CSE 207: Data Structures
Section: 2, Fall-2019

A Project Report
on
The Word Dictionary

Submitted by:
Group Members:

Sk Mohammad Asem	2017-3-60-068
Sirajum Maria Muna	2017-3-60-020

Submitted to:
Dr. Maheen Islam
Assistant Professor, Dept. of Computer Science and Engineering
East West University

Date of Submission:12-12-2019

1. Problem Statement

In this project we have to build a word dictionary. In this dictionary words are arranged in alphabetical order along with its meaning. The dictionary has some applications such as, add new words with its meaning, search any word(if not found then show similar word), delete any word.

2. System Requirements

The system on which the project is implemented has the following properties:

Processor: Intel Core i3,1.90 GHz
RAM: 8.00 GB
Operating System: Windows 8.1 , 64-bit
IDE (Integrated Development Environment): Code-Blocks

3. System Design

In this section, the algorithm of the project is described as below:

1. Play an infinity loop.
2. Scan the option from the keyboard.
3. If option is equals to 1 then enter insert function. This function inserts word with its meaning.
4 . If option is equals to 2 then enter search option. If it is fond then prints “found” and if it is not then show the similar word ..
5. If option is equals to 3 then it delete any inserted word.
6.Every option has an condition to go back to the main menu.

4. Implementation

In this section, important parts of the source code are explained.

Insert any word:

The function is for inserting any word. At first it was empty. So when root is being send it checks that the root is empty to it inserts word in the place of root. And copy in another tree. For second insert it inserts data in the end by filling up the left and right side of the tree.

```
struct node* insert(struct node* root, char data[], char mean[])

{

   if(root==NULL)

   {

      n++;

      struct node *cur;
```

```
        cur=(struct node*)malloc(sizeof(struct node));
        strcpy(cur->word,data);
        strcpy(cur->wordM,mean);
//        cur->word=data;
        cur->right=NULL;
        cur->left=NULL;
        root=cur;
        return root;
    }
    else if(strcmp(root->word,data)>0)
    {
        root->left=insert(root->left,data,mean);
    }
    else if(strcmp(root->word,data)<0)
    {
        root->right=insert(root->right,data,mean);
    }

    return root;
}
```

Search word: It takes input from the main function. It matches the search word in the inserted words and then print its query.

```
struct node* search(struct node * root,char data[])
{
```

```
    if(root!=NULL)
    {
        if(strcmp(root->word,data)==0)
            return root;
        else if(strcmp(root->word,data)>0)
            return search(root->left,data);
        else if(strcmp(root->word,data)<0)
            return search(root->right,data);
    }
    else
    {
        return root;
    }
}
```

Find similar word: If the searched word is not found then it prints its similar word.

```
void searchSimilar(struct node* root,char st[],int prntf)
{
    if(root!=NULL)
    {
        strcpy(dic,root->word);
        int i,sum=0,len;
        len=strlen(st);
        len=len/2;
      //  printf("len %d %s\n",len,dic);
```

```
        for(i=0; i<len; i++)
        {
           if(dic[i]==st[i])
              sum+=1;
          // printf("%d %d\n",i,sum);
        }
        if(sum==len)
        {
           if(prntf==0)
           {
              printf("Did you mean ");
              prntf=1;
              printf(" %c%s%c",ch,dic,ch);
           }
           else
              printf(", %c%s%c",ch,dic,ch);
        }
        searchSimilar(root->left,st,prntf);
        searchSimilar(root->right,st,prntf);
     }
}
```

Synonym: If the searched word is found then it prints found command along with its synonym.

```
void synonym(struct node* root,struct node* sRoot)
{
```

```
  if(root!=NULL )
  {
    if(strcmp(root->wordM,sRoot->wordM)==0 && root!=sRoot)
    {
      if(y==0)
      {
        printf("The given words have similiar meaning of our searched word: ");
        printf(" %c%s%c",ch,root->word,ch);
        y=1;
      }
      else
        printf(", %c%s%c",ch,root->word,ch);
    }
    synonym(root->left,sRoot);
    synonym(root->right,sRoot);
  }
}
```

