UNIVERSITI SAINS MALAYSIA

SCHOOL OF COMPUTER SCIENCE

CPT113-PROGRAMMING METHODOLOGY & DATA STRUCTURES

ASSIGNMENT 1

Anagram Generator

NAME : HASHIM BIN ROSLI

IC NUMBER : 001008-07-0447

MATRIC NUMBER : 146869

GROUP : B

LECTURER NAME : ENCIK MUHD AZAM BIN OSMAN

CONTENT :

INTRODUCTION :

Anagram is rearrange letters of a word and form other words with same letters but different position of original word. In this program user can generate anagram for any word either from text file or key-in by keyboard. The anagrams that generate by this program only can generate word with different permutation of letter’s position without any condition.

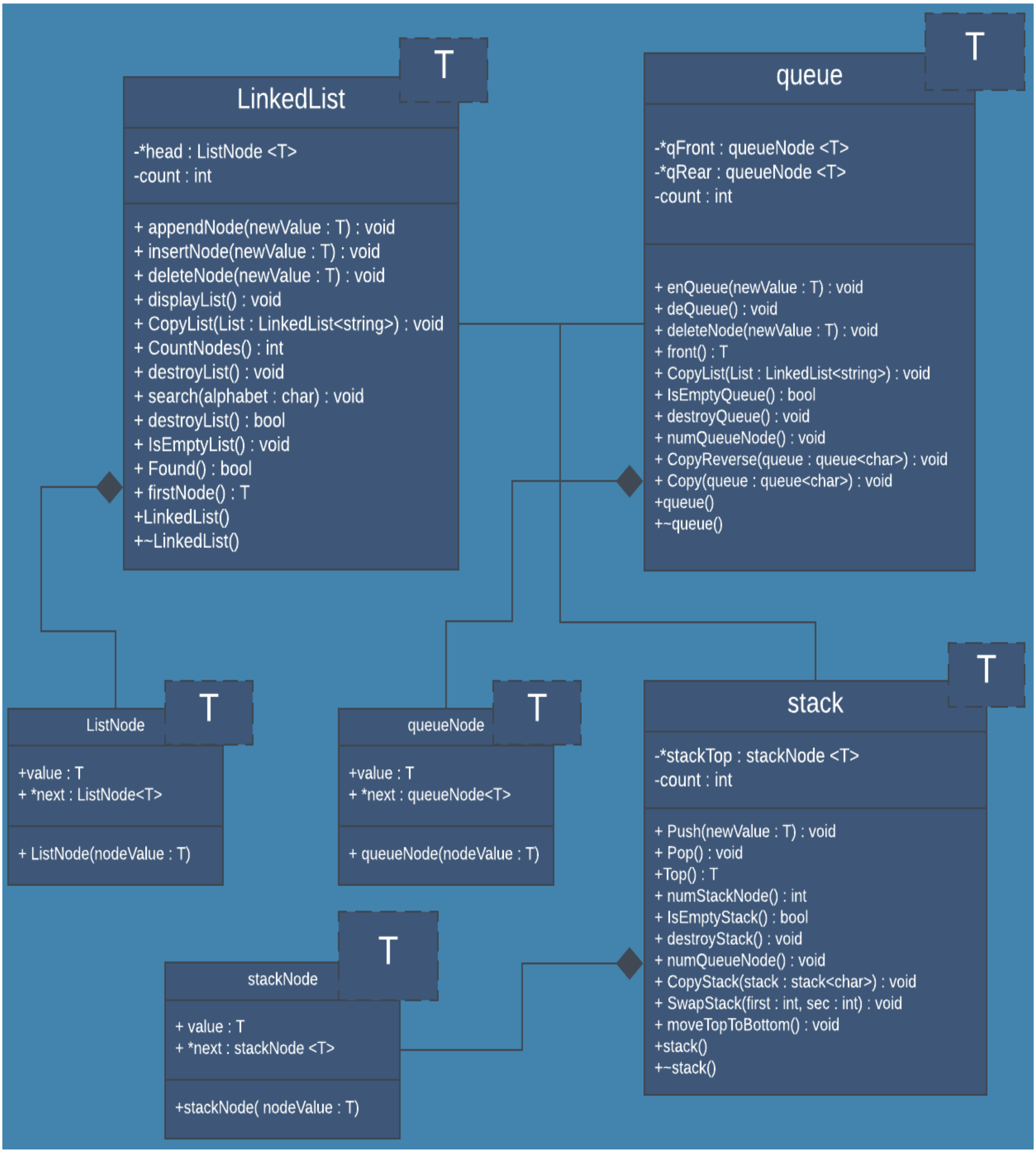
Other than generating anagrams, user can manage the word with its anagram such as delete, update or add new word and generating new anagram. After this program end, all word will be saved inside text file.

This program also implementing linked list, stack and queue data structure for user to manage the anagrams. Linked list is functioning to store the original word and its anagrams while stack and queue is used to generate anagram from the original word.

PSEUDOCODE :

```
1. Reading text file name.txt and store it into object linked list.
2. Display all word inside the text file.
3. Storing all first node of Linked List objects in to Stack and Queue.
4. Display menu.
5. User have given to choose from menu :
    1) If user choose to display all word, the program displaying all first node of Link List objects.
    2) If user choosing to Display all anagram, the program will generate for all anagram for all first node of link list objects. The anagrams will be save into object of its original word.
    3) If user choosing to search all anagram with specific starting letter, the program will search anagram inside link list with same starting letter.
    4) If user choosing to add new word, the program add new node into link list and generating anagram of its word. The anagram also saved into the link list of same original word.
    5) If user choosing to update the program find if the word inserted by user same with first node of link list. If found it, the program will delete all node of the link list and insert the new word into same link list and generate anagram of the word. The anagram of new word will be save into same link list.
    6) If user choosing to delete word and its anagram, the program will destroy the link list that have the word that the user want to destroy.
    7) If user want to end program, the program will shut down, otherwise user can choose again in the menu.
6. Program end.
```

UML DIAGRAM :


LinkedList
T
-*head : ListNode <T>
-count : int
+ appendNode(newValue : T) : void
+ insertNode(newValue : T) : void
+ deleteNode(newValue : T) : void
+ displayList() : void
+ CopyList(List : LinkedList<string>) : void
+ CountNodes() : int
+ destroyList() : void
+ search(alphabet : char) : void
+ destroyList() : bool
+ IsEmptyList() : void
+ Found() : bool
+ firstNode() : T
+LinkedList()
+~LinkedList()
queue
T
-*qFront : queueNode <T>
-*qRear : queueNode <T>
-count : int
+ enQueue(newValue : T) : void
+ deQueue() : void
+ deleteNode(newValue : T) : void
+ front() : T
+ CopyList(List : LinkedList<string>) : void
+ IsEmptyQueue() : bool
+ destroyQueue() : void
+ numQueueNode() : void
+ CopyReverse(queue : queue<char>) : void
+ Copy(queue : queue<char>) : void
+queue()
+~queue()
ListNode
T
+value : T
+ *next : ListNode<T>
+ ListNode(nodeValue : T)
queueNode
T
+value : T
+ *next : queueNode<T>
+ queueNode(nodeValue : T)
stack
T
-*stackTop : stackNode <T>
-count : int
+ Push(newValue : T) : void
+ Pop() : void
+Top() : T
+ numStackNode() : int
+ IsEmptyStack() : bool
+ destroyStack() : void
+ numQueueNode() : void
+ CopyStack(stack : stack<char>) : void
+ SwapStack(first : int, sec : int) : void
+ moveTopToBottom() : void
+stack()
+~stack()
stackNode
T
+ value : T
+ *next : stackNode <T>
+stackNode(nodeValue : T)

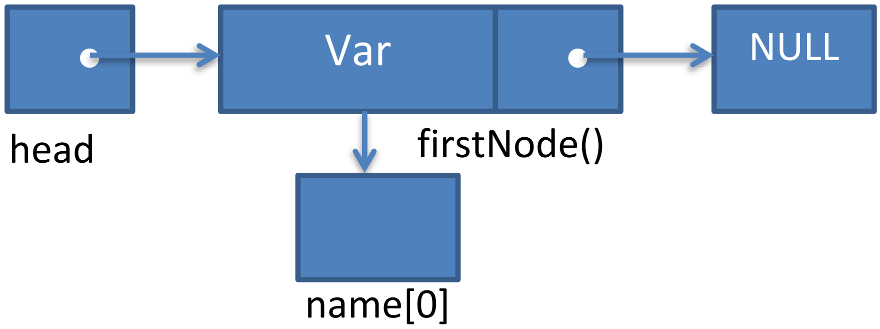
LINKED LIST,STACK AND QUEUE DIAGRAM:

FUNCTION -

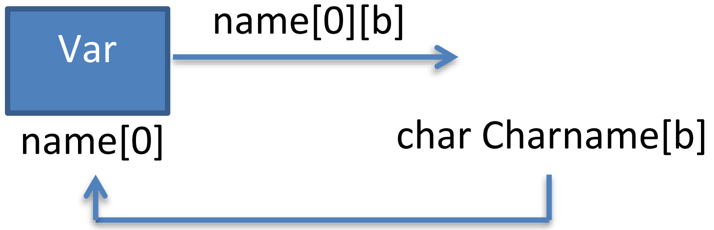
StoreintoChar(LinkedList<string> List[],stack<char> Stack[],queue<char> Queue[],int size)

List[0] :

1.

