Experiment No: 01

Title: Program for performing set operations using array.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 1

Title: Program for performing set operations using array.

Objective: To understand set operation. Representation and implementation of operation of sets using array

Problem Statement:

In Second year Computer Engineering class of M students, set A of students play cricket and set B of students play badminton. Write C/C++ program to find and display -
i.Set of students who play either cricket or badminton or both
ii. Set of students who play both cricket and badminton
iii.Set of students who play only cricket
iv. Set of students who play only badminton
v.Number of students who play neither cricket nor badminton
(Note-While realizing the set duplicate entries are to avoided)

Outcomes:

1. List of students who play cricket or badminton or both
2. List of students who play both cricket and badminton
3. List of students who only play cricket
4. List of students who only play badminton
5. Number of students who play neither games

Software and Hardware Requirement:

1. Linux operating system
2. Eclipse IDE with g++ compiler

Theory:

Set: a set is a collection of objects which are called the members or elements of that set. If we have a set we say that some objects belong (or do not belong) to this set, are (or are not) in the set.

Examples: the set of students in this room; the English alphabet may be viewed as the set of letters of the English language; the set of natural numbers

Operations on sets:

1. **Union :** The union of A and B, written A ∪ B , is the set whose elements are just the elements of A
   or B or of both.

2. **Intersection:** The intersection of A and B , written A ∩ B , is the set whose elements are just the elements of both A and B.
3. **Difference:** Another binary operation on arbitrary sets is the difference "A minus B" , written A – B, which 'subtracts' from A all elements which are in B. [Also called relative complement: the complement of B relative to A.]
4. **Complement:** This operation is creating a set $\bar{A}$, which is the set consisting of everything not in A

Array

An **array** is a collection of data items, all of the same type, accessed using a common name. A one-dimensional **array** is like a list; A two dimensional **array** is like a table; The **C** language places no limits on the number of dimensions in an **array**, though specific implementations may.

Declaring Arrays

- Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.
- Examples:
  int i, j, intArray[10]

Initializing Arrays

- Arrays may be initialized when they are declared, just as any other variables.
- Place the initialization data in curly {} braces following the equals sign. Note the use of commas in the examples below.
- An array may be partially initialized, by providing fewer data items than the size of the array. The remaining array elements will be automatically initialized to zero.
- If an array is to be completely initialized, the dimension of the array is not required. The compiler will automatically size the array to fit the initialized data.

  int intArray[6] = { 1, 2, 3, 4, 5, 6 };

Using Arrays

- Elements of an array are accessed by specifying the index (offset) of the desired element within square [ ] brackets after the array name.

- Array subscripts must be of integer type. (int, long int, char, etc.)
- **VERY IMPORTANT:** Array indices start at zero in C, and go to one less than the size of the array. For example, a five element array will have indices zero through four. This is because the index in C is actually an offset from the beginning of the array. (The first element is at the beginning of the array, and hence has zero offset.)

Algorithm:

[I] Union(char a[ ][ ], char b[ ][ ], char c[ ][ ],int term1,int term2)

Precondition: Accept the two sets of name.
Post condition: Union of two sets
Return: integer n, i.e. total number of elements in set c

1. k=0
2.flag=0
3.for i=0 to term1, i++
  a. copy name present in a[i] to c[k]
  b. increment k
4.for j=0 to term2, j++
  a. flag=0
  b. for i=0 to term1, i++
    i. if (strcmp(a[i],b[j])==0) then
      set flag to 1
  c. if (flag==0) then
      i.copy name present in b[j] to c[k]
      ii.increment k
5.n=k
6.for k=0 to n, k++
  a. Display names present in set c
7.return n

[II] Intersection(char a[ ][ ], char b[ ][ ], int term1,int term2)

Precondition: Accept the two sets of name.
Post condition: Intersection of two sets
Return: Nil

1.k=0
2.flag=0
3.for i=0 to term1, i++
  a. flag=1
  b. for j=0 to term2, j++
    i. if (strcmp(a[i],b[j])==0) then
      set flag to 0
  c. if (flag==0) then

   i. copy name present in a[i] to c[k]
   ii. increment k
4.n=k
5.for k=0 to n, k++
  b. Display names present in set c
6.return n

[III] difference(char a[ ][ ], char b[ ][ ], int term1,int term2)

Precondition: Accept the two sets of name.
Post condition: Difference of two sets
Return: Nil
1.k=0
2.flag=0
3.for i=0 to term1, i++
  d. flag=1
  e. for j=0 to term2, j++
    i. if (strcmp(a[i],b[j])==0) then
      set flag to 0 and break
  f. if (flag==1) then
    i. copy name present in a[i] to c[k]
    ii. increment k
4.n=k
5.for k=0 to n, k++
  c. Display names present in set c
6.return n

Conclusion:

Thus I have studied concept of set and its representation using array. I have also implemented all the operations of set.

