KATHEERAVAN A/L BALASUBRAMANIAM

000530-07-0621

147744

GROUP D

CPT 113 ASSIGNMENT 2

Dr. NUR HANA BINTI SAMSUDIN

29 MAY 2020

CONTENTS

Problem Analysis

The programs main purpose is to generate all the possible anagrams of a word. For a word with n-length, there are n! numbers of possible anagrams. Words are read from a file and words can also be entered by user. After all the words has been generated, user must be able to see all the possible anagrams of a word or all the anagrams starting with a specific alphabets. User must be able to also edit an existing word where it will be replaced by a new word. Besides that, user must also be able to delete a word and also its anagrams. There should not be any duplicate words in the program. There should not be any duplicates in the anagrams of a word when generated. The anagrams does not need to have a meaning or displayed in specific order.

Specific Requirements

1) Input
   -words with n-letters read from file
   -words with n-letters input by user

2) Process
   -generates all possible anagram of a word
   -reading words from file
   -appending words to the link list
   -inserting words to the link list
   -deleting a word from the link list
   -editing a word in the link list
   -checking for duplicates in words in link list and also their anagrams

3) Output
   -displaying all possible anagrams of a word
   -displaying all possible anagrams of a word staring with specific letter
   -displaying all the word in the main link list

4) Constrains
   -input validation for integer variables will not work if an alphabet or symbol is entered.
   -this program will not work if the word read from file or entered by user contains two or more same letters in it
   -the longer the word the longer the time taken to generate anagrams.

UML Diagrams for Classes

queue

queueNode
-value : T
-*next : queueNode

-*front : queueNode
-*rear : queueNode

+queue() { front = NULL, rear = NULL, count = 0; }
+~queue() { clear(); }
+enqueue(T) :void
+dequeue(T &) : void
+isempty() const : void
+clear() : void

stack

stackNode
-value: T
-*next : stackNode

-*top : stackNode;

+ stack() { top = NULL; }
+ ~stack();
+ push(T) : void
+ pop(T &) : void
+ isempty() const : void

Linklist

ListNode
-value : T
-*next : ListNode
-*permute : ListNode

-*head : ListNode

+ append(T) : void
+ append2(T,T) : void
+ display() : void
+ display2(T) : void
+ display3(T, T) : void
+ insert(T,T) : void
+ deleteNode(T) : void
+ validatelist(T, int &) : void
+ Linklist()
+ ~Linklist()

Psuedocode of Program

1) **Class Linklist**
   1.0 private *head
   2.0 struct value
   3.0 struct *next
   4.0 struct * permute
   5.0 public append(T) to append word to main list
   6.0 public append2(T) to append anagrams of word in permute pointer of each node
   7.0 public display(T) to display all word in list
   8.0 public display2(T) to display all anagram in a word
   9.0 public display3(T,T) to display anagrams of a word starting with specific letter
   10.0 public insert(T,T) to insert a word after another word
   11.0 public deleteNode(T) to delete a word and its anagrams
   12.0 public validatelist(T,int&) to check if a word entered by user already in list

2) **Class stack**
   1.0 private *top
   2.0 private struct value
   3.0 private struct *next
   4.0 public push(T) to push a word to stack
   5.0 public pop(T &) to pop a word from stack
   6.0 public isempty() const to check whether stack is empty

3) **class queue**
   1.0 private *front
   2.0 private *rear
   3.0 private count
   4.0 private struct value
   5.0 private struct *next
   6.0 pub;ic enqueue(T) to enqueue a word to queue
   7.0 public dequeue(T &) to dequeue a word to a queue
   8.0 public isempty() const to check whether queue is empty
   9.0 public clear() to be used as an destructor

4) **Linklist<T>::~Linklist()**
   1.0 Start
   2.0 creating a node called nodeptr
   3.0 creating a node called nextnode
   4.0 declaring nodeptr as head
   5.0 while nodeptr not equals to null
      5.1 nextnode is declared as next node of nodeptr
      5.2 deleting nodeptr
      5.3 declaring nodeptr as nextnode
   6.0 endwhile
   7.0 End

5) void Linklist<T>::append(T w)

1.0 start
2.0 creating two nodes called nodePtr and newData
3.0 declaring newData as new node
4.0 setting value of newData as w
5.0 pointing the newData to null
6.0 if no head
    6.1 head is declared as newData
7.0 else
    7.1 nodeptr is declared as head
    7.2 while nodeptr pointing to next
    7.3 nodeptr is declared as next point of nodeptr
    7.4 endwhile
8.0 nodptr points to newData
9.0 endif
10.0 End

6) void Linklist<T>::append2(T w,T word)

1.0 Start
2.0 Creating two node called newData and nodePtr
3.0 Declaring newData as new node
4.0 Setting value of newData node to w
5.0 Declaring nodePtr as head
6.0 While nodePtr not null and value of nodePtr not word
    6.1 nodeptr will point to next node
7.0 if value of newData not word
    7.1 while nodeptr pointing to next(permute) node
    7.2 nodeptr point to next node(permute)
    7.3 endwhile
    7.4 nodePtr permute points to newData
8.0 endif
9.0 end

7) void Linklist<T>::display()

1.0 start
2.0 creating two nodes called nodeptr and node
3.0 declaring int num as 1
4.0 declaring nodePtr as head
5.0 while nodeptr
    5.1 display num and nodeptr
    5.2 nodeptr points to next node
    5.3 increase num by 1
    5.4 endwhile
6.0 end

8) void Linklist<T>::display2(T w)

1.0 start
2.0 creating two nodes called nodeptr and node
3.0 declaring int num as 1
4.0 if no head
    4.1 Display message
5.0 else
    5.1 declaring nodeptr as head
    5.2 while value of nodeptr is not w
    5.3 nodeptr points to next node
    5.4 endwhile
    5.5 while nodeptr
    5.6 display num and nodeptr
    5.7 nodeptr points to next(permute) node (nodeptr=nodeptr->permute)
    5.8 increase num by 1
    5.9 endwhile
    5.10 display w and num-1
6.0 endif
7.0 end

9) void Linklist<T>::display3(T w,T a)

1.0 start
2.0 creating a node called nodeptr
3.0 initialiase num as 1 and count as 0
4.0 declaring nodeptr as head
5.0 if not head
    5.1 display message
6.0 else
    6.1 while nodeptr value is not w
    6.2 nodeptr points to next node
    6.3 endwhile
    6.4 while nodeptr
    6.5 word is declared as nodeptr
    6.6 if variable word's first index is same as a first index
    6.7 display num and word
    6.8 increase num by one
    6.9 end if
    6.10 nodeptr points to next(permute) node
    6.11 endwhile
    6.12 cout num-1 and a
7.0 endif
8.0 end

10) void Linklist<T>::insert(T w,T prev)

1.0 start
2.0 creating three node called nodeptr,node and newdata
3.0 declaring newData as new node
4.0 setting value of newData node as w
5.0 if value of head is prev
    5.1 newdata points to next node of head
    5.2 head point to newData
6.0 else
    6.1 declaring nodeptr as head
    6.2 declaring node as head
    6.3 while value of nodeptr is not prev
    6.4 nodeptr point to next node
    6.5 node is declared as next node of nodeptr
    6.6 endwhile
    6.7 nodeptr points to newData
    6.8 newData points to node
7.0 endif
8.0 end