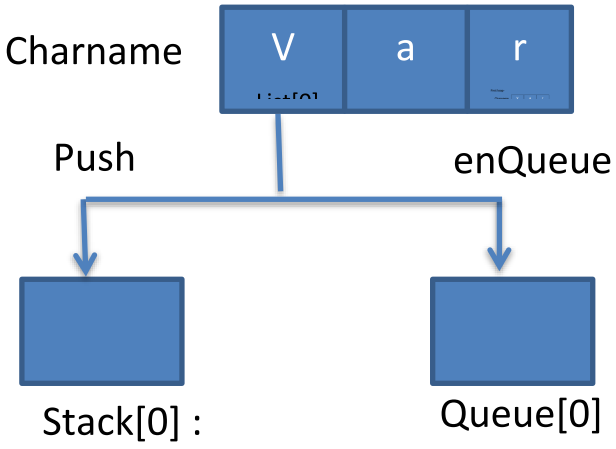


2. Loop for b=name[0].length() and b keep add 1.

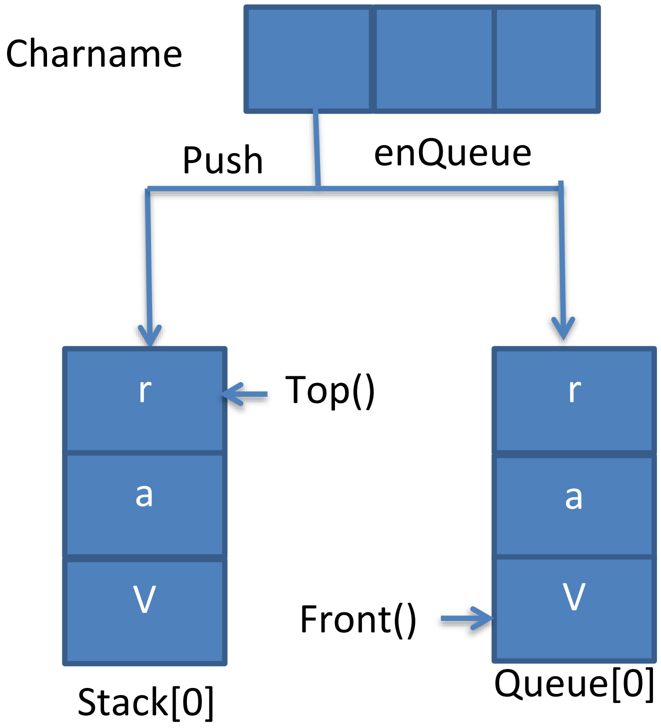


3. Loop for Charname.length() keep add 1.

First loop-

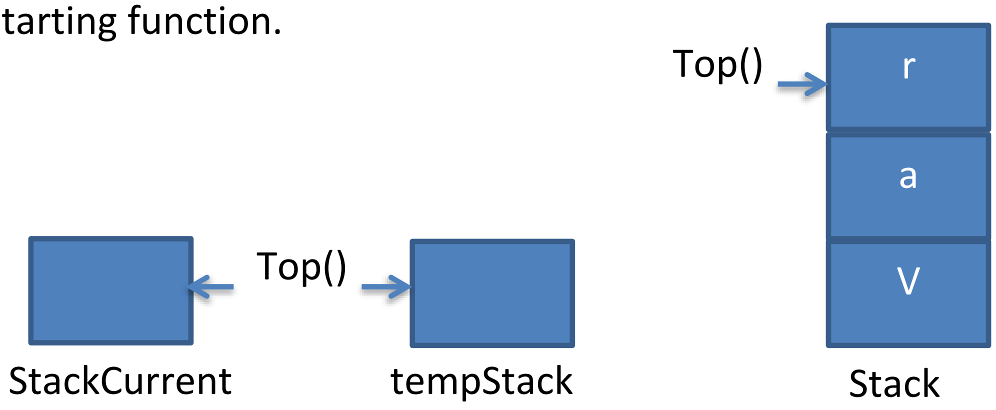


End loop-

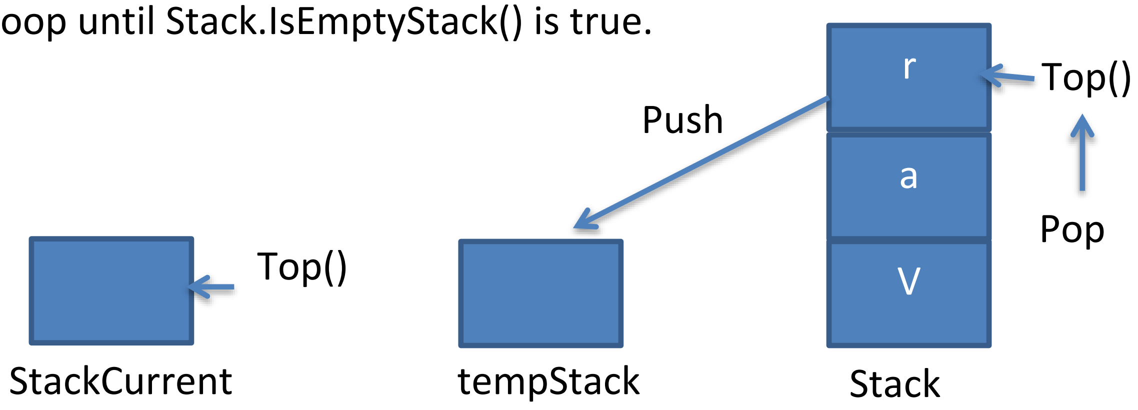


CopyStack()

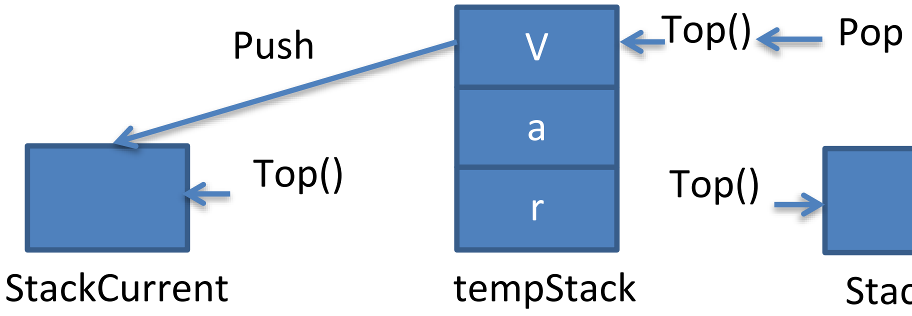
1. Starting function.



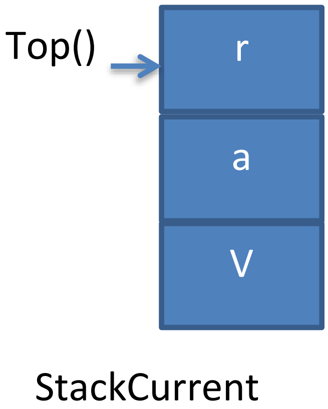
2. Loop until Stack.IsEmptyStack() is true.



3. Loop until tempStack.IsEmptyStack is true.

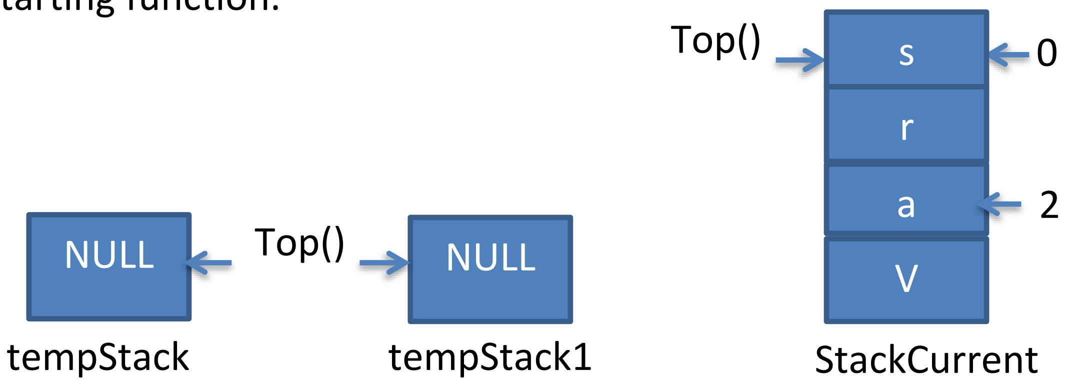


4. After end loop.

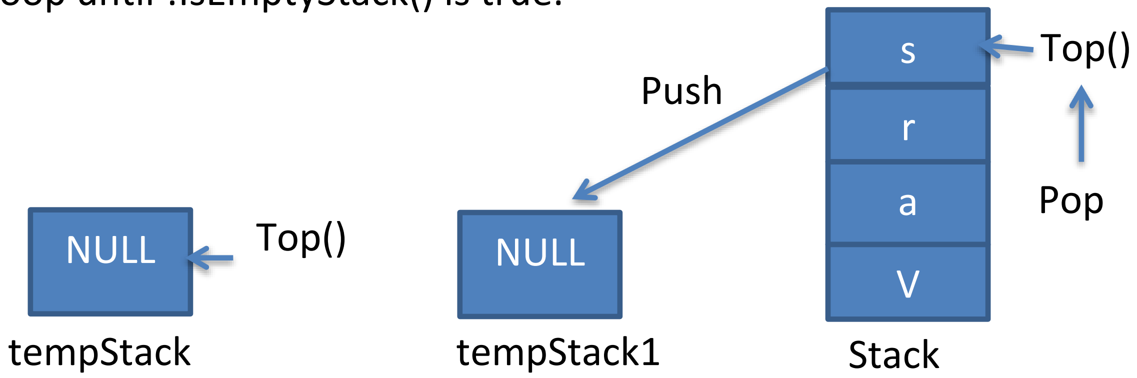


SwapStack(0,2)