.

Experiment No: 02

Title: Program for calculating average score, maximum and minimum marks, most scored marks using array of structure.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 2

Title: Program for calculating average score, maximum and minimum marks, most scored marks using array of structure.

Objective: To understand the use of structure and its implementation using array.

Problem Statement:

Write C/C++ program to store marks scored for first test of subject 'Data Structures and Algorithms' for N students. Compute-
i. The average score of class
ii. Highest score and lowest score of class
iii. Marks scored by most of the students
iv. list of students who were absent for the test

Outcomes:

6. The average score of whole class
7. Maximum marks scored
8. Minimum marks scored
9. Marks scored by most of the students
10. Number of students who were absent for test.

Software and Hardware Requirement:

3. Linux operating system
4. Eclipse IDE with g++ compiler

Theory:

Array
An **array** is a collection of data items, all of the same type, accessed using a common name. A one-dimensional **array** is like a list; A two dimensional **array** is like a table; The **C** language places no limits on the number of dimensions in an **array**, though specific implementations may.

Declaring Arrays

- Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.
- Examples:
  inti, j, intArray[10]

Initializing Arrays

- Arrays may be initialized when they are declared, just as any other variables.
- Place the initialization data in curly {} braces following the equals sign. Note the use of commas in the examples below.
- An array may be partially initialized, by providing fewer data items than the size of the array. The remaining array elements will be automatically initialized to zero.
- If an array is to be completely initialized, the dimension of the array is not required. The compiler will automatically size the array to fit the initialized data.

  intintArray[6] = { 1, 2, 3, 4, 5, 6 };

Using Arrays

- Elements of an array are accessed by specifying the index (offset) of the desired element within square [ ] brackets after the array name.
- Array subscripts must be of integer type. (int, long int, char, etc.)
- **VERY IMPORTANT:** Array indices start at zero in C, and go to one less than the size of the array. For example, a five element array will have indices zero through four. This is because the index in C is actually an offset from the beginning of the array. (The first element is at the beginning of the array, and hence has zero offset.)

Structure

The ordinary variables can hold one piece ofinformation and arrays can hold a number of pieces ofinformation of the same data type. These two data types canhandle a great variety of situations. But quite often we deal withentities that are collection of dissimilar data types.
For example, suppose you want to store data about a book. Youmight want to store its name (a string), its price (a float) andnumber of pages in it (an int). If data about say 3 such books is to be stored, then we can follow two approaches:

(a) Construct individual arrays, one for storing names, another forstoring prices and still another for storing number of pages.
(b) Use a structure variable.

A structure contains a number of data types grouped together.These data types may or may not be of the same type.

Declaring a Structure

The following statement declares thestructure type:

```
struct book
{
char name ;
float price ;
int pages ;
```

```
} ;
```

This statement defines a new data type called **struct book**.
Once the new structure data type has been defined one or morevariables can be declared to be of that type. For example thevariables **b1**, **b2**, **b3** can be declared to be of the type **struct book**,as,

```
struct book b1, b2, b3 ;
```

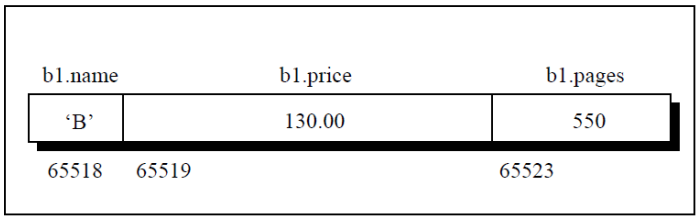
This statement sets aside space in memory.

Accessing Structure Elements

They use a dot (.)operator. So to refer to **pages** of the structure defined in above example following line is used,
b1.pages

How Structure Elements are Stored

Whatever be the elements of a structure, they are always stored incontiguous memory locations. Actually the structure elements are stored in memory as shown inthe figure



Array of Structures

In order to store data of 100 books we would berequired to use 100 different structure variables from **b1** to **b100**,which is definitely not very convenient. A better approach wouldbe to use an array of structures. Following program shows how to use an array of structures.