Delete word: We can enter the word which we want to delete and it matches if the word is found in the word list(means tree). If it is found then it deletes the word. And prints command.

```
struct node* deleteNode(struct node *root,char data[])
{

  if(root == NULL)
    return  root;
```

```
else if (strcmp(root->word,data) > 0 )
{
   root->left =  deleteNode(root->left,data);
   return root;
}
else if (strcmp(root->word,data)<0)
{
   root->right= deleteNode(root->right,data);
   return root;
}
else
{
   if(root->right==NULL && root->left==NULL)
   {
      temp=root;
      root=NULL;
      free(temp);
      n--;
      return root;
   }
   else if (root->left == NULL )
   {
      temp = root;
      root = root->right;
```

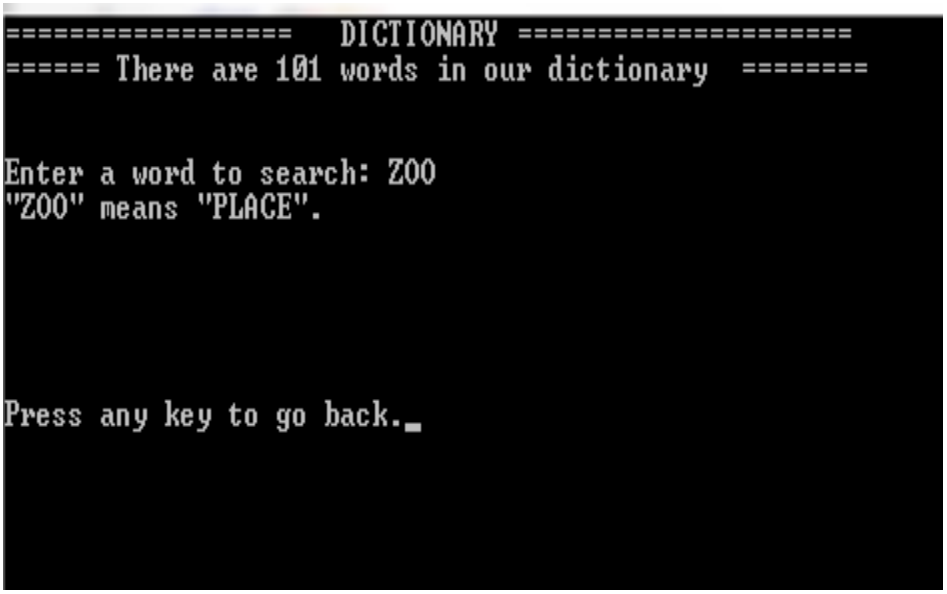
```
            free(temp);
            n--;
            return root;
        }
        else if (root->right == NULL )
        {
            temp = root;
            root = root->left;
            free(temp);
            n--;
            return root;
        }
        else
        {
            temp2=findMinChar(root->right);
            strcpy(root->word,temp2->word);
            strcpy(root->wordM,temp2->wordM);
            root->right = deleteNode(root->right,temp2->word);
            return root;
        }
    }
}
```

5. Testing Results

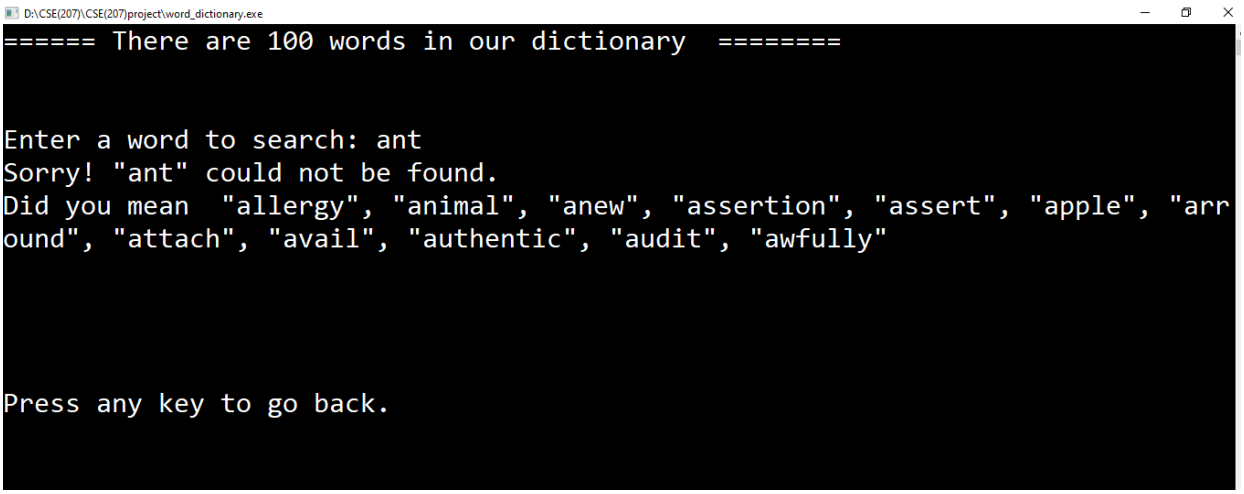
In this section, some sample input-outputs are explained with appropriate screenshot.