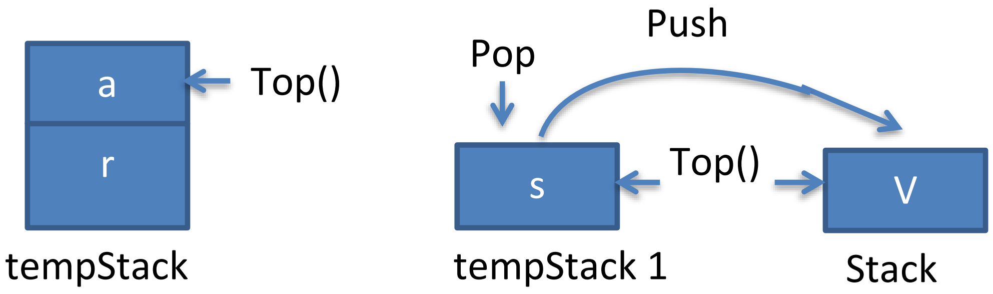
1. Starting function.



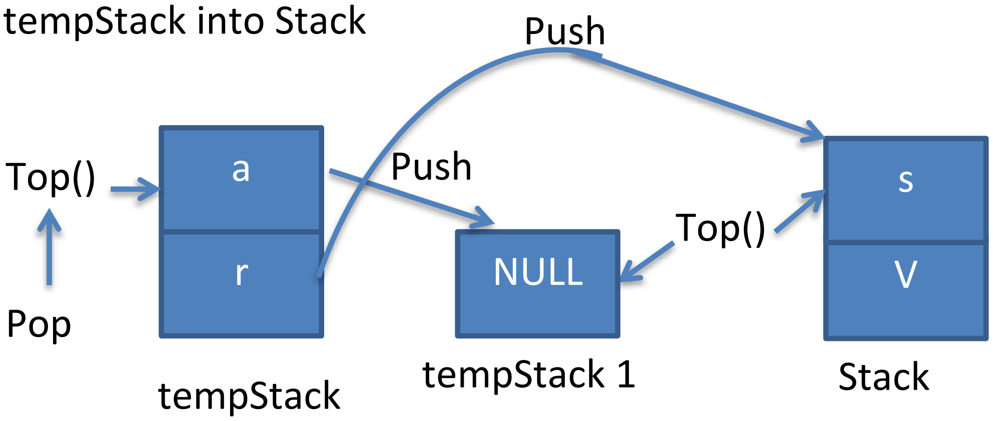
2. Loop until !IsEmptyStack() is true.



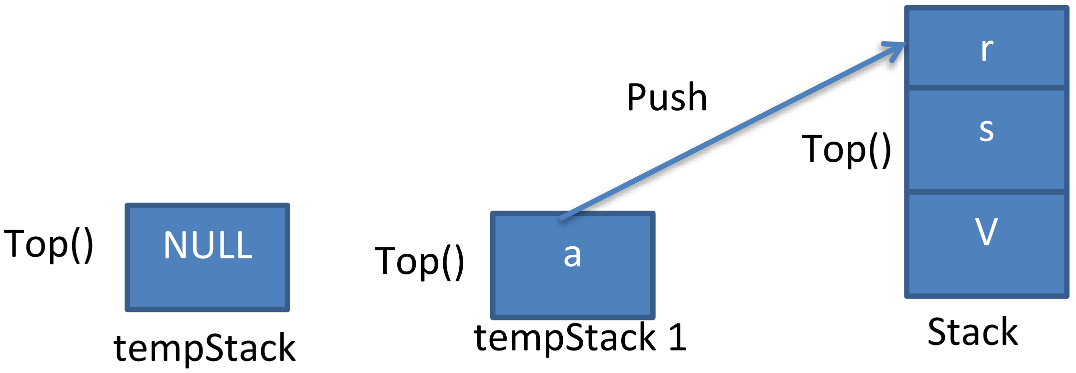
3. End Loop and Push tempStack1 into Stack and pop tempStack1.



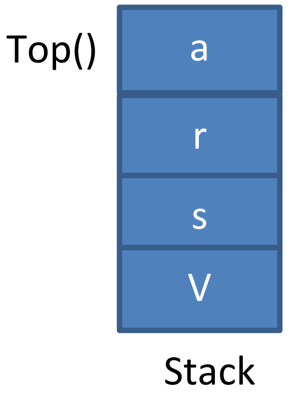
4. Push tempStack into tempStack1 and pop tempStack1. Push all tempStack into Stack



5. Push tempStack1 into Stack and pop tempStack1.

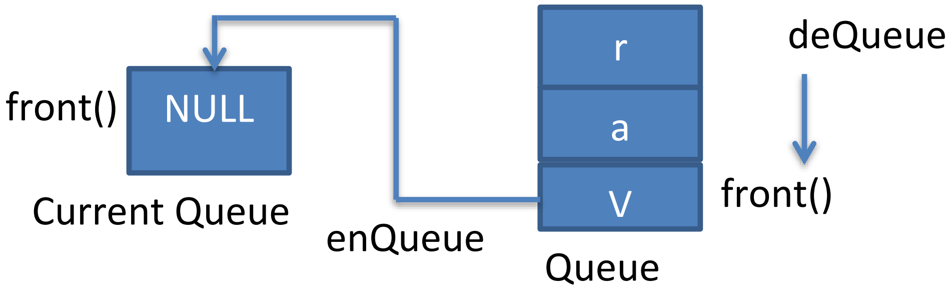


6. After end Function.

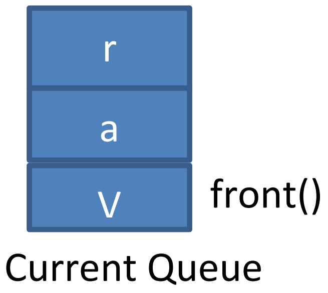


Copy(queue<char> Queue)

1. Beginning of function and loop until Queue is Empty.

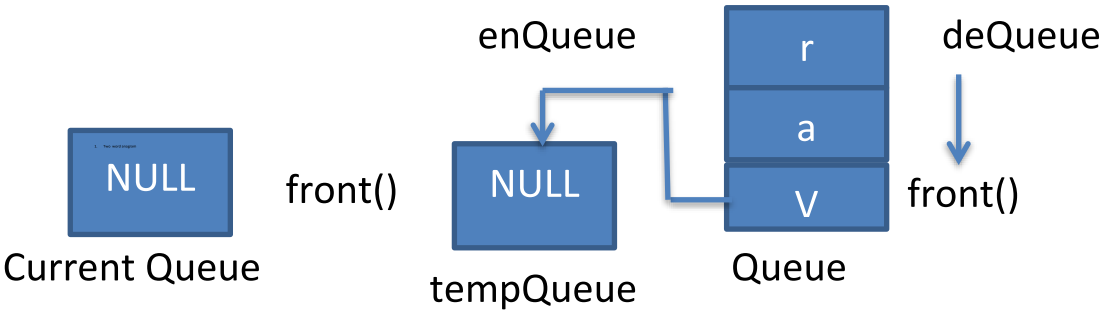


2. End of function.

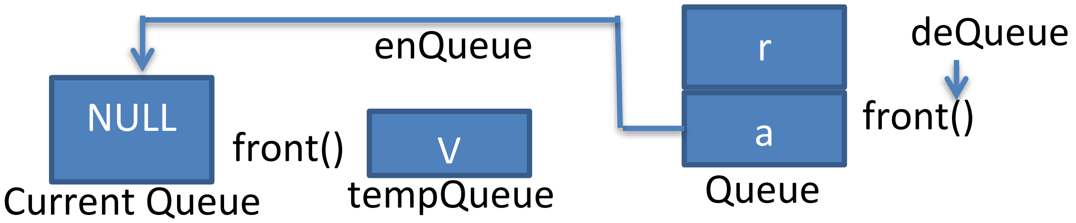


CopyReverse(queue<char> Queue)

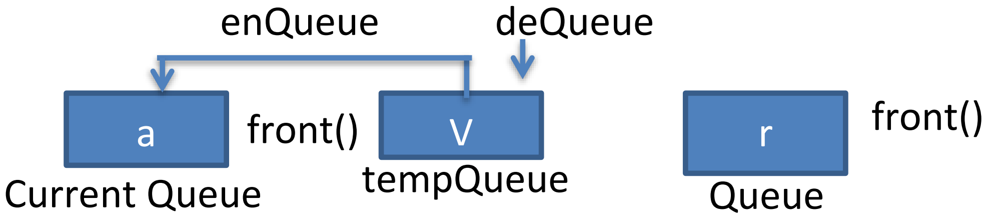
1. Beginning of function .