/* Usage of an array of structures */

```
main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b[100] ;
inti ;
for ( i = 0 ; i<= 99 ; i++ )
{
printf ( "\nEnter name, price and pages " ) ;
scanf ( "%c %f %d", &b[i].name, &b[i].price, &b[i].pages ) ;
}
```

```
for ( i = 0 ; i<= 99 ; i++ )
printf ( "\n%c %f %d", b[i].name, b[i].price, b[i].pages ) ;
}
```

Algorithm:

[I] average(struct student s[20],int n)
Precondition: Accept the test record of all students.
Post condition: Average of marks
Return: Nil
1. k=0
2.sum=0
3.for i=0 to n, i++
  c. if i^{th} student is not absent (i.e. s[i].attendance!="AB") then do
    i. sum=sum+s[i].marks
    ii. increment k
4.avg=sum/k
5.Display average, avg
6.Stop

[II]high_low(struct student s[20],int n)

Precondition: Accept the test record of all students.
Post condition: Maximum and minimum marks scored in class
Return: Nil
7.min=999
8.max=-1
9.for i=0 to n, i++
  g. if i^{th} student is not absent (i.e. s[i].attendance!="AB") then do
    i. if (s[i]>max) then do
    max=s[i].marks
    ii. if (s[i]<min) then do
    min = s[i].marks
10. Display highest score, max
11. Display lowest score, min
12. stop

[III]most_scored(struct student s[20],intn,intmax_marks)

Precondition: Accept the test record of all students and maximum marks.
Post condition: Marks scored by most of the students
Return: Nil
7.max = -1
8.declare an array count[max_marks+1] and initialize all elements to 0.

9.for i=0 to n, i++
  a. count[s[i].marks]++
10. for i=1 to max_marks, i++
  a. if (count[i]>max) then
    i. max = count[i]
    ii. k = i
11. Display marks scored by most of the students, max.
12. Stop

[IV] absent(struct student s[20],int n)
Precondition: Accept the test record of all students.
Post condition: Display list of students who were absent for test
Return: Nil
1.for i=0 to n, i++
  a. if student is absent (i.e. s[i].attendance = ”AB”) then do
    print name of that student.
2.Stop

Conclusion:

Thus I have studied concept of structure and array of structure. Also implemented a program for performing various operation on students’ test record using array of structure.

.

Experiment No: 03

Title: Program for storing the library record and performing various operations

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 3

Title: Program for storing the library record and performing various operations.

Objective: To understand the use of structure and its implementation using array.

Problem Statement:

Department library has N books. Write C/C++ program to store the cost of books in array in ascending order. Books are to be arranged in descending order of their cost. Write function for
a)Reverse the contents of array without using temporary array.
b)Copy costs of books those with cost less than 500 in new array
c)Delete the duplicate entries using temporary array
d)Delete duplicate entries without using temporary array
e)Count number of books with cost more than 500.

Outcomes:

11. The library data displayed in reverse order
12. List of books with cost less than 500
13. List of books without duplicates
14. Number of books with cost more than 500.

Software and Hardware Requirement:

5. Linux operating system
6. Eclipse IDE with g++ compiler

Theory:

Array
An **array** is a collection of data items, all of the same type, accessed using a common name. A one-dimensional **array** is like a list; A two dimensional **array** is like a table; The **C** language places no limits on the number of dimensions in an **array**, though specific implementations may.

Declaring Arrays

- Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.
- Examples:
  inti, j, intArray[10]

Initializing Arrays

- Arrays may be initialized when they are declared, just as any other variables.
- Place the initialization data in curly {} braces following the equals sign. Note the use of commas in the examples below.
- An array may be partially initialized, by providing fewer data items than the size of the array. The remaining array elements will be automatically initialized to zero.
- If an array is to be completely initialized, the dimension of the array is not required. The compiler will automatically size the array to fit the initialized data.

  intintArray[6] = { 1, 2, 3, 4, 5, 6 };

Using Arrays

- Elements of an array are accessed by specifying the index (offset) of the desired element within square [ ] brackets after the array name.
- Array subscripts must be of integer type. (int, long int, char, etc.)
- **VERY IMPORTANT:** Array indices start at zero in C, and go to one less than the size of the array. For example, a five element array will have indices zero through four. This is because the index in C is actually an offset from the beginning of the array. (The first element is at the beginning of the array, and hence has zero offset.)