11) void Linklist<T>::deleteNode(T w)

1.0 start
2.0 creating 4 nodes called nodeptr,previousNode,temp and node
3.0 if no head returns
4.0 endif
5.0 if value of head is w
    5.1 node is declared as head
    5.2 while value of node is not w
    5.3 node points to next node
    5.4 endwhile
    5.5 while node
    5.6 temp points to the next(permute) node
    5.7 if value of node is not w
    5.8 delete node
    5.9 endif
    5.10 node is declared as temp
    5.11 endwhile
    5.12 nodeptr is declared as head
    5.13 head points to the next node of head
    5.14 delete nodeptr
6.0 else
    6.1 nodeptr is declared as head
    6.2 while value of nodeptr is not w
    6.3 nodeptr points to the next node
    6.4 endwhile
    6.5 while nodeptr
    6.6 temp points to the next(permute) node
    6.7 if nodeptr value is not w

6.8 delete nodeptr
6.9 endif
6.10 nodeptr is declared as temp
6.11 endwhile
6.12 node is declared as head
6.13 previousNode is declared as head
6.14 while value of nodePtr is not w
6.15 previousNode id declared as node
6.16 node points to the next node
6.17 endwhile
6.18 if node
6.19 previous node points to next node of node
6.20 delete node
6.21 endif

7.0 endif
8.0 display w
9.0 end

12) void Linklist<T>::validatelist(T w, int & list)

1.0 start
2.0 creating a node called nodeptr
3.0 declaring nodeptr as head
4.0 while nodeptr
4.1 if value of nodeptr is w
4.2 declare list as 1
4.3 endif
4.4 nodeptr point to next node
5.0 endwhile
6.0 end

13) void Linklist<T>::displayana()

1.0 start
2.0 create nodes called nodeptr,print
3.0 declare nodeptr as head
4.0 while nodeptr
4.1 declare print as nodeptr
4.2 while print
4.3 display print
4.4 point print node to next(permute) node
4.5 endwhile
4.6 point nodeptr to next node
5.0 endwhile
6.0 end

14) stack<T>::~stack()

1.0 start
2.0 creating two node caleed nodeptr and nextnode
3.0 declaring nodeptr as top
4.0 while nodeptr is not null
    4.1 nextnode declared as next node of nodeptr
    4.2 delete nodeptr
    4.3 nodeptr declared as next node
5.0 endwhile
6.0 end

15) void stack<T>::push(T w)

1.0 start
2.0 creating a node called newnode
3.0 declaring newnode as new node
4.0 setting value of newnode as w
5.0 pointing newnode to null
6.0 if stack is empty
    6.1 declaring top as newnode
    6.2 pointing newnode to null
7.0 else
    7.1 pointing newnode to top
    7.2 declaring top as newnode
8.0 endif
9.0 end

16) void stack<T>::pop(T & w)

1.0 start
2.0 craeting a node called temp
3.0 if stack is empty
    3.1 display message
4.0 else
    4.1 declaring w as value of top
    4.2 pointing temp to next node of top
    4.3 delete top
    4.4 declaring top as temp
5.0 endif
6.0 end

17) bool stack<T>::isempty() const

1.0 start
2.0 if not top
    status =true
3.0 else
    status=false
4.0 return status
5.0 end

18) void queue<T>::enqueue(T w)

1.0 start
2.0 creating a node called newnode
3.0 declaring newnode as new node
4.0 setting value of newnode to w
5.0 pointing new node to null
6.0 if queue is empty
    6.1 declare front as newnode
    6.2 declaring rear as newnode
7.0 else
    7.1 pointing rear to newnode
    7.2 declaring rear as newnode
8.0 endif
9.0 count is increased by one
10.0 end

19) void queue<T>::dequeue(T &w)

1.0 start
2.0 creating a node called nodeptr
3.0 if queue is empty
    3.1 display message
4.0 else
    4.1 declaring w as value of front
    4.2 declaring nodeptr as front
    4.3 delete nodeptr
    4.4 decrease count by one
5.0 endif
6.0 end

20) bool queue<T>::isempty() const

1.0 start
2.0 if front equals to null return true
3.0 else return false
4.0 end

21) void queue<T>::clear()

1.0 start
2.0 create node called nodeptr
3.0 while queue is not empty
    3.1 declare nodeptr as front
    3.2 pointing front to the next node
    3.3 delete nodeptr
4.0 endwhile
5.0 declare count as 0
6.0 end

22) int main()

```
1.0 start
2.0 input filename
3.0 opening file
4.0 input validation for filename
5.0 if dataReadFile
    while not end of file
        read word from file
        call function of l.append()
        call function of anagram
    endwhile
    endif
6.0 closing file
7.0 while begin equals to 1
    input menuchoice
    input validation for menuchoice
    switch(menuchoice)
8.0 switch case 1
    function to print all words in the list
    break
9.0 switch case 2
    input search
    input validation for search
    while menu equals to 1
        input mchoice
        switch(mchoice)
        case 1 display all words in list
        case 2 (display starting with specific word)
                input alpha
                input validation for alpha
                displaying all anagram of word search staring with alpha
                break
        case 3 (update word)
                input nword
                input validation of nword
                insert nword after search in list
                delete search
                generate anagram for nword
                displaying anagram of n word
                search equals to nword
                break


        case 4 (add word after searched word)

                input nword
                input validation for nword
                insert nword after searched word
```

```
                generate anagram for nword
                display nword and search
                display anagram of nword
                break
        case 5 (delete searched word)
                delete searched word
                menu equals to zero
                break
        case 6 (exit search menu)
                menu equals to zero
                break
        endswitch
        endif
        endwhile
        break
10.0    switch case 3
        input qty
        input validation for qty
        for I equals to 0
                input appending
                input validation for appending
                append qppending to list
                generate anagram for appending
                display anagram for appending
                increase I by one
        end for
        break
11.0    switch case 4
        displaying all anagrams of all words.
        Break
12.0    switch case 5
        begin equals to zero
        break
13.0    endswitch
14.0    endwhile
15.0    end
```