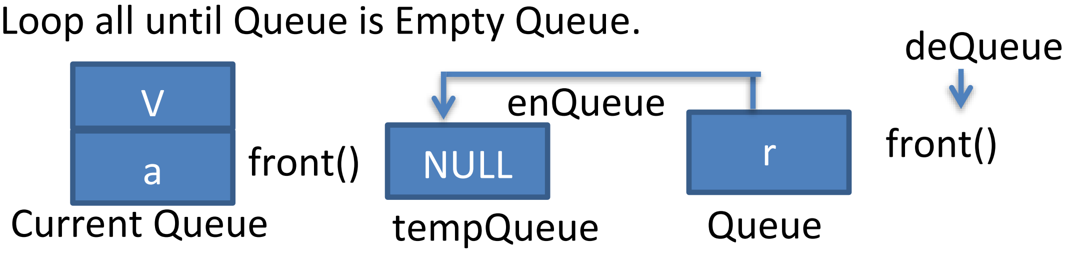
2. After enQueue Queue to tempQueue and deQueue Queue.



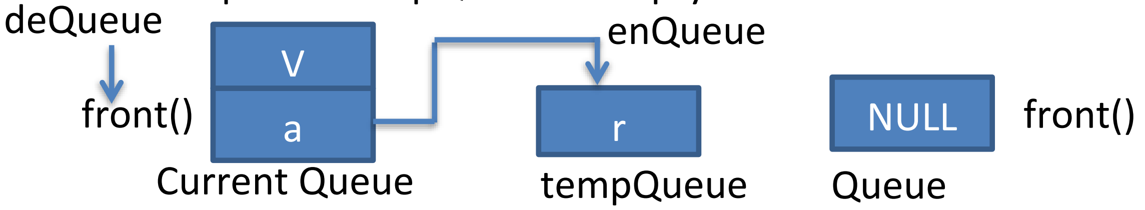
3. After enQueue Queue to current Queue and deQueue Queue.



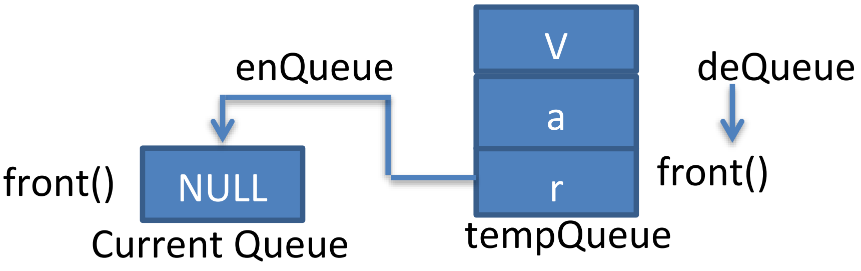
3. Loop all until Queue is Empty Queue.



Loop until tempQueue is empty.



Loop until current Queue is empty.



CODING : Linklist.h

```
#include <iostream>
using namespace std;
//Linklist.h

template <class T>
class ListNode
{
public:
        T value;
        ListNode<T> *next;
        ListNode (T nodeValue){
                value = nodeValue;
      next = NULL;
        }
};

template <class T>
class LinkedList
{
        private:
                ListNode<T> *head;
                int count;

        public:
                LinkedList(){ head = NULL; count=0;}
                ~LinkedList();
                void appendNode(T); //to add node
                void insertNode(T); //to insert node
                void deleteNode(T); //to delete node
                void displayList() const; //to display all nodes
                void CopyList(LinkedList<string> List);
                int CountNodes(){return count;}
                void destroyList();
                void search(char);
                bool IsEmptyList();
                bool Found(T);
                T firstNode();
};
template <class T>
bool LinkedList<T>::IsEmptyList()
{
    ListNode<T> *nodePtr;
    nodePtr=head;
    if(nodePtr!=NULL)
        return true;
    else
        return false;
}
template <class T>
void LinkedList<T>::appendNode(T newValue)
{
    ListNode<T> *newNode;
    ListNode<T> *nodePtr;
    newNode = new ListNode<T>(newValue);

    if (!head)
    head = newNode;
    else
    {
    nodePtr = head;
    while (nodePtr->next)
    nodePtr = nodePtr->next;

    nodePtr->next = newNode;
    }
    count++;
}
template <class T>
void LinkedList<T>::displayList() const
{
    ListNode<T> *nodePtr;
    nodePtr = head;
    int i=1;
    nodePtr = nodePtr->next;
    while (nodePtr){
   cout <<"\n\t\t->"<<i<<". "<< nodePtr->value;
    nodePtr = nodePtr->next;
    i++;
    }
}
template <class T>
T LinkedList<T>::firstNode()
{
    ListNode<T> *nodePtr;
    nodePtr = head;
   return nodePtr->value;
}
template <class T>
void LinkedList<T>::search(char alphabet)
{
        ListNode<T> *nodePtr;
        bool found=false;
        nodePtr = head;
        char w[nodePtr->value.length()];
        int length=nodePtr->value.length();
        while(nodePtr){
                for(int i=0;i<length;i++)
                        w[i]=nodePtr->value[i];
     nodePtr = nodePtr->next;
        }
        nodePtr = head;
        while (nodePtr){
        if(nodePtr->value[0]==alphabet){
        found=true;
        cout<<"\t\t-> "<<nodePtr->value<<endl;
                }
     nodePtr = nodePtr->next;
        }
        if (!found)
          cout <<"\n\n\tInfo not found in the list....\n";
}
template <class T>
bool LinkedList<T>::Found(T word)
{
        ListNode<T> *nodePtr;
        bool found=false;
        nodePtr = head;

        while (nodePtr &&!found){
        if(nodePtr->value==word)
        found=true;
        else
        nodePtr = nodePtr->next;
        }
        return found;
}
template <class T>
void LinkedList<T>::insertNode(T newValue)
```

```
{
        ListNode<T> *newNode;
        ListNode<T> *nodePtr;
        ListNode<T> *previousNode = NULL;
        newNode = new ListNode<T>(newValue);

        if (!head){
    head = newNode;
    newNode->next = NULL;
        }
        else {
        nodePtr = head;
        previousNode = NULL;

        while (nodePtr != NULL && nodePtr->value < newValue){
        previousNode = nodePtr;
        nodePtr = nodePtr->next;
        }

        if (previousNode == NULL){
        head = newNode;
        newNode->next = nodePtr;
        }
        else {
        previousNode->next = newNode;
        newNode->next = nodePtr;
        }
        }
        count++;
}
template <class T>
void LinkedList<T>::deleteNode(T searchValue)
{
        ListNode<T> *nodePtr;
        ListNode<T> *previousNode;

        if (!head)
        cout <<"List is Empty\n";

        if (head->value == searchValue){
        nodePtr=head;
        head=head->next;
        delete nodePtr;
        }
        else{
        nodePtr = head;

        while (nodePtr != NULL && nodePtr->value != searchValue){
        previousNode = nodePtr;
        nodePtr = nodePtr->next;
        }

        if (nodePtr){
        previousNode->next = nodePtr->next;
        delete nodePtr;
        }
        }
        count--;
}
template <class T>
void LinkedList<T>::CopyList(LinkedList<string> List)
{
        while(!List.IsEmptyList()){
            appendNode(List.firstNode());
            List.deleteNode(List.firstNode());
        }
}
template <class T>
void LinkedList<T>::destroyList()
{
        ListNode<T> *nodePtr;
        ListNode<T> *nextNode;
        nodePtr = head;

        while (nodePtr){
        nextNode = nodePtr->next;
        delete nodePtr;
        nodePtr = nextNode;
        count--;
        }

}
template <class T>
LinkedList<T>::~LinkedList()
{
        ListNode<T> *nodePtr;
        ListNode<T> *nextNode;
        nodePtr = head;
        while (nodePtr != NULL){
        nextNode = nodePtr->next;
        delete nodePtr;
        nodePtr = nextNode;
        }
        count=0;
}
```

CODING : Queue.h

```
#include <iostream>
using namespace std;
//Queue.h
template <class T>
class queueNode{
        public:
                T value; // Node value
                queueNode<T> *next; // Pointer to the next node
                // Constructor
                queueNode (T nodeValue){
                        value = nodeValue;
                        next = NULL;}
};
template <class T>
class queue {
        private:
                queueNode<T> *qFront;
                queueNode<T> *qRear;
                int count;
        public:
                void enQueue(T);
                void deQueue();
                T front(); //to return front value of a Queue
                bool IsEmptyQueue();
                void destroyQueue();
                void numQueueNode(){
                        return count;}
                void CopyReverse(queue<char>);
                void Copy(queue<char>);
                queue();
                ~queue();
};
template <class T>
void queue <T>::enQueue(T newValue){
        queueNode<T> *newNode;
        newNode = new queueNode<T>(newValue);
        if (qFront==NULL){
                qFront=newNode;
                qRear=newNode;
        }
        else {
                qRear->next=newNode;
                qRear=newNode;
        }
        count++;
}
template <class T>
void queue <T>::deQueue(){
        queueNode<T> *nodePtr;
        if (qFront==NULL)
                cout << "Queue is Empty...\n";
        else {
                nodePtr=qFront;
                qFront=qFront->next; //eqv to nodePtr->next
                delete nodePtr;
                count--;
        }
}
template <class T>
T queue <T>::front() {
        return qFront->value;
}
template <class T>
        bool queue<T>::IsEmptyQueue() {
        if (qFront==NULL)
                return true;
        else
                return false;
}
template <class T>
void queue <T>::destroyQueue(){
        queueNode<T> *nodePtr;
        while (!IsEmptyQueue()){ //while (qFront!=Null)
```

```
		nodePtr=qFront;
		qFront=qFront->next;
		delete nodePtr;
	}
	count=0;
}

template <class T>
void queue <T>::CopyReverse(queue<char> Queue){
	queue<char> tempQueue;
	tempQueue.enQueue(Queue.front());
	Queue.deQueue();
	enQueue(Queue.front());
	Queue.deQueue();
	enQueue(tempQueue.front());
	tempQueue.deQueue();
	while(!Queue.IsEmptyQueue()){
		tempQueue.enQueue(Queue.front());
		Queue.deQueue();
		while(!IsEmptyQueue()){
			tempQueue.enQueue(front());
			deQueue();
		}
		while(!tempQueue.IsEmptyQueue()){
			enQueue(tempQueue.front());
			tempQueue.deQueue();
		}
		count++;
	}
}
template <class T>
void queue <T>::Copy(queue<char> Queue){
	while(!Queue.IsEmptyQueue()){
		enQueue(Queue.front());
		Queue.deQueue();
		count++;
	}
```

```
}
template <class T>
queue <T>::queue(){
        qFront=NULL;
        qRear=NULL;
        count=0;
}
template <class T>
queue <T>::~queue(){
        destroyQueue();
}
```

CODING : Stack.h

```
#include <iostream>
using namespace std;
//Stack.h

template <class T>
class stackNode{
	public:
		T value; // Node value
		stackNode<T> *next; // Pointer to the next node
		// Constructor
		stackNode (T nodeValue){
			value = nodeValue;
			next = NULL;}
};

template <class T>
class stack {
	private:
		stackNode <T> *stackTop;
		int count;
	public:
		T Top(); // return top value of the stack
		void Pop(); //delete the top of the stack
		void Push(T); //add new data to the stack
		bool IsEmptyStack();
		int numStackNode(){
			return count;}
		void destroyStack();//delete all nodes in the stack
		void CopyStack(stack<char> );
		void SwapStack(int,int);
		void moveTopToBottom();
		stack();
		~stack();
};

template <class T>
```

```
stack <T> ::stack(){
        stackTop=NULL;
        count=0;
}

template <class T>
stack <T> ::~stack(){
        destroyStack();
}

template <class T>
T stack <T> ::Top(){
        if (!IsEmptyStack())
        return stackTop->value;
}

template <class T>
void stack <T> ::Pop(){
        stackNode<T> *nodePtr;
        if (IsEmptyStack())
                cout <<"Stack is Empty...\n";
        else {
                nodePtr=stackTop;
                stackTop=stackTop->next;
                delete nodePtr;
                count--;
        }
}

template <class T>
void stack <T> ::Push(T newValue){
        stackNode<T> *newNode;
        newNode = new stackNode<T>(newValue);
        if (stackTop==NULL) //if (IsEmptyStack()
                stackTop=newNode;
        else {
                newNode->next=stackTop;
        stackTop=newNode;
        count++;
    }
}

template <class T>
bool stack <T> ::IsEmptyStack() {
    if (stackTop==NULL)
        return true;
    else
        return false;
}

template <class T>
void stack <T> ::CopyStack(stack<char> Stack) {
    stack<T> tempStack;
    if(Stack.IsEmptyStack())
        cout<<"Stack Empty !"<<endl;
    else{
        while(!Stack.IsEmptyStack()){
            tempStack.Push(Stack.Top());
            Stack.Pop();
        }
        while(!tempStack.IsEmptyStack()){
            Push(tempStack.Top());
            tempStack.Pop();
        }
    }
 }
 template <class T>
void stack <T> ::SwapStack(int first,int sec) {
    stack<char> tempStack,tempStack1;
    int i=0;
    while(!IsEmptyStack()){
        if(i>sec)
            break;
        if(i==first){
            tempStack1.Push(Top());
            Pop();
        }
        else{
            tempStack.Push(Top());
            Pop();
        }
        i++;
    }
    Push(tempStack1.Top());
    tempStack1.Pop();
    tempStack1.Push(tempStack.Top());
    tempStack.Pop();
    if(!tempStack.IsEmptyStack()){
        while(!tempStack.IsEmptyStack()){
            Push(tempStack.Top());
            tempStack.Pop();
        }
    }
    Push(tempStack1.Top());
    tempStack1.Pop();
}
template <class T>
void stack <T> ::moveTopToBottom(){
    stack<char> tempStack,tempStack1;
    tempStack1.Push(Top());
    Pop();
    while(!IsEmptyStack()){
        tempStack.Push(Top());
        Pop();
    }
    Push(tempStack1.Top());
    tempStack1.Pop();
    while(!tempStack.IsEmptyStack()){
        Push(tempStack.Top());
        tempStack.Pop();
    }
```

```
}
template <class T>
void stack <T> ::destroyStack() {
        stackNode <T> *nodePtr;
        while (!IsEmptyStack()) { //while (stackTop!=NULL)
                nodePtr=stackTop;
                stackTop=stackTop->next;
                delete nodePtr;
        }
 }
```

CODING : MainProgram.cpp

```
#include <iostream>
#include <string.h>
#include <windows.h>
#include <fstream>
#include "Linklist.h"
#include "Stack.h"
#include "Queue.h"
using namespace std;
int readfile(LinkedList<string>[],int[]);
void StoreintoChar(LinkedList<string>[],stack<char>[],queue<char>[],int);
void Anagram(LinkedList<string>[],int,int[]);
void copyStack(stack<char>,stack<char>&,int);
void copyQueue(queue<char>,queue<char>&,int);
int main(){
	int size=1000;
	LinkedList<string> List[size];
	stack<char> Stack[size],Stack1[size];
	queue<char> Queue[size],Queue1[size];
	int StackQueueSize[size];
	int C,c=0,var;
	string word,word2;

	cout<<"\n\tWelcome to Anagram Generator !!"<<"\n\n\t";
	system("pause");
	system("cls");
	cout<<"\n\n\t\tRead File . . . . .\n\t";
	size=readfile(List,StackQueueSize);
	system("pause");

	cout<<"\tWords inside file : "<<"\n\n";
	for(int i=0;i<size;i++){
		cout<<"\t\t->"<<List[i].firstNode();
		cout<<endl;
	}
	StoreintoChar(List,Stack,Queue,size);
	system("pause");
	system("cls");

	do{	//display menu
		cout<<"\n\n\t--------------------------------------------------------------------------------
\n\t\t\t\tAnagram Generator";
		cout<<"\n\t--------------------------------------------------------------------------------\t";

		cout<<"\n\n\t\tChoose the following option :\n\n\t\t[1]-List of
Words."<<"\n\t\t[2]-All Anagram Permutation.";
```

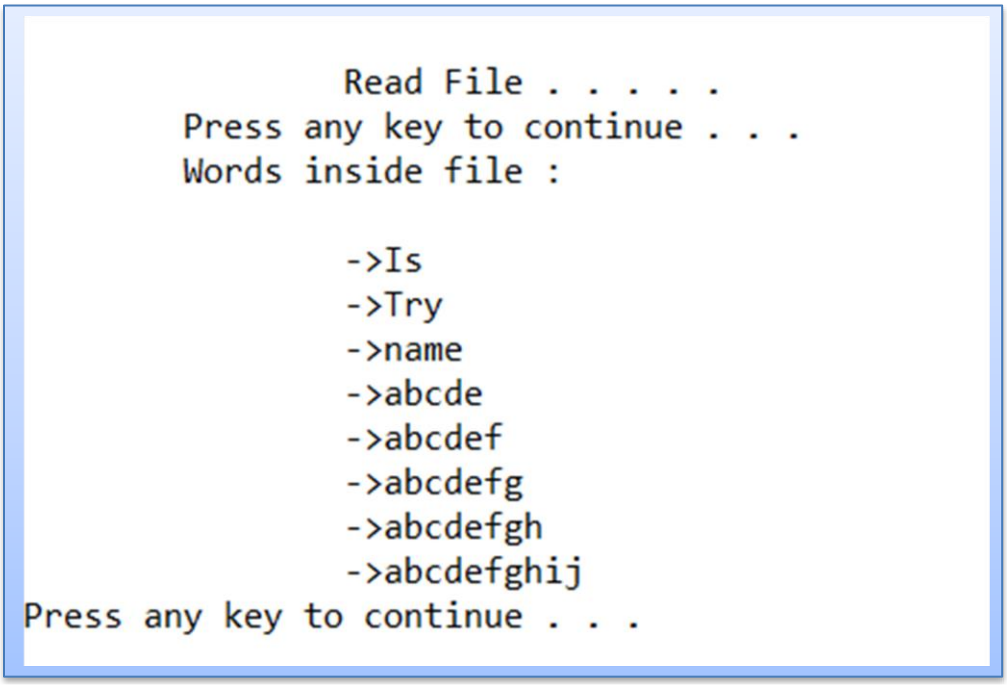
```
        cout<<"\n\t\t[3]-Anagram with Starting Letter. "<<"\n\t\t[4]-Add
Word/Words.";
        cout<<"\n\t\t[5]-Update Word and it Anagram."<<"\n\t\t[6]-Delete word and it
Anagram.";
        cout<<"\n\t\t[7]-Exit."<<"\n\n\t\tOption : ";
        cin>>C;system("CLS");
        //task that provide by the program
        switch(C){
            case 1:  cout<<"\n\tList of Words : "<<"\n\n";
                    for(int i=0;i<size;i++){
                        cout<<"\t\t->"<<List[i].firstNode();
                        cout<<endl;
                    }
                    system("pause");
                    system("CLS");
                    break;
            case 2: Anagram(List,size,StackQueueSize);
                    cout<<"\n\tList of Words with it Anagram
:"<<endl<<endl;
                    for(int i=0;i<size;i++){
                        cout<<"\n\t\tWord "<<i+1<<" -
"<<List[i].firstNode();
                        List[i].displayList();
                    }
                    system("pause");
                    system("CLS");
                    break;
            case 3: do{
                        cout<<"\n\tEnter Word that you want to search :
";
                        cin>>word;
                        bool found=false;
                        int i=0;
                        while(i<size){
                            if(word==List[i].firstNode()){
                                found=true;break;}
                            i++;
                        }
                        if(!found)
                            cout<<"\n   There is no '"<<word<<"'
word in this program !"<<endl<<endl;
                        else{
                            cout<<"\n\tEnter Starting letter for
Anagram : ";
                            char a;
                            cin>>a;
                            cout<<"\n\tAnagram : "<<endl;
                            List[i].search(a);
                            break;
                        }
                        system("pause");
                        system("CLS");
                        cout<<"\n   Do you want to keep searching ? [1-
yes,2-no]\n\tChoice :";
                        cin>>c;
                        if(c==2){
                            system("pause");system("cls");
                            break;}
                        else if(c==1){
                            system("pause");system("cls");}
                        else{
                            while(c!=1||c!=2){
                                cout<<"\tEnter Again : ";
                                cin>>c;
                            }
                        }
                    }while(c==1);
                    system("pause");
                    system("CLS");
                    break;
            case 4: cout<<"\n   How many word you want to enter ? \n   ";
                    cin>>var;
                    int i;
                    for(i=0;i<var;i++){
                        cout<<"\n\tWord "<<i+1<<" : ";
                        cin>>word;
                        List[i+size].appendNode(word);
                        StackQueueSize[i+size]=word.length();
                    }
                    size=size+i;
                    Anagram(List,size,StackQueueSize);
                    system("pause");
                    system("CLS");
                    break;
            case 5: do{
                        cout<<"\n\tEnter Word that you want to Update :
";
                        cin>>word;
                        bool found=false;
                        int i=0;
                        while(i<size){
                            if(word==List[i].firstNode()){
                                found=true;
                                List[i].destroyList();
                                break;
                            }
                            i++;
                        }
                        if(!found)
                            cout<<"\n   There is no '"<<word<<"'
word in this program !"<<endl<<endl;
                        else{
                            cout<<"\n\tEnter new updated word : ";
                            cin>>word2;
                            for(int j=0;j<StackQueueSize[i];j++){
                                Stack[i].Pop();
                                Queue[i].deQueue();
                            }
                            for(int j=0;j<word.length();j++){
                                Stack[i].Push(word[j]);
                                Queue[i].enQueue(word[j]);
                            }
                            List[i].appendNode(word);
                            Anagram(List,size,StackQueueSize);
                            cout<<"\n\t\tWord "<<i+1<<" -
"<<List[i].firstNode();

                            List[i].displayList();
                        }
                        system("pause");
                        system("CLS");
                        cout<<"\n   Do you want to keep Updating ? [1-
yes,2-no]\n\tChoice :";

                        cin>>c;
                        if(c==2){
                            system("pause");system("cls");
                            break;}
                        else if(c==1){
                            system("pause");system("cls");}
                        else{
                            while(c!=1||c!=2){
                                cout<<"\tEnter Again : ";
                                cin>>c;
                            }
                        }
                    }while(c==1);
                    system("pause");
                    system("CLS");
                    break;
            case 6: do{
                        cout<<"\n\tEnter Word that you want to delete
word and its anagram : ";

                        cin>>word;
                        bool found=false;
                        int i=0;
                        while(i<size){
                            if(List[i].firstNode()==word){
                                found=true;
                                break;
                            }i++;
                        }
                        if(!found)
                            cout<<"\n   There is no '"<<word<<"'
word in this program !"<<endl<<endl;
                        else{
                            for(int j=0;j<StackQueueSize[i];j++){

    List[i].deleteNode(List[i].firstNode());
                            }
                            while(i<size){
                                List[i].CopyList(List[i+1]);
                                List[i+1].destroyList();

    StackQueueSize[i]=StackQueueSize[i+1];
                                i++;
                            }
                        }
                        size--;
                        system("pause");
                        system("CLS");
                        cout<<"\n   Do you want to keep Deleting ? [1-
yes,2-no]\n\tChoice :";
                        cin>>c;
                        if(c==2){
                            system("pause");system("cls");
                            break;}
                        else if(c==1){
                            system("pause");system("cls");}
                        else{
                            while(c!=1||c!=2){
                                cout<<"\tEnter Again : ";
                                cin>>c;
                            }
                        }
                    }while(c==1);
                    system("pause");
                    system("CLS");
                    break;
            case 7:

                    system("pause");
                    system("CLS");
                    break;
```

```
			default:cout<<"\tChoice is not valid...\nEnter again\n";
		}
	}while(C!=9);
}
int readfile(LinkedList<string> List[],int StackQueueSize[]){
	ifstream readData;
	string name[1000];
	int i=0;

	readData.open("name.txt");
	while(!readData.eof()){
		if(readData.eof()==true)
			break;
		else{
			readData>>name[i];
			List[i].appendNode(name[i]);
			StackQueueSize[i]=name[i].length();
			i++;
		}
	}
	readData.close();
	return i;
}
void StoreintoChar(LinkedList<string> List[],stack<char> Stack[],queue<char> Queue[],int size){
	string name[size];
	int StackQueueSize[size];
	stack<char> Stack1[size];
	queue<char> Queue1[size];
	for(int i=0;i<size;i++){
		name[i]=List[i].firstNode();
		StackQueueSize[i]=name[i].length();
	}
	for(int a=0;a<size;a++){
		char charname[name[a].length()];
		for(int b=0;b<StackQueueSize[a];b++)
			charname[b]=name[a][b];
		for(int b=0;b<StackQueueSize[a];b++){
			Stack[a].Push(charname[b]);
			Queue[a].enQueue(charname[b]);
		}
	}
	cout<<"Stack : \t\tQueue :"<<endl<<endl;
	for(int i=0;i<size;i++){
		Stack1[i].CopyStack(Stack[i]);
		for(int j=0;j<StackQueueSize[i];j++){
			cout<<Stack[i].Top();
			Stack[i].Pop();
```

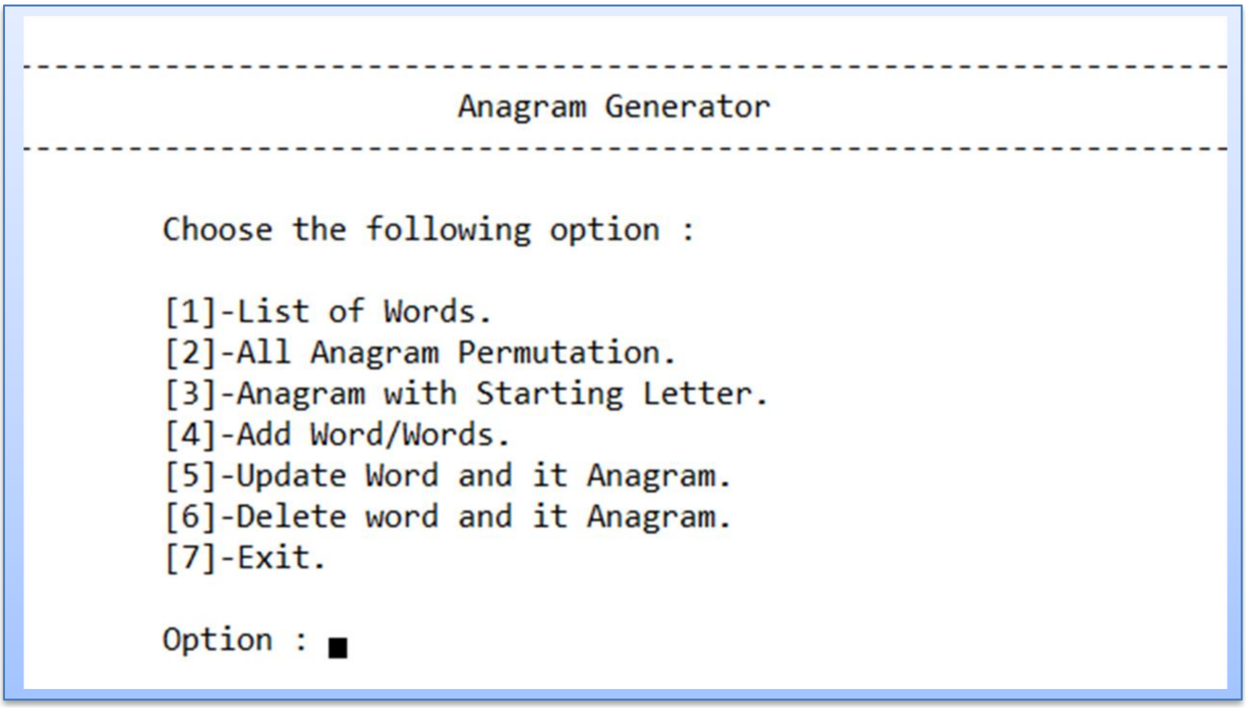
```
        }
        Stack[i].CopyStack(Stack1[i]);
        cout<<"\t\t\t";
        Queue1[i].Copy(Queue[i]);
        for(int j=0;j<StackQueueSize[i];j++){
            cout<<Queue[i].front();
            Queue[i].deQueue();
        }
        Queue[i].Copy(Queue1[i]);
        cout<<endl;
    }
}
void copyStack(stack<char> Stack,stack<char> &tempStack,int StackQueueSize){
    queue<char> tempQueue;
        for(int j=0;j<StackQueueSize;j++){
            tempQueue.enQueue(Stack.Top());
            Stack.Pop();
        }
        for(int j=0;j<StackQueueSize;j++){
            tempStack.Push(tempQueue.front());
            tempQueue.deQueue();
        }
}
void copyQueue(queue<char> Queue,queue<char> &tempQueue,int StackQueueSize){
        for(int j=0;j<StackQueueSize;j++){
            tempQueue.enQueue(Queue.front());
            Queue.deQueue();
        }
}
void Anagram(LinkedList<string> List[],int size,int StackQueueSize[]){
    stack<char> Stack[size],tempStack[size],tempStack2[size],tempStack3[size];
    queue<char> Queue[size],tempQueue[size];
    StoreintoChar(List,Stack,Queue,size);
    string word1, word2,word3,word4;
    for(int i=0;i<size;i++){
        for(int j=0;j<StackQueueSize[i];j++)
            Queue[i].deQueue();
    }
    for(int i=0;i<size;i++){
        if(List[i].CountNodes()==1){
            tempStack[i].CopyStack(Stack[i]);
            if(StackQueueSize[i]==2){
                word2="";
                for(int j=0;j<StackQueueSize[i];j++){
                    word2=word2+Stack[i].Top();
                    Stack[i].Pop();
                }
                List[i].appendNode(word2);
        Stack[i].CopyStack(tempStack[i]);
        Stack[i].SwapStack(0,1);
        word2="";
        for(int j=0;j<StackQueueSize[i];j++){
                word2=word2+Stack[i].Top();
                Stack[i].Pop();
        }
        List[i].appendNode(word2);
        Stack[i].CopyStack(tempStack[i]);
}
else if(StackQueueSize[i]==3){
        for(int j=0;j<StackQueueSize[i];j++){
                word1="";
                word2="";
                Stack[i].moveTopToBottom();
                tempStack2[i].CopyStack(Stack[i]);
                for(int k=0;k<StackQueueSize[i];k++){
                        word1=word1+Stack[i].Top();
                        Stack[i].Pop();
                }

                Stack[i].CopyStack(tempStack2[i]);
                for(int k=0;k<StackQueueSize[i];k++){
                        tempQueue[i].enQueue(tempStack2[i].Top());
                        tempStack2[i].Pop();
                }
                Queue[i].CopyReverse(tempQueue[i]);
                for(int k=0;k<StackQueueSize[i];k++){
                        word2=word2+Queue[i].front();
                        Queue[i].deQueue();
                        tempQueue[i].deQueue();
                }
                if(List[i].Found(word1)){
                        if(word1==List[i].firstNode())
                                List[i].appendNode(word1);}
                else
                        List[i].appendNode(word1);
                if(List[i].Found(word2)){
                        if(word2==List[i].firstNode())
                                List[i].appendNode(word2);}
                else
                        List[i].appendNode(word2);
        }
        for(int k=0;k<StackQueueSize[i];k++)
                Stack[i].Pop();
        Stack[i].CopyStack(tempStack[i]);
}
else {
```

```
                for(int j=0;j<StackQueueSize[i];j++){
                    word1="";
                    word2="";
                    Stack[i].moveTopToBottom();
                    tempStack2[i].CopyStack(Stack[i]);
                    for(int k=0;k<StackQueueSize[i];k++){
                        word1=word1+Stack[i].Top();
                        Stack[i].Pop();
                    }
                    Stack[i].CopyStack(tempStack2[i]);
                    for(int k=0;k<StackQueueSize[i];k++){
                        tempQueue[i].enQueue(tempStack2[i].Top());
                        tempStack2[i].Pop();
                    }
                    Queue[i].CopyReverse(tempQueue[i]);
                    for(int k=0;k<StackQueueSize[i];k++){
                        word2=word2+Queue[i].front();
                        Queue[i].deQueue();
                        tempQueue[i].deQueue();
                    }
                    if(List[i].Found(word1)){
                        if(word1==List[i].firstNode())
                            List[i].appendNode(word1);
                        else
                            cout<<"Same word with a word in link
list! "<<endl;}
                    else
                        List[i].appendNode(word1);
                    if(List[i].Found(word2)){
                        if(word2==List[i].firstNode())
                            List[i].appendNode(word2);
                        else
                            cout<<"Same word with a word in link
list! "<<endl;}
                    else
                        List[i].appendNode(word2);
                }
                for(int k=0;k<StackQueueSize[i];k++)
                    Stack[i].Pop();
                Stack[i].CopyStack(tempStack[i]);
                for(int j=0;j<StackQueueSize[i];j++){
                    Stack[i].moveTopToBottom();
                    tempStack2[i].CopyStack(Stack[i]);
                    for(int k=0;k<(StackQueueSize[i]-1);k++){
                        Stack[i].SwapStack(k,k+1);
                        tempStack3[i].CopyStack(Stack[i]);
                        word3="";
                        word4="";
                    for(int m=0;m<StackQueueSize[i];m++){
                        word3=word3+Stack[i].Top();
                        Stack[i].Pop();
                    }
                    for(int m=0;m<StackQueueSize[i];m++){

tempQueue[i].enQueue(tempStack3[i].Top());
                        tempStack3[i].Pop();
                    }
                    Queue[i].CopyReverse(tempQueue[i]);
                    for(int m=0;m<StackQueueSize[i];m++){
                        word4=word4+Queue[i].front();
                        Queue[i].deQueue();
                        tempQueue[i].deQueue();
                    }
                    if(List[i].Found(word3)==false)
                        List[i].appendNode(word3);
                    if(List[i].Found(word4)==false)
                        List[i].appendNode(word4);
                    Stack[i].CopyStack(tempStack2[i]);
                }
                for(int k=0;k<StackQueueSize[i];k++)
                    tempStack2[i].Pop();
            }
            for(int k=0;k<StackQueueSize[i];k++)
                Stack[i].Pop();
            Stack[i].CopyStack(tempStack[i]);
            int a=StackQueueSize[i]-1;
            while(a!=1){
                word1="";
                word2="";
                if(a==StackQueueSize[i]-1){
                    int count=1;
                    Stack[i].SwapStack(count,a);
                    tempStack2[i].CopyStack(Stack[i]);
                    for(int k=0;k<StackQueueSize[i];k++){
                        word1=word1+Stack[i].Top();
                        Stack[i].Pop();
                    }
                    Stack[i].CopyStack(tempStack[i]);
                    for(int k=0;k<StackQueueSize[i];k++){

tempQueue[i].enQueue(tempStack2[i].Top());
                        tempStack2[i].Pop();
                    }
                    Queue[i].CopyReverse(tempQueue[i]);
                    for(int k=0;k<StackQueueSize[i];k++){
                        word2=word2+Queue[i].front();
                        Queue[i].deQueue();
                        tempQueue[i].deQueue();
                    }
                    if(List[i].Found(word1)==false)
                        List[i].appendNode(word1);
                    if(List[i].Found(word2)==false)
                        List[i].appendNode(word2);
                }
                else{
                    int count=0;
                    while(count<a-1){
                        word3="";
                        word4="";
                        Stack[i].SwapStack(count,a);
                        tempStack2[i].CopyStack(Stack[i]);
                        for(int k=0;k<StackQueueSize[i];k++){
                            word1=word1+Stack[i].Top();
                            Stack[i].Pop();
                        }
                        Stack[i].CopyStack(tempStack[i]);
                        for(int k=0;k<StackQueueSize[i];k++){

                    tempQueue[i].enQueue(tempStack2[i].Top());
                            tempStack2[i].Pop();
                        }
                        Queue[i].CopyReverse(tempQueue[i]);
                        for(int k=0;k<StackQueueSize[i];k++){
                            word2=word2+Queue[i].front();
                            Queue[i].deQueue();
                            tempQueue[i].deQueue();
                        }
                        if(List[i].Found(word3)==false)
                            List[i].appendNode(word3);
                        if(List[i].Found(word4)==false)
                            List[i].appendNode(word4);
                        count++;
                    }
                }
                a--;
            }
        }
    }
}}
```

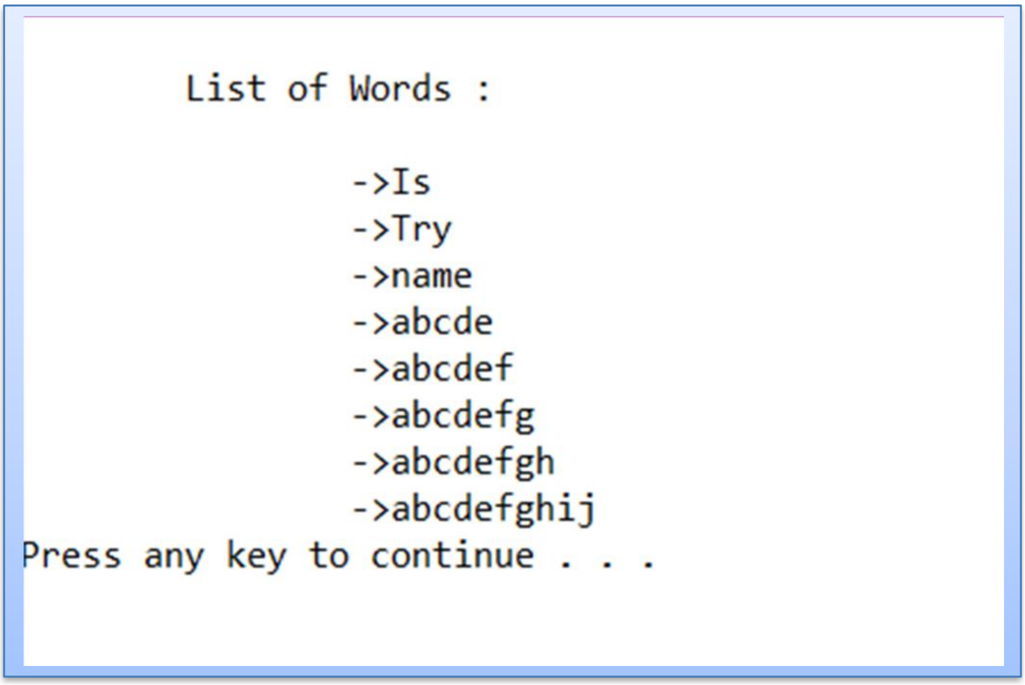
OUTPUT :



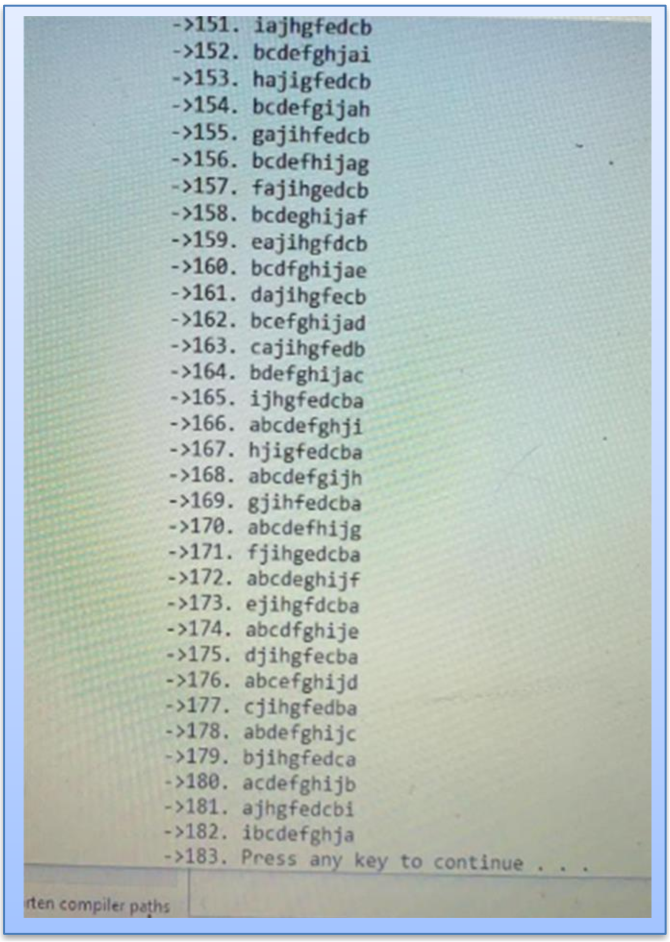
```
                Read File . . . . .
        Press any key to continue . . .
        Words inside file :

                ->Is
                ->Try
                ->name
                ->abcde
                ->abcdef
                ->abcdefg
                ->abcdefgh
                ->abcdefghij
Press any key to continue . . .
```



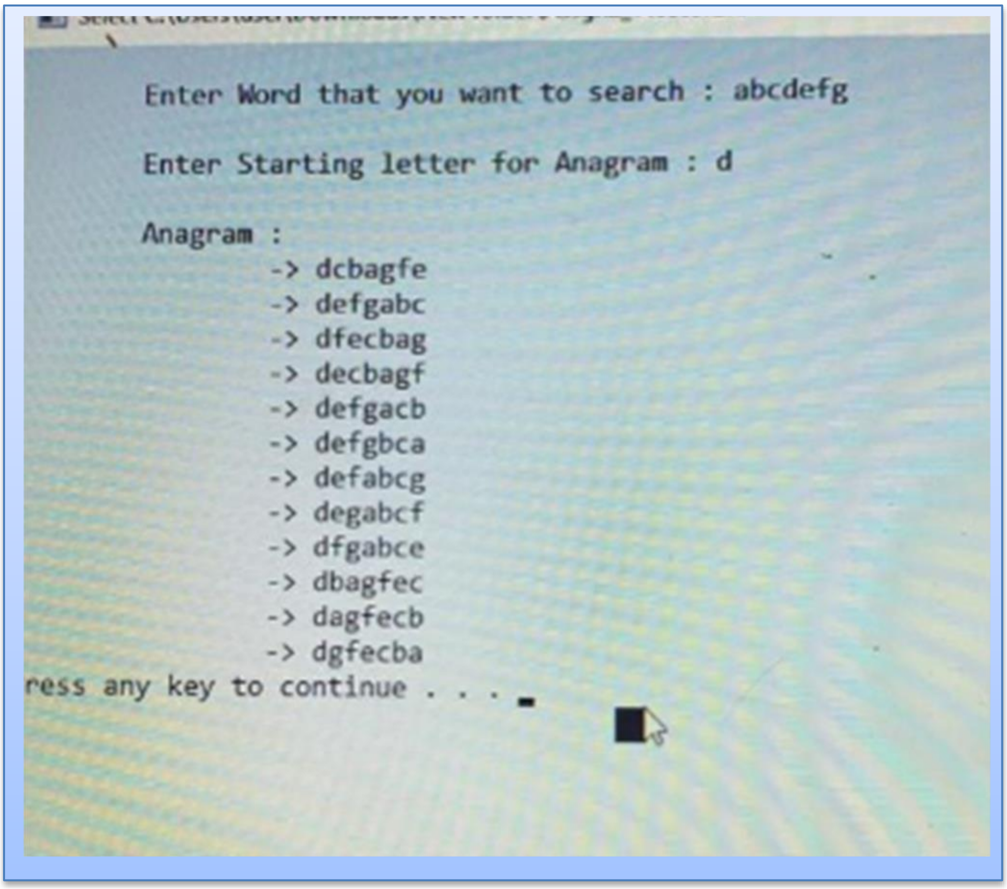
```
------------------------------------------------------------
                      Anagram Generator
------------------------------------------------------------

       Choose the following option :

       [1]-List of Words.
       [2]-All Anagram Permutation.
       [3]-Anagram with Starting Letter.
       [4]-Add Word/Words.
       [5]-Update Word and it Anagram.
       [6]-Delete word and it Anagram.
       [7]-Exit.

       Option : █
```



```
        List of Words :

                ->Is
                ->Try
                ->name
                ->abcde
                ->abcdef
                ->abcdefg
                ->abcdefgh
                ->abcdefghij
Press any key to continue . . .
```



```
->151. iajhgfedcb
->152. bcdefghjai
->153. hajigfedcb
->154. bcdefgijah
->155. gajihfedcb
->156. bcdefhijag
->157. fajihgedcb
->158. bcdeghijaf
->159. eajihgfdcb
->160. bcdfghijae
->161. dajihgfecb
->162. bcefghijad
->163. cajihgfedb
->164. bdefghijac
->165. ijhgfedcba
->166. abcdefghji
->167. hjigfedcba
->168. abcdefgijh
->169. gjihfedcba
->170. abcdefhijg
->171. fjihgedcba
->172. abcdeghijf
->173. ejihgfdcba
->174. abcdfghije
->175. djihgfecba
->176. abcefghijd
->177. cjihgfedba
->178. abdefghijc
->179. bjihgfedca
->180. acdefghijb
->181. ajhgfedcbi
->182. ibcdefghja
->183. Press any key to continue . . .
```



```
       Enter Word that you want to search : abcdefg

       Enter Starting letter for Anagram : d

       Anagram :
               -> dcbagfe
               -> defgabc
               -> dfecbag
               -> decbagf
               -> defgacb
               -> defgbca
               -> defabcg
               -> degabcf
               -> dfgabce
               -> dbagfec
               -> dagfecb
               -> dgfecba
ress any key to continue . . .
```