Structure

The ordinary variables can hold one piece of information and arrays can hold a number of pieces of information of the same data type. These two data types can handle a great variety of situations. But quite often we deal withentities that are collection of dissimilar data types.
For example, suppose you want to store data about a book. You might want to store its name (a string), its price (a float) and number of pages in it (an int). If data about say 3 such books is to be stored, then we can follow two approaches:

(c) Construct individual arrays, one for storing names, another for storing prices and still another for storing number of pages.
(d) Use a structure variable.

A structure contains a number of data types grouped together. These data types may or may not be of the same type.

Declaring a Structure

The following statement declares the structure type:

```
struct book
{
char name ;
float price ;
int pages ;
```

```
} ;
```

This statement defines a new data type called **struct book**.
Once the new structure data type has been defined one or morevariables can be declared to be of that type. For example thevariables **b1**, **b2**, **b3** can be declared to be of the type **struct book**,as,

```
struct book b1, b2, b3 ;
```

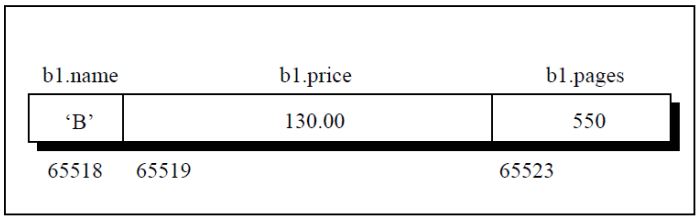
This statement sets aside space in memory.

Accessing Structure Elements

They use a dot (.)operator. So to refer to **pages** of the structure defined in above example following line is used,
b1.pages

How Structure Elements are Stored

Whatever be the elements of a structure, they are always stored incontiguous memory locations. Actually the structure elements are stored in memory as shown inthe figure



Array of Structures

In order to store data of 100 books we would berequired to use 100 different structure variables from **b1** to **b100**,which is definitely not very convenient. A better approach wouldbe to use an array of structures. Following program shows how to use an array of structures.

/* Usage of an array of structures */

```
main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b[100] ;
inti ;
for ( i = 0 ; i<= 99 ; i++ )
{
printf ( "\nEnter name, price and pages " ) ;
scanf ( "%c %f %d", &b[i].name, &b[i].price, &b[i].pages ) ;
}
```

```
for ( i = 0 ; i<= 99 ; i++ )
printf ( "\n%c %f %d", b[i].name, b[i].price, b[i].pages ) ;
}
```

Algorithm:

[I] sort(struct book b[20],int n)
Precondition: Accept the books record.
Post condition: Books in ascending order of costs
Return: Nil
1.for i=0 to n, i++
d. for j = 0 to n-i-1, j++
iii. if (b[j].cost > b[j+1].cost) then do
temp = b[j]. cost
b[j].cost = b[j+1].cost
b[j+1].cost = temp
strcpy(S,b[j].name)
strcpy(b[j].name,b[j+1].name)
strcpy(b[j+1].name,S)
2.call display(b,n) in order to display list of books along with cost in ascending order
3.Stop

[II] reverse(struct book b[20],int n)
Precondition: Accept the books record
Post condition: Books in reverse order.
Return: Nil
13. for i=0& j =n-1, increment i to n/2, i++ & j--
a. temp = b[j]. cost
b. b[j].cost = b[i].cost
c. b[i].cost = temp
d. strcpy(S, b[j].name)
e. strcpy(b[j].name, b[i].name)
f. strcpy(b[i].name, S)
14. call display(b,n) in order to display list of books along with cost in ascending order
15. stop

[III]copy(struct book b[20],int n)
Precondition: Accept the books record
Post condition: List of books with cost less than 500
Return: Nil
13. j=0
14. for i=0 to n, i++
b. if (b[i].cost < 500) then do
i. temp[j].cost = b[i].cost

ii. strcpy(temp[j].name,b[i].name)
iii. increment j by 1

15. call display(temp,j) to display list of books with cost less than 500
16. Stop

[IV] count(struct book b[20],int n)
Precondition: Accept the books record
Post condition: List of books with cost less than 500
Return: Nil
1.count=0
2.for i=0 to n, i++
c. if (b[i].cost > 500) then do
i. increment count by 1
3. Display number of books with cost more than 500, count
4.Stop

[V]dup1(struct book b[20],int n)
Precondition: Accept the books record
Post condition: List of books with no duplicate entry
Return: Nil
1.k=1
2.temp[0].cost=b[0].cost;
3.strcpy(temp[0].name,b[0].name);
4.i=1;
5.while(i<n)
- j=0
- while(j<k)
  - **i.** if(strcmp(b[i].name,temp[j].name)==0)
    - increment i by 1
    - j=0
  - **ii.** else
    - j++

d. if(j==k)
i. temp[k].cost=b[i].cost
ii. strcpy(temp[k].name,b[i].name)
iii. k++
iv. i++

6.call display(temp,k) functionto display all books record with no duplicate entry
7.Stop

[VI]dup2(struct book b1[20],int n1)
Precondition: Accept the books record
Post condition: List of books with no duplicate entry

Return: Nil
1.n=n1
2.fori= 0 to n , i++
1. b[i].cost=b1[i].cost
2. strcpy(b[i].name,b1[i].name)

3.fori=0 to n , i++
a. for j=i+1 to n , j++
i. if(strcmp(b[i].name,b[j].name)==0) then do
I. for k=j to n-1 , k++
b[k].cost=b[k+1].cost;
strcpy(b[k].name,b[k+1].name);
II. n--;

4.call dsiplay(b,n) function to display all books record with no duplicate entry
5.Stop

Conclusion:

Thus I have studied concept of structure and array of structure. Also implemented a program for various operations on library data.

Experiment No: 04

Title: Program for performing all linked list operations.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 4

Title: Program for performing all linked list operations.

Objective: To understand and implement all linked list operations.

Problem Statement:

Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of Second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member‘s information using singly linked list. Store student PRN and Name. Write functions to
a)Add and delete the members as well as president or even secretary.
b)Compute total number of members of club
c)Display members
d)Display list in reverse order using recursion
e)Two linked lists exists for two divisions. Concatenate two lists.

Outcomes:

a)Add and delete the members as well as president or even secretary.
b)Compute total number of members of club
c)Display members
d)Display list in reverse order using recursion
e)Two linked lists exists for two divisions. Concatenate two lists.

Software and Hardware Requirement:

1. Linux operating system
2. Eclipse IDE with g++ compiler

Theory:

Algorithm:

[I] Algorithmcreate()
Precondition: Empty linked list
Post condition: created linked list
Return:head node of the resultant linked list
1. Set flag=TRUE
2. do
a. Read PRN number and name of student, val, n
b. Allocate memory for Newnode

c. if(New==NULL) then display message "Memory not allocated"
d. else
i. Set New->PRN=val, New->name to n and New->next=NULL
e. if(flag==TRUE) then
i. Set head=New, temp=head and flag=FALSE
f. else
i. Set temp->next=New and temp=New
g.Read ans
h. while(ans=='y'||ans=='Y') go to step 2
3. return head

[II] Algorithm display(node *head)
Precondition: head
Post condition: linked list
Return: Nil
1. Set temp=head
2. if(temp==NULL) then Display message "List is empty" and go to step 4.
3. while(temp!=NULL)
a. Display temp->PRN and temp->name
b. temp=temp->next
4. Stop

[III] Algorithm count()
Precondition: linked list
Post condition: count of number of members
Return: Nil
1. count=0 and temp=head
2. if(temp==NULL)then Display message "List is empty" and go to step 5.
3. while(temp!=NULL)
a. Increment count
b. temp=temp->next
4. Display count
5. Stop

[IV] Algorithm reverse(node *head)
Precondition: head linked list
Post condition: reverse of linked list
Return: Nil
1. if(head!=NULL) then call reverse(head->next)
2. else go to step 4
3. Display head->PRN and head->name
4. Stop

[V] Algorithm remove()
1. Set temp=head
2. Read key
3. while(temp!=NULL)
    a. if(temp->PRN==key) then go to step 4
    b. Set prev=temp and temp=temp->next
4. if(temp==NULL) then display message "Node not found"
5. else
    a. if(temp==head)
        head=temp->next
    b. else
        prev->next=temp->next
    c. delete temp;
6. return head

[VI] Algorithm insert_Secretary()
Precondition: linked list
Post condition: linked list with newly inserted value
Return: Nil
1. Allocate memory for New node
2. Read PRN number and name of student
3.if(head==NULL) then
    head=New
4.else
    a.temp=head;
    b.while(temp->next!=NULL)
        temp=temp->next;
    c.temp->next=New;
    d.New->next=NULL;
5. Stop

[VII] Algorithm insert_member()
Precondition: linked list
Post condition: linked list with newly inserted value
Return: Nil
1. Allocate memory for New node
2. Read PRN number and name of student
3.if(head==NULL) then
    head=New
4.else
    a. Read key
    b. temp=head
    c. do

i. if(temp->PRN==key)
I. New->next=temp->next
II. temp->next=New
III. go to step 5
ii. else
temp=temp->next
d. while(temp!=NULL) go to step c
5. Stop

[VII] Algorithm insert_president()
Precondition: linked list
Post condition: linked list with newly inserted value
Return:node head
1. Allocate memory for New node
2. Read PRN number and name of student
3.if(head==NULL) then
head=New
4.else
a.temp=head;
b.New->next=temp;
c.head=New;
5.return head
}

[ ]Algorithmconcat(node *head1,node *head2)
Precondition: head of two linked list
Post condition: head of concatenated linked list
Return:node pointer
1. Set temp=head1
2. while(temp->next!=NULL)
temp=temp->next
3. temp->next=head2
4. return head1;

Conclusion:

Thus I have studied and implemented all the operations of linked list.

Experiment No: 05

Title: Program for performing set operations using linked list.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 5

Title: Program for performing set operations using linked list.

Objective: To understand set operation. Representation and implementation of operation of sets using linked list

Problem Statement:

Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C/C++ program to store two sets using linked list. compute and display

i.Set of students who like either vanilla or butterscotch or both
ii.Set of students who like both vanilla and butterscotch
iii.Set of students who like only vanilla not butterscotch
iv.Set of students who like only butterscotch not vanilla
v. Number of students who like neither vanilla nor butterscotch

Outcomes:

15. List of students who like either vanilla or butterscotch or both
16. List of students who like both vanilla and butterscotch
17. List of students who like only vanilla not butterscotch
18. List of students who like only butterscotch not vanilla
19. Number of students who like neither vanilla nor butterscotch

Software and Hardware Requirement:

7. Linux operating system
8. Eclipse IDE with g++ compiler

Theory:

Set:a setis a collection of objects which are called the membersor elementsof that set. If we have a set we say that some objects belong(or do not belong) to this set, are(or are not) inthe set.

Examples: the set of students in this room; the English alphabet may be viewed as the set of letters of the English language; the set of natural numbers

Operations on sets:

5. **Union :**The union of A and B, written A ∪ B , is the set whose elements are just the elements of A
   or B or of both.

6. **Intersection:** The intersection of A and B , written A ∩ B , is the set whose elements are just the elements of both A and B.
7. **Difference:**Another binary operation on arbitrary sets is the difference "A minus B" , written A – B, which 'subtracts' from A all elements which are in B. [Also called relative complement: the complement of B relative to A.]
8. **Complement:**This operation is creating a set $\bar{A}$, which is the set consisting of everything not in A

Linked List

Algorithm:

[I] Algorithmdifference(struct node *head1, struct node *head2)
Precondition: Accept the two sets of name in the form of linked list.
Post condition: Difference of two sets
Return:head node of the resultant linked list(Difference)
1.Set p=head1, i.e. head of 1st linked list
2. Initialize the 3rd linked list as empty i.e. head3=NULL
3. while(p!=NULL)
- **a.** Set flag=0
- **b.** Set q=head2, i.e. head of 2ndlinked list
- **c.** while(q!=NULL)
  - **i.** if(strcmp(p->name,q->name)==0)
    - **I.** Set flag to 1 and go to step d
  - **ii.** else
    - **I.**Move q to the next node, q=q->next
- **d.** if(flag!=1)
  - **i.** if(head3==NULL)
    - **I.** Allocate memory for New node
    - **II.** strcpy(New->name,p->name)
    - **III.** Set r=New and r->next=NULL
    - **IV.** Make New node as head node of resultant linked list
  - **ii.**else
    - **I.** Allocate memory for New node
    - **II.** r->next=New
    - **III.** strcpy(New->name,p->name)
    - **IV.** Set r=r->next and r->next=NULL
- **e.** Move p to the next node, p=p->next

4. return head3 i.e. head of resultant linked list

5. Stop

[II] Algorithm Intersection(struct node *head1, struct node *head2)
Precondition: Accept the two sets of name in the form of linked list.
Post condition: Intersection of two sets
Return: Nil
1.Set p=head1, i.e. head of 1st linked list
2. Initialize the 3rd linked list as empty i.e. head3=NULL
3. while(p!=NULL)
- **a.** Set q=head2, i.e. head of 2nd linked list
- **b.** while(q!=NULL)
  - **i.**if(strcmp(p->name,q->name)==0)
    - **I.** if(head3==NULL)
      - **1.**Allocate memory for New node
      - **2.** strcpy(New->name,p->name)
      - **3.** Set r=New and r->next=NULL;
      - **4.** Make new node as head of linked list
    - **II.** else
      - **1.**Allocate memory for New node
      - **2.** r->next=New
      - **3.** strcpy(New->name,p->name)
      - **4.** Set r=r->next and r->next=NULL
    - **III.** Go to step c
  - **ii.** Move q to the next node, q=q->next
- **c.** Move p to the next node, p=p->next

4. Call display(head3) function to display resultant linked list containing intersection
5. Stop

[III] Algorithm Union(struct node *head1,struct node *head2)
Precondition: Accept the two sets of name in the form of linked list.
Post condition: Union of two sets
Return: head node of the resultant linked list(Union)
1. Set q=head2, i.e. head of 2nd linked list
2. Copy all contents of first linked list in third, head3=head1
3. Set r=head3, i.e. head of 3rd linked list
4. while(r->next!=NULL)
- Move r to the next node, r=r->next

5. while(q!=NULL)
- **a.** Set p=head1, i.e. head of 1st linked list
- **b.** Initialize flag to 0
- **c.** while(p!=NULL)
  - **i.** if(strcmp(p->name,q->name)==0)
    - Set flag to 1 and go to step d

ii. else

Move p to the next node, p=p->next

d. if(flag==0)

i. Allocate memory for New node

ii. r->next=New

iii. strcpy(New->name,q->name);

iv. Set r=r->next and r->next=NULL

e. Move q to the next node, q=q->next

6. return head3, i.e. head of resultant linked list containing union

7. Stop

Conclusion:

Thus I have studied concept of set and its representation using array. I have also implemented all the operations of set.

.

Experiment No: 06

Title: Program for performing operations on linked list.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 6

Title: Program for performing operations onlinked list.

Objective: To understand various operations and their implementation for linked list

Problem Statement:

Write C++ program to store set of negative and positive numbers using linked list. Write functions
a) Insert numbers
b) Delete nodes with negative numbers
c) To create two more linked lists using this list, one containing all positive numbers and other containing negative numbers
d) For two lists that are sorted; Merge these two lists into third resultant list that is sorted**Outcomes:**

1. Insert numbers
2. Delete nodes with negative numbers
3. To create two more linked lists using this list, one containing all positive numbers and other containing negative numbers
4. For two lists that are sorted; Merge these two lists into third resultant list that is sorted

Software and Hardware Requirement:

1. Linux operating system
2. Eclipse IDE with g++ compiler

Theory:

Algorithm:

[I] Algorithmcreate()
Precondition: Empty linked list.
Post condition: Create linked
Return:head node of the resultant linked list
1.Set temp = NULL and ans='y'
2.Set flag=TRUE
3.do
a. Read val
b. Allocate memory for New node
c.if(New==NULL) then

i. Display message " Memory not allocated"
ii. Set New->num=val and New->next=NULL
iii.if(flag==TRUE) then
I.head=New
II.temp=head.
III.flag=FALSE
iv. else
I.temp->next=New
II.temp=New
d.Read ans
4.while(ans=='y'||ans=='Y')
5.return head;
6. Stop

[II] Algorithm display(node *head)
Precondition: Head of linked list.
Post condition: Nil
Return: Nil
1.Set p=head
2.if(p==NULL) then
a. Display message "List is empty" and go to step 4
3.while(p!=NULL)
a. Display p->num
b. Set p=p->next
4.Stop

[II] Algorithminsert_last()
Precondition: Nil.
Post condition: Linked list with new inserted node
Return: Nil
1. Allocate memory for New node
2. Read New->num
3.if(head==NULL) then
a.head=New
4.else
a.temp=head
b.while(temp->next!=NULL) do
temp=temp->next
c.temp->next=New
d.New->next=NULL
5. Stop

[II] Algorithminