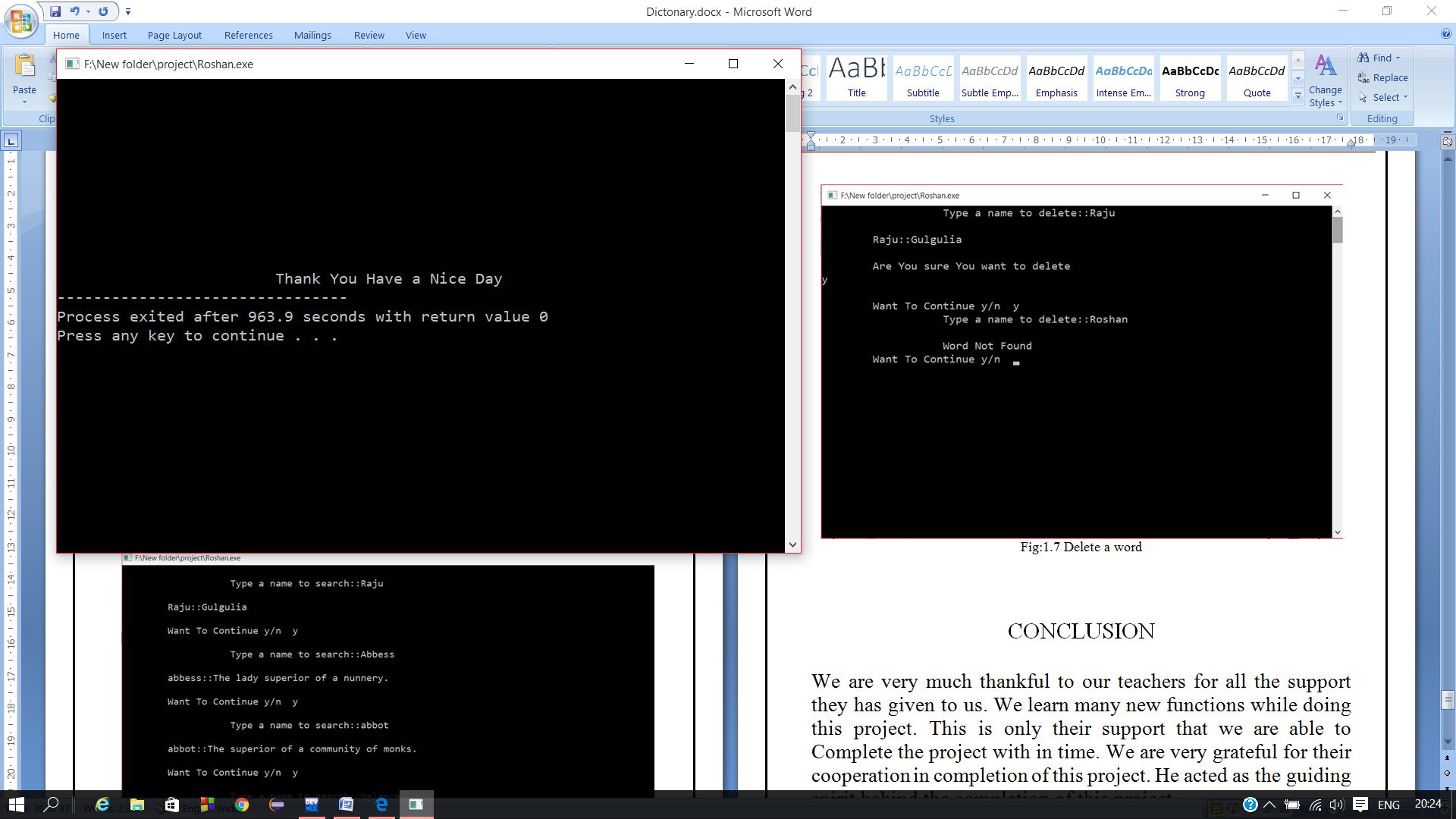


Fig:1.8 Exit

CONCLUSION

We are very much thankful to our teachers for all the support they has given to us. We learn many new functions while doing this project. This is only their support that we are able to Complete the project with in time. We are very grateful for their cooperation in completion of this project. He acted as the guiding spirit behind the completion of this project.

Special Thanks to our IBM Experts that they given us their time to clear our doubts and give us an exert lecture. They also helped in completion of Project.