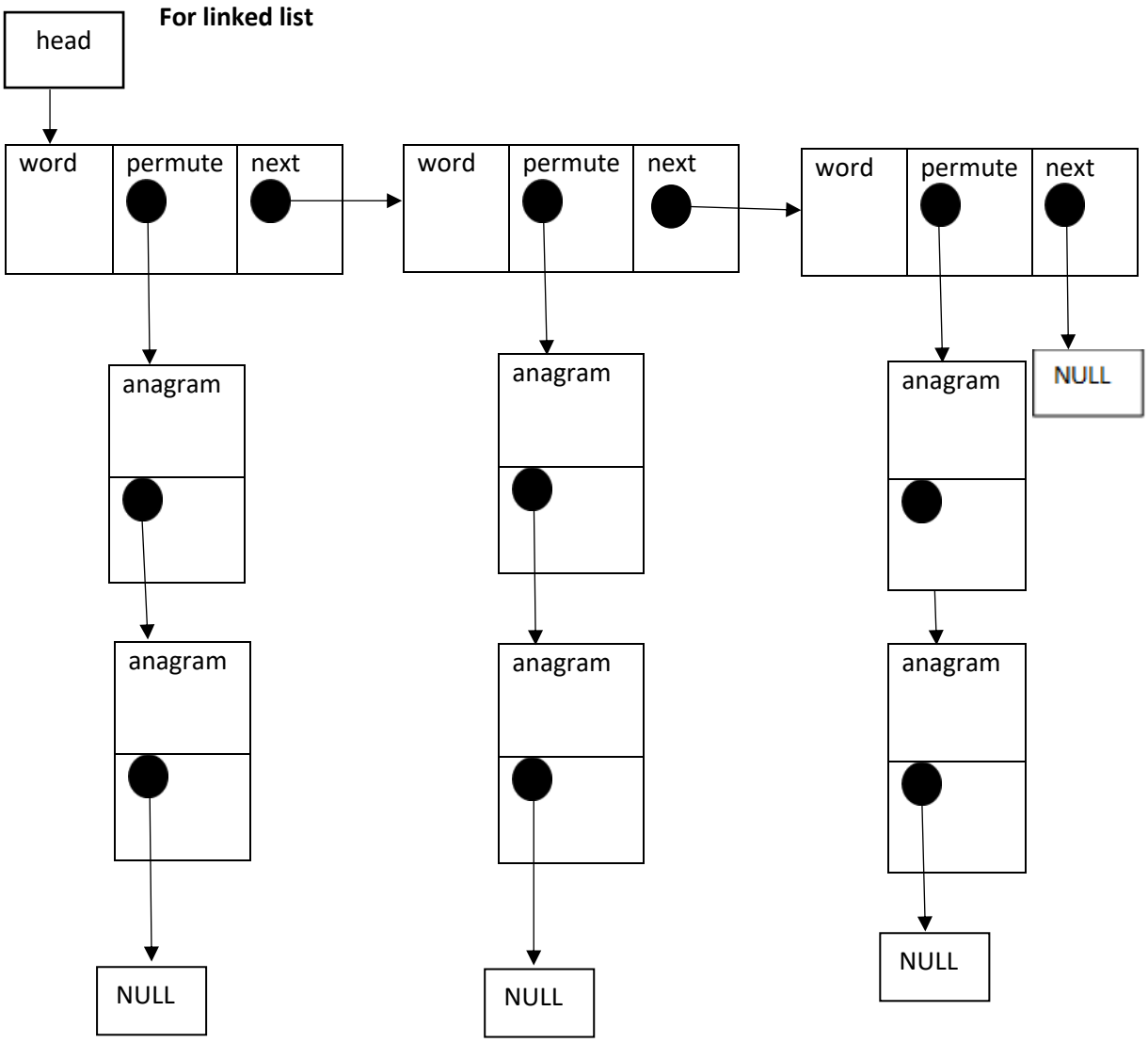
23) void anagram()

```
1.0 start
2.0 enqueuing word into queue
3.0 for i=0
        if stack is empty num equals to 1
        else if queue is empty num equals to 2

        if num  equals to 1
                while queue is not empty
```

```
                dequeue word from queue
                for x=0
                        temp equals to w[x](each charcter in word)
                        wrd equals to word
                        if w[x]equals to wrd[z] and counter is 0
                                swapping characters in wrd
                                push anagram to stack
                        endif
                        for z=0
                        if w[x]equals to wrd[z] and counter is 1
                                for u=0
                                        ifwrd[u] not equals to w[x] check equals to 1
                                        else if wrd[u] equals to w[x] check is 0
                                                break
                                        endif
                                        increase u by one
                                end for
                                if check is 1
                                        swapping word with character
                                        pushing anagram to stack
                                endif
                                check is equals to o
                        endif
                        increase z by one
                        endfor
                        increase x by one
                endfor
        endwhile
else if num equals to 2
        while queue is not empty
                pop word from stack
                for x=0
                        temp equals to w[x](each charcter in word)
                        wrd equals to word
                        for z=0
                        if w[x]equals to wrd[z]
                                for u=0
                                        ifwrd[u] not equals to w[x] check equals to 1
                                        else if wrd[u] equals to w[x] check is 0
                                                break
                                        endif
                                        increase u by one
                                end for
                                if check is 1
                                        swapping word with character
                                        pushing anagram to stack
                                endif
                                check is equals to o
```

```
                    endif
                    increase z by one
                    endfor
                    increase x by one
            endfor
        endwhile
    endif
    increase I by one
    num equals to 0
    counter equals to 1
    endfor (this is for the first loop that changes position in word in each loop)
    if ouput equals to 1
        while stack is not empty
            pop word
            append2 ( append anagram to the permute pointer of each word)
        endwhile

    else if output equals to 2

        while queue is not empty
            dequeue word
            append2 ( append anagram to the permute pointer of each word)
        endwhile

    endif

4.0 end
```

Data Structure Diagram



-In this assignment link list with two pointers were used.

-The two pointers are next and permute.

-Pointer next will point to the next node which contains the word read from file or added by user

-The last word will point to NULL.

-Pointer permute is used point to the next anagram of a word which is generated in anagram function.

-Each permute pointer of each words node points to that words anagrams node.

-The last anagram node will point to NULL.

-for example ali,abu and chong are read from file

-the words will be appended to list using next pointer

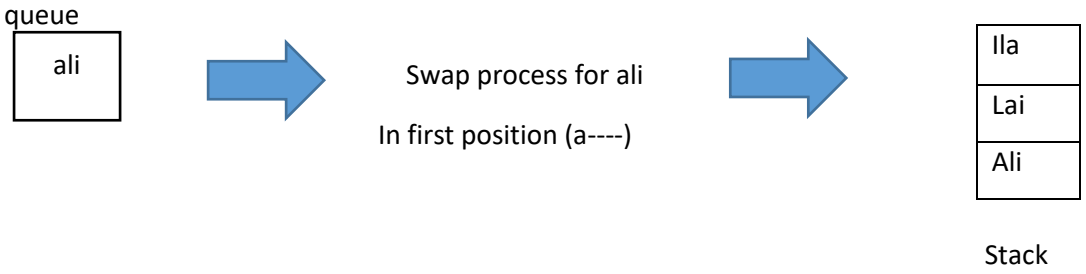
-after generating their anagrams each words anagrams will be appended using permute pointer

Queue and Stack

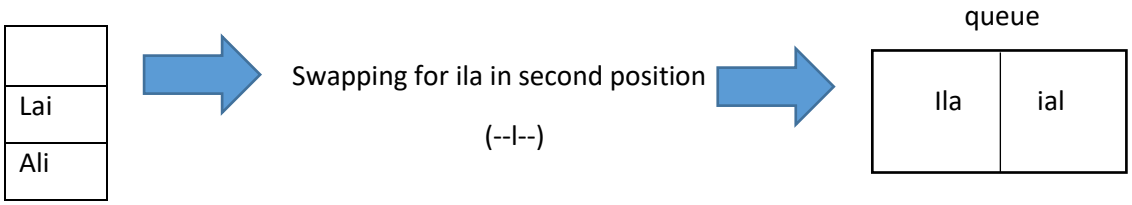
-queue and stack were used for generating anagram

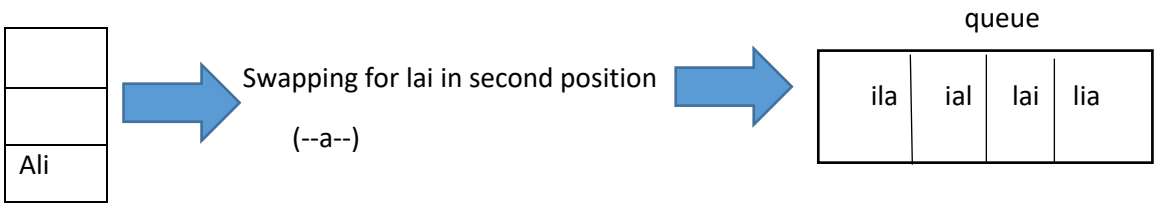
-for example first the word ali will be enqueue to queue.

-the anagram generated in first swap will than be popped to stack



-then for second swapping, words will be popped from stack one by one and after generating anagram and enqueued to queue





-this process will continue until swapping happen at the second last character of a word

-queue and stack will be used vise versa for each position of word during swapping process.(first position queue, second position stack and for third position queue again)

Output of Program

1) input file name with input validation

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Enter file name that contains words : kjhgf

Error!!! Wrong File Name !!!
Enter correct file name that contains words : words.txt
```

2) main menu

```
!!!MAIN MENU!!!

Enter a number according to the below options.
1.Display all the words in list.
2.Search
3.Append word to list.
4.Display Anagrams of All Words.
5.Exit
Options:
```

3) Displaying all words in list (option 1 in main menu)

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\AS

Words that are in the list

1 ali
2 abu
3 oil
4 lap
5 pod
6 cat
7 dog
Press any key to continue . . .
```

4) Searching (second option in main menu)
-input validation for searching word

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Enter the word that you want to search: hate
Error!!!The word you are searching does not exist in the list.
Enter correct word: chia
```

-search menu

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Searching...(WORD : ali)
1.Display All Anagrams of Word.
2.Display Anagrams of word staring with specific letter.
3.Update word.
4.Add word.
5.Delete word.
6.Exit
Option:
```

i)Displaying anagram of searched word(option 1 in search menu)

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Searching...(WORD : ali)
1.Display All Anagrams of Word.
2.Display Anagrams of word staring with specific letter.
3.Update word.
4.Add word.
5.Delete word.
6.Exit
Option: 1
Anagram of word ali are...

1 ali
2 ial
3 ila
4 lai
5 lia
6 ail

Word ali has 6 different anagrams.

Press any key to continue . . .
```

ii)Displaying anagram of searched word starting with specific letter (option 2 in search menu)

-input validation for letter

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Searching...(WORD : ali)
1.Display All Anagrams of Word.
2.Display Anagrams of word staring with specific letter.
3.Update word.
4.Add word.
5.Delete word.
6.Exit
Option: 2
Enter one of the letters in the searched word: k
Error!!!The letter that you entered does not exist in the word you are searching.
Enter the correct alphabets:
```

-displaying anagrams of word ali starting with letter l

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Searching...(WORD : ali)
1.Display All Anagrams of Word.
2.Display Anagrams of word staring with specific letter.
3.Update word.
4.Add word.
5.Delete word.
6.Exit
Option: 2
Enter one of the letters in the searched word: k
Error!!!The letter that you entered does not exist in the word you are searching.
Enter the correct alphabets: l

1 lai
2 lia

Word ali has 2 different anagrams starting with l.

Press any key to continue . . .
```

iii)Updating searched word with another word and displaying its anagram(option 3 in search menu)

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Searching...(WORD : ali)
1.Display All Anagrams of Word.
2.Display Anagrams of word staring with specific letter.
3.Update word.
4.Add word.
5.Delete word.
6.Exit
Option: 3

Update word.

Replace the word ali with........
Enter new word:mark
Word ali is deleted from the list together with its anagrams.

The word ali has been replaced to mark

Its anagrams are....

1 mark
2 mkra
3 mkar
4 mrka
5 mrak
6 makr
7 akrm
8 akmr
9 arkm
10 armk
11 amkr
12 amrk
13 rkam
14 rkma
15 rakm
16 ramk
17 rmka
18 rmak
19 kram
20 krma
21 karm
22 kamr
23 kmra
24 kmar

Word mark has 24 different anagrams.

Press any key to continue . . .
```

-in the word list word ali is replaced by mark

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\AS

Words that are in the list

1 mark
2 abu
3 oil
4 lap
5 pod
6 cat
7 dog
Press any key to continue . . .
```

iv)Adding a word after searched word(option 4 in search menu)

-input validation for new word(input by user)

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Searching...(WORD : mark)
1.Display All Anagrams of Word.
2.Display Anagrams of word staring with specific letter.
3.Update word.
4.Add word.
5.Delete word.
6.Exit
Option: 4
Adding word after the word mark.
Enter the word you want to add: dog
Error!!!The word alraedy exist in the list.
Enter new word: _
```

-new word(iron) added after the word mark

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Searching...(WORD : mark)
1.Display All Anagrams of Word.
2.Display Anagrams of word staring with specific letter.
3.Update word.
4.Add word.
5.Delete word.
6.Exit
Option: 4
Adding word after the word mark.
Enter the word you want to add: dog
Error!!!The word alraedy exist in the list.
Enter new word: iron

Word iron has been added after the word mark.

Its anagrams are....

1 iron
2 inor
3 inro
4 ionr
5 iorn
6 irno
7 rnoi
8 rnio
9 roni
10 roin
11 rino
12 rion
13 onri
14 onir
15 orni
16 orin
17 oinr
18 oirn
19 nori
20 noir
21 nroi
22 nrio
23 nior
24 niro

Word iron has 24 different anagrams.

Press any key to continue . . .
```

-new word added after mark in list

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM

Words that are in the list

1 mark
2 iron
3 abu
4 oil
5 lap
6 pod
7 cat
8 dog
Press any key to continue . . .
```

V)Deleting searched word

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe
Searching...(WORD : mark)
1.Display All Anagrams of Word.
2.Display Anagrams of word staring with specific letter.
3.Update word.
4.Add word.
5.Delete word.
6.Exit
Option: 5
Deleting word....
Word mark is deleted from the list together with its anagrams.

Press any key to continue . . .
```

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\A

Words that are in the list

1 iron
2 abu
3 oil
4 lap
5 pod
6 cat
7 dog
Press any key to continue . . .
```

5) Appending word to list (option 3 in main menu)

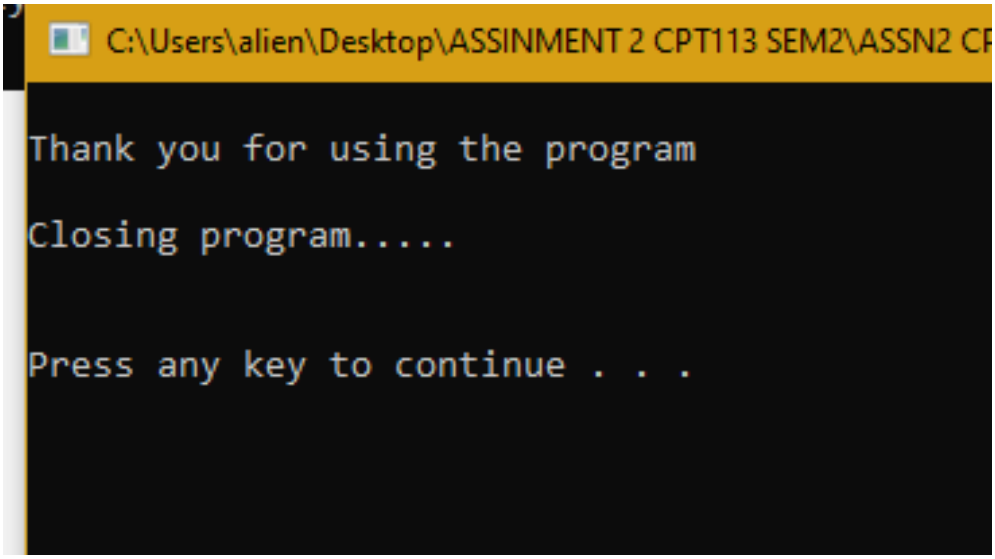
```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT11

!!!MAIN MENU!!!

Enter a number according to the below options.
1.Display all the words in list.
2.Search
3.Append word to list.
4.Display Anagrams of All Words.
5.Exit
Options:3

Appending words to list.......

How many new words are you going to append?
Quantity: 1

Word : kath

1 kath
2 khta
3 khat
4 ktha
5 ktah
6 kaht
7 ahtk
8 ahkt
9 athk
10 atkh
11 akht
12 akth
13 thak
14 thka
15 tahk
16 takh
17 tkha
18 tkah
19 htak
20 htka
21 hatk
22 hakt
23 hkta
24 hkat

Word kath has 24 different anagrams.

Press any key to continue . . .
```

6)Displaying anagram of all words (option 4 in main menu)

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\ASSN2 CPT113 SEM2.exe

!!!MAIN MENU!!!

Enter a number according to the below options.
1.Display all the words in list.
2.Search
3.Append word to list.
4.Display Anagrams of All Words.
5.Exit
Options:4

Displaying all anagrams of all word in list.......
iron
inor
inro
ionr
iorn
irno
rnoi
rnio
roni
roin
rino
rion
onri
onir
orni
orin
oinr
oirn
nori
noir
nroi
nrio
nior
niro
abu
uab
uba
bau
bua
aub
oil
loi
lio
iol
ilo
oli
lap
pla
pal
alp
apl
lpa
pod
dpo
dop
opd
odp
pdo
cat
tca
tac
```

```
C:\Users\alien\Desktop\ASSINMENT 2 CPT113 SEM2\AS
abu
uab
uba
bau
bua
aub
oil
loi
lio
iol
ilo
oli
lap
pla
pal
alp
apl
lpa
pod
dpo
dop
opd
odp
pdo
cat
tca
tac
act
atc
cta
dog
gdo
god
odg
ogd
dgo
kath
khta
khat
ktha
ktah
kaht
ahtk
ahkt
athk
atkh
akht
akth
thak
thka
tahk
takh
tkha
tkah
htak
htka
hatk
hakt
hkta
hkat
Press any key to continue . . .
```

7) Exiting program (option 5 in main menu)



```cpp
#pragma once
#ifndef main_linklist_H
#define main_linklist_H
#include<string>
using namespace std;


template <class T>
class Linklist
{
public:

    struct ListNode
    {
        T value;
        ListNode* next;  //pointer that will point to next node
        ListNode* permute = NULL; //pointer that will point to the anagrams
    };
    ListNode* head; //starting node of link list

public:
    void append(T); //function to append value to the main linklist
    void append2(T, T);//function to append annagrams to each node in the main list
    void display();//function to display all words
    void display2(T);//function to display anagrams of a searched word
    void display3(T, T);//function to create anagrams ofa  searched word starting with a specific letter
    void insert(T, T);//function to insert value into the list
    void deleteNode(T);//function to delete node
    void validatelist(T, int&);//function for validation of words.
    void displayana();//function to display all anagrams of all words
    Linklist() { head = NULL; }//constructor
    ~Linklist();//destructor
};

template<class T>
Linklist<T>::~Linklist() //destructor
{
    ListNode* nodeptr;
    ListNode* nextnode;

    nodeptr = head;

    while (nodeptr != NULL)
    {
        nextnode = nodeptr->next;
        delete nodeptr;
        nodeptr = nextnode;
    }
}

```

```cpp
template <class T>
void Linklist<T>::append(T w)//function to append value to the main linklist
{
    ListNode* newData;
    ListNode* nodePtr;

    newData = new ListNode;
    newData->value = w;
    newData->next = NULL;

    if (!head) //head has no value newnode will become head
    {
        head = newData;
    }
    else
    {
        nodePtr = head;

        while (nodePtr->next)  //looping until end of list
        {
            nodePtr = nodePtr->next;
        }
        nodePtr->next = newData; //adding  new node to the end of list

    }
}

template <class T>
void Linklist<T>::append2(T w, T word)//function to append annagrams to each node in the main list
{
    //word is the words in the main link list
    //w is the anagram which will be appended using permute link list
    ListNode* newData; //node to hold the anagrams
    ListNode* nodePtr; //node used to traverse the main list
    newData = new ListNode; //creating new node
    newData->value = w; //adding value to newnode

    nodePtr = head;
    while (nodePtr != NULL && nodePtr->value != word) //loops until it reaches word
    {
        nodePtr = nodePtr->next;
    }

    if (newData->value != word) //check whether the newdata is same with word to avoid repeatsion
    {
        while (nodePtr->permute)
        {
            nodePtr = nodePtr->permute;
        }
```

```cpp

        nodePtr->permute = newData; //appending anagrams using permute pointer
    }
}

template <class T>
void Linklist<T>::display()//function to display all words
{
    ListNode* nodePtr;
    int num = 1;
    nodePtr = head;

    cout << endl;
    while (nodePtr)
    {
        cout << num << " " << nodePtr->value << endl; //display all the words in the main list.
        nodePtr = nodePtr->next;
        num += 1;
    }
}

template <class T>
void Linklist<T>::display2(T w)//function to display anagrams of a searched word
{
    ListNode* nodePtr;
    ListNode* node;
    int num = 1;

    if (!head)
    {
        cout << "List is empty.\n";
    }
    else
    {
        nodePtr = head;
        while (nodePtr->value != w) //searching for the node that has the same as the word entered by user
        {
            nodePtr = nodePtr->next;
        }
        cout << endl;
        while (nodePtr)  //loops to print all the anagrams of the word searched by user
        {
            cout << num << " " << nodePtr->value << endl;
            nodePtr = nodePtr->permute;
            num += 1;
        }
        cout << "\nWord " << w << " has " << num - 1 << " different anagrams.\n\n";
    }
```

```cpp
}

template <class T>
void Linklist<T>::display3(T w, T a)//function to create anagrams of a searched word starting with a specific letter
{
    ListNode* nodePtr;
    int num = 1, count = 0;
    string word;
    nodePtr = head;

    if (!head)
    {
        cout << "List is empty.\n";
    }
    else
    {
        while (nodePtr->value != w)
        {
            nodePtr = nodePtr->next; //searching for the node with same value as searched word using next pointer
        }
        cout << endl;
        while (nodePtr)  //displaying the node values which are the anagrams using permute pointer
        {
            word = nodePtr->value;
            if (word[0] == a[0])
            {
                cout << num << " " << word << endl;
                num += 1;
            }
            nodePtr = nodePtr->permute;
        }
        cout << "\nWord " << w << " has " << num - 1 << " different anagrams ";
        cout << "starting with " << a << ".\n\n";
    }
}

template<class T>
void Linklist<T>::insert(T w, T prev)//function to insert value into the list
{
    ListNode* nodePtr;
    ListNode* node;
    ListNode* newdata;

    int check = 0;

    newdata = new ListNode;
    newdata->value = w;


```

```
    if (head->value == prev)
    {
        newdata->next = head->next;
        head->next = newdata;
    }
    else
    {
        nodePtr = head;
        node = head;
        while (nodePtr->value != prev)
        {
            nodePtr = nodePtr->next;
            node = nodePtr->next;
        }

        nodePtr->next = newdata;
        newdata->next = node;
    }
}

template<class T>
void Linklist<T>::deleteNode(T w)//function to delete node
{
    ListNode* nodePtr;
    ListNode* previousNode;
    ListNode* temp;
    ListNode* node;

    if (!head)
        return;

    if (head->value == w)
    {
        node = head;
        while (node->value != w)
        {
            node = node->next;
        }

        while (node)//deleteing all the anagrams first
        {
            temp = node->permute;
            if (node->value != w)
            {
                delete node;
            }
            node = temp;
        }

        nodePtr = head;
        head = head->next;
        delete nodePtr;
```

```
    }
    else
    {
        nodePtr = head;
        while (nodePtr->value != w)
        {
            nodePtr = nodePtr->next;
        }

        while (nodePtr)//deleteing all the anagrams first
        {
            temp = nodePtr->permute;
            if (nodePtr->value != w)
            {
                delete nodePtr;
            }
            nodePtr = temp;
        }

        node = head;
        previousNode = head;
        while (node->value != w)//deleting the node with value entered by user
        {
            previousNode = node;
            node = node->next;
        }

        if (node)
        {
            previousNode->next = node->next;
            delete node;
        }
    }
    cout << "Word " << w << " is deleted from the list together with its anagrams.\n\n";
}

template<class T>
void Linklist<T>::validatelist(T w, int& list)//function for validation of words.
{
    //This function is used to validate the words entered by user
    //It is used to detect if a new word added by user is already exist in the list and show error
    //also used when searching for a word , it will show error if the word enterd by user not already existing in the list.
    //because when seacrching a word it must be already existing in the list.
    ListNode* nodeptr;
    nodeptr = head;
    while (nodeptr)
    {
        if (nodeptr->value == w)
        {
```

```
            list = 1;
        }
        nodeptr = nodeptr->next;
    }
}

template<class T>
void Linklist<T>::displayana()
{
    ListNode* nodeptr;
    ListNode* print;

    nodeptr = head;
    while (nodeptr)
    {
        print = nodeptr;
        while (print)
        {
            cout << print->value << endl;
            print = print->permute;
        }
        nodeptr = nodeptr->next;
    }
}
#endif
```

```cpp
#pragma once
#ifndef main_stack_H
#define main_stack_H
#include<string>
using namespace std;

template<class T>
class stack
{
private:
    struct stackNode
    {
        T value; //variable that holds the value of a pointer
        stackNode* next; //pointer that will point to the next value
    };
    stackNode* top; //pointer that is at the top of the stack

public:
    stack() { top = NULL; } //constructor
    ~stack(); //destructor
    void push(T); //function to enter value into the stack
    void pop(T&); //function to remove value from top of the stack
    bool isempty() const; //function to check whether stack is empty
};

template<class T>
stack<T>::~stack()
{
    stackNode* nodeptr;
    stackNode* nextnode;

    nodeptr = top;

    while (nodeptr != NULL)
    {
        nextnode = nodeptr->next;
        delete nodeptr;
        nodeptr = nextnode;
    }
}

template<class T>
void stack<T>::push(T w) //function to enter value into the stack
{
    stackNode* newnode = NULL; //new pointer to hold the value that will be pushed to the stack
    newnode = new stackNode; //creating new node
    newnode->value = w; //giving value to the node
    newnode->next = NULL; //pointing the newnode to null
    if (isempty()) //checking if stack is empty
    {
        top = newnode; //pushing value to the top of the stack
```

```
        newnode->next = NULL; //pointing the newnode to null
    }
    else
    {
        newnode->next = top; //pointing the newnode to top
        top = newnode;
    }

}

template<class T>
void stack<T>::pop(T& w) //function to remove value from top of the stack
{
    stackNode* temp = NULL; //creating a new node

    if (isempty()) //checking if stack is empty
    {
        cout << "The stack is empty\n";
    }
    else
    {
        //removing value from top of the stack
        w = top->value;
        temp = top->next;
        delete top;
        top = temp;
    }
}

template<class T>
bool stack<T>::isempty() const //function to check whether stack is empty
{
    bool status;

    if (!top)
        status = true;
    else
        status = false;

    return status;
}

#endif
```

```cpp
#pragma once
#ifndef main_queue_H
#define main_queue_H
#include<string>
using namespace std;


template<class T>
class queue //class to use queue data structure
{
private:
    struct queueNode
    {
        T value;
        queueNode* next; //pointer that will point to the next value
    };
    queueNode* front; //starting of the queue
    queueNode* rear;//contains values after the front
    int count;

public:
    queue() { front = NULL, rear = NULL, count = 0; }//constructor
    ~queue() { clear(); }//desstructor
    void enqueue(T); // function to put value inside the queue
    void dequeue(T&);//function to remove a value from queue
    bool isempty() const;//function to check a queue is empty or not
    void clear(); //function to clear all the queue values and make it empty
};

template<class T>
void queue<T>::enqueue(T w)// function to put value inside the queue
{
    queueNode* newnode = NULL; //node to hold the new data
    newnode = new queueNode; //creating a node
    newnode->value = w; //giving a value to the newnode node
    newnode->next = NULL;//pointing the next value of newnode to null

    if (isempty()) //checking whther queue is empty
    {
        front = newnode;
        rear = newnode;
    }
    else
    {
        rear->next = newnode;
        rear = newnode;
    }
    count++;
}

template<class T>
void queue<T>::dequeue(T& w)//function to remove a value from queue
```

```
{
    queueNode* nodeptr = NULL;
    if (isempty()) //checking whether queue is empty
    {
        cout << "Queue is empty.\n";
    }
    else
    {
        w = front->value; //value that will be deleted from queue
        nodeptr = front; //declaring nodeptr as front
        front = front->next; //giving value of next pointer of front to front
        delete nodeptr; // removing  value from begining of queue
        count--;
    }
}

template<class T>
bool queue<T>::isempty() const//function to check a queue is empty or not
{
    if (front == NULL) //checking whether front value is null
        return true;
    else
        return false;
}

template<class T>
void queue<T>::clear()//function to clear all the queue values and make it empty
{
    queueNode* nodeptr;
    while (!isempty()) //until the queue becomes empty the begining value in the queue will be deleted
    {
        nodeptr = front;
        front = front->next;
        delete nodeptr;
    }
    count = 0;
}

#endif

```

```cpp
#include<iostream>
#include<fstream>
#include<string>
#include "queue.h"
#include "stack.h"
#include "linklist.h"
using namespace std;

void anagram(string, Linklist<string>&, stack<string>, queue<string>); //function to creating anagram of a word

int main()
{
    Linklist<string> l; //object for linklist class
    queue<string> q;//object for queue class
    stack<string> s;//object for stack class

    fstream dataReadFile; //file declaration
    int option, num, qty, begin = 1;
    char menuchoice;
    string word, appending, filename;

    cout << "Enter file name that contains words : ";
    cin >> filename; //input filename that has all the words

    dataReadFile.open(filename); //opening file

    while (!dataReadFile) //input validation if the filename entered does not exist or cannot open
    {
        //while loop will loop  until the file entered by user can be opened
        cout << "\nError!!! Wrong File Name !!!\n";
        cout << "Enter correct file name that contains words : ";
        cin >> filename;
        dataReadFile.open(filename);
    }

    if (dataReadFile) //if the file is opened
    {
        while (!dataReadFile.eof()) //until the end of line in the file the while loop will loop
        {
            dataReadFile >> appending; //reading word from file
            l.append(appending);//function that adds the word read from file to main link list
            anagram(appending, l, s, q);//function that creates anagram for the word read from the file
        }
    }
    dataReadFile.close(); //closing file

    system("CLS");//clearing commmend prom
```

```cpp

    while (begin == 1)//loop for main menu
    {
        system("CLS");
        cout << "\n!!!MAIN MENU!!!\n";  //main menu
        cout << "\nEnter a number according to the below options.\n";
        cout << "1.Display all the words in list.\n2.Search\n";
        cout << "3.Append word to list.\n4.Display Anagrams of All Words.\n5.Exit \n";
        cout << "Options:";//option that are given to user
        cin >> menuchoice;//each option will go to different case in switch.

        while (menuchoice != '1' && menuchoice != '2' && menuchoice != '3' && menuchoice != '4' && menuchoice != '5')
        {//input validation for menuchoice
            cout << "Input the correct value.\n";
            cout << "Options:";
            cin >> menuchoice;//it will loop until input is correct
        }

        switch (menuchoice)
        {
        case'1': //when user choose option one above
        {
            system("CLS");
            cout << "\n\nWords that are in the list\n";
            l.display();//function to display all the words in the main link list
            system("PAUSE");
            break;//exiting the switch
        }
        case'2':
        {
            string search, nword;
            int check = 0, menu = 1, check2 = 0;
            char mchoice;
            system("CLS");
            cout << "Enter the word that you want to search: ";
            cin >> search;
            l.validatelist(search, check);//function to check if the word enterd by user exist in the list
            while (check != 1)//input validation for word
            {
                //it will loop until the word entered by user already exist in the main link list
                cout << "Error!!!The word you are searching does not exist in the list.\n";
                cout << "Enter correct word: ";
                cin >> search;
                l.validatelist(search, check);
            }
            while (menu == 1)
            {
```

```cpp
                system("CLS");
                cout << "Searching...(WORD : " << search << ")\n";
                cout << "1.Display All Anagrams of Word.\n2.Display Anagrams of word staring with specific letter.\n";
                cout << "3.Update word.\n4.Add word.\n5.Delete word.\n6.Exit \nOption: ";
                cin >> mchoice;
                while (mchoice != '1' && mchoice != '2' && mchoice != '3' && mchoice != '4' && mchoice != '5' && mchoice != '6')
                {//input validation for mchoice
                    cout << "Input the correct value.\n";
                    cout << "Options:";
                    cin >> mchoice;//it will loop until input is correct
                }
                switch (mchoice)
                {
                case'1':
                {
                    cout << "Anagram of word " << search << " are...\n";
                    l.display2(search);//displays all anagrams of the word
                    system("PAUSE");
                    break;
                }
                case'2':
                {
                    int acheck = 1;
                    string alpha;
                    cout << "Enter one of the letters in the searched word: ";
                    cin >> alpha;//one of the searched word letter
                    for (int i = 0; i < search.length(); i++) { if (alpha[0] == search[i]) { acheck = 0; } }//loop to check wheter the alphabet entered is one of the letters in the word.
                    while (acheck != 0)//input validation for alphabets
                    {
                        //will looop until the alphabet entered is one of the alphabets in the word
                        cout << "Error!!!The letter that you entered does not exist in the word you are searching.\n";
                        cout << "Enter the correct alphabets: ";
                        cin >> alpha;
                        for (int i = 0; i < search.length(); i++) { if (alpha[0] == search[i]) { acheck = 0; } }
                    }
                    l.display3(search, alpha);
                    system("PAUSE");
                    break;
                }
                case'3':
                {
                    cout << "\nUpdate word.\n\n";
                    cout << "Replace the word " << search << " with..... \nEnter new word:";
```

```
                    cin >> nword; //word that will replace the searched word
                    l.validatelist(nword, check2);//check whether the word entred is in the link list
                    while (check2 == 1)
                    {
                        //it will loop until the word entered by user does not exist in the main link list
                        //this is to avoid duplicates
                        check2 = 0;
                        cout << "Error!!!The word alraedy exist in the list.\n";
                        cout << "Enter new word: ";
                        cin >> nword;
                        l.validatelist(nword, check2);
                    }
                    l.insert(nword, search);//function to insert word after the word we want to edit
                    l.deleteNode(search);//function to delete the word user want to delete
                    cout << "The word " << search << " has been replaced to " << nword << endl;
                    anagram(nword, l, s, q);//function to generate anagrams
                    cout << "\nIts anagrams are....\n";
                    l.display2(nword);
                    edit = 1;
                    system("PAUSE");
                    search = nword;
                    break;
                }
                case'4':
                {
                    cout << "Adding word after the word " << search << ".\n";
                    cout << "Enter the word you want to add: ";
                    cin >> nword; //word that will be inserted after the searched word
                    l.validatelist(nword, check2);//check whether the word entred is in the link list
                    while (check2 == 1)
                    {
                        //it will loop until the word entered by user does not exist in the main link list
                        //this is to avoid duplicates
                        check2 = 0;
                        cout << "Error!!!The word alraedy exist in the list.\n";
                        cout << "Enter new word: ";
                        cin >> nword;
                        l.validatelist(nword, check2);
                    }
                    l.insert(nword, search);//function to insert a word after the searched word
                    anagram(nword, l, s, q);//creating the inserted words anagrams
                    cout << "\nWord " << nword << " has been added after the word
```

```
                        " << search << ".\n";
                    cout << "\nIts anagrams are....\n";
                    l.display2(nword);
                    system("PAUSE");
                    break;
                }
                case'5':
                {
                    cout << "Deleting word....\n";
                    l.deleteNode(search);//function to delete the word entered by user
                    menu = 0;
                    system("PAUSE");
                    break;
                }
                case'6':
                {
                    menu = 0;
                    break;
                }
                }
            }
            break;
        }
        case'3':
        {//appending word to list option
            int check3 = 0;
            cout << "\nAppending words to list.......\n";
            cout << "\nHow many new words are you going to append?\nQuantity: ";
            cin >> qty; //quantity of word that we want to add
            while (qty < 0)//input validation for qty
            {
                cout << "Input the correct value.\n";
                cout << "Options:";
                cin >> qty;//it will loop until input is correct
            }
            cout << endl;
            for (int i = 0; i < qty; i++)
            {
                cout << "Word : ";
                cin >> appending;//word that we want to append to end of main list
                l.validatelist(appending, check3);//check whether the word entred is in the link list
                while (check3 == 1)
                {
                    //it will loop until the word entered by user does not exist in the main link list
                    //this is to avoid duplicates
                    check3 = 0;
                    cout << "Error!!!The word alraedy exist in the list.\n";
                    cout << "Enter new word: ";
```

```
                    cin >> appending;
                    l.validatelist(appending, check3);
                }
                l.append(appending);//appends new word to the main link list
                anagram(appending, l, s, q);//creates new words anagrams
                l.display2(appending);//displays new words anagrams
                system("PAUSE");
                system("CLS");
            }
            break;
        }
        case'4':
        {
            string all;
            cout << "\nDisplaying all anagrams of all word in list.......\n";
            l.displayana();//function to display all words of  with thier anagrams
            system("PAUSE");
            system("CLS");
            break;
        }
        case'5':
        {
            //option in the main menu to exit the whole program.
            system("CLS");//clearing command prompt
            cout << "\nThank you for using the program\n\n";
            cout << "Closing program.....\n\n\n";
            system("PAUSE");
            begin = 0;
        }
        }
    }
    return 0;
}


void anagram(string word, Linklist<string>& l, stack<string> s, queue<string> q)
{
    //this function is to generate anagrams of a word
    //the method used is swapping method
    //how the function works is
    /*lets say we have word "abcd" this method will swap all characters "a","b","c","d" in each position of the word "abcd"*/
    //when a position in the word is swapped , swapping will happen in the second position while the first position character will be fixed
    //lets say in the first position of the word "abcd" we swap b , then in the position b it will be swapped to a creating anagram "bacd"
    //each time swapping happens the character we are swapping with must not be as same as the characters in the position that we already swap
    //swapping will not occur in the last character of the word
    //so swapping will happen length of the word-1 times (if 4 charcters long means swap will happen 3 timess)
```

```cpp

    //stack and queue is used in the generation of anagram
    //when the word is send to the anagram function it will be enqueued to the queue
    //then when swapping happen in first postion the word will be dequeued and anagram is generated and pushed to stack
    //after that for the second position words will be poped from satck one by one and the created anagram will be enqueue to queue
    //for the third time queue will be used again
    //this will happen until the swapping happens in all the position until the position before last character in the word.
    //anagrams generated at the last swapping are the final complete anagrams of the word.

    char temp;
    string w, wrd, traverse = word;
    w = word; //w is the variable that will hold each character of word, changing depending on the for loop for swapping
    int length, num = 0, counter = 0, check = 0, output = 0;
    length = word.length();
    q.enqueue(word);

    for (int i = 0; i < length - 1; i++)//this for loop is for the position of the word.
    {
        if (s.isempty()) { num = 1; }//to check whether to use stack or queue
        else if (q.isempty()) { num = 2; }

        if (num == 1) //using queue
        {
            while (!q.isempty())//until queue is empty
            {
                output = 1;
                q.dequeue(word);
                for (int x = 0; x < length; x++)//for loop for each characters in a word
                {
                    temp = w[x];
                    wrd = word;
                    for (int z = 0; z < length; z++)//for loop for searching the position of the character that used to swap in the word before swapping
                    {
                        if ((w[x] == wrd[z]) && counter == 0)//for first swapping
                        {
                            wrd[z] = wrd[i];
                            wrd[i] = temp;
                            s.push(wrd);
                        }
                        if ((w[x] == wrd[z]) && counter == 1) //for the rest of the swapping
                        {
```

```
                            for (int u = 0; u < i; u++) // to check whether the characted used for swapping is same as the characters in the position where swapping already occured
                            {
                                if (wrd[u] != w[x])
                                {
                                    check = 1;
                                }
                                else if (wrd[u] == w[x])
                                {
                                    check = 0;
                                    break;
                                }
                            }
                            if (check == 1) //swapping process
                            {
                                wrd[z] = wrd[i];
                                wrd[i] = temp;
                                s.push(wrd);//pushing the generated anagram to stack
                            }
                            check = 0;
                        }
                    }
                }
            }
        }
        else if (num == 2)//using stack
        {
            while (!s.isempty()) //until stack is empty
            {
                output = 2;
                s.pop(word); //popping the word from stack
                for (int x = 0; x < length; x++)//for loop for each characters in a word
                {
                    temp = w[x];
                    wrd = word;
                    for (int z = 0; z < length; z++)//for loop for searching the position of the character that used to swap in the word before swapping
                    {
                        if (w[x] == wrd[z])
                        {
                            for (int u = 0; u < i; u++)// to check whether the characted used for swapping is same as the characters in the position where swapping already occured
                            {
                                if (wrd[u] != w[x])
                                {
                                    check = 1;
                                }
```

```
                                else if (wrd[u] == w[x])
                                {
                                    check = 0;
                                    break;
                                }
                            }
                            if (check == 1)//swapping process
                            {
                                wrd[z] = wrd[i];
                                wrd[i] = temp;
                                q.enqueue(wrd);//enqueuing word to queue
                            }
                            check = 0;
                        }
                    }
                }
            }
        }
        num = 0;
        counter = 1;
    }

    //after all the anagrams has been genereted it checks where is the anagrams in , in the stack or queue
    if (output == 1)
    {
        while (!s.isempty())
        {
            s.pop(wrd);
            l.append2(wrd, traverse);//appending the generated anagrams to the permute pointer for each word in the main link list
        }
    }
    else if (output == 2)
    {
        while (!q.isempty())
        {
            q.dequeue(wrd);
            l.append2(wrd, traverse);//appending the generated anagrams to the permute pointer for each word in the main link list
        }
    }
}
```