Test case 1:

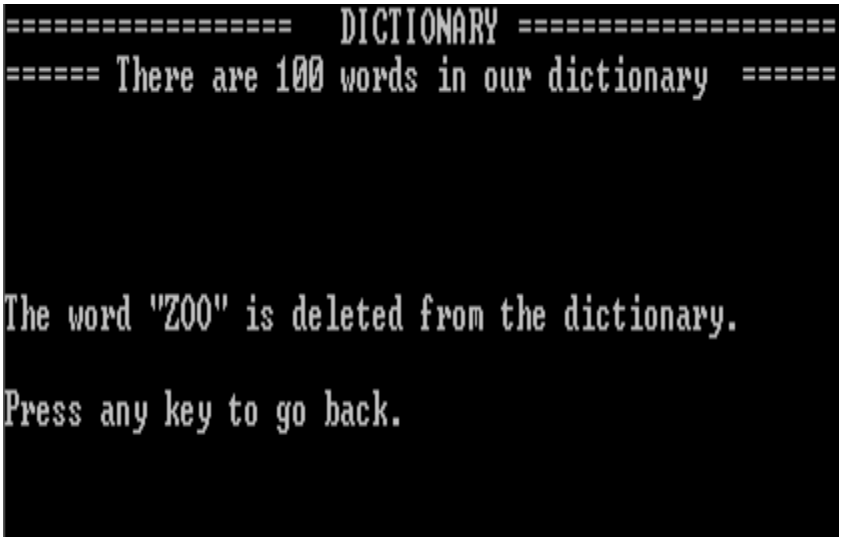
Test case 2:



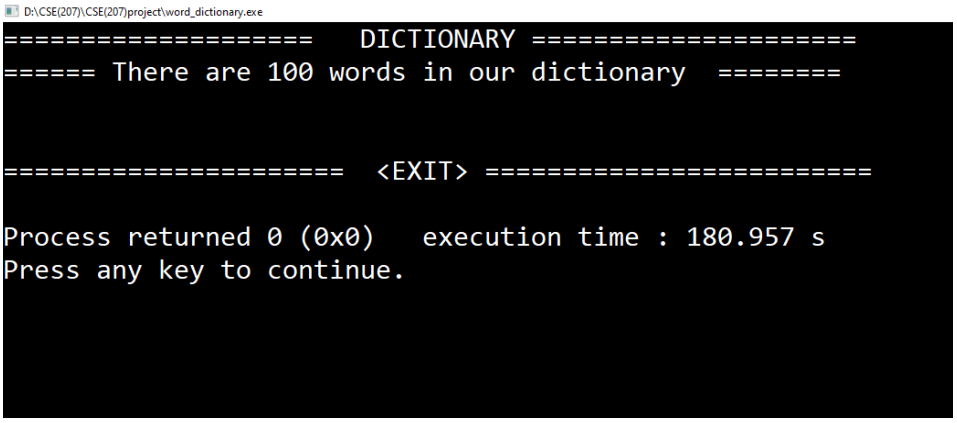
Test case 3:



Test case 4:



Test Case 5:



6. Future Scope

There are some limitations of our project. The first limitation it can't match lower case and upper case letter. We can't control file at run the program. Another limitation is we cannot store more than 227 words in our file ase file has some word limitations. Then we need to add multiple files in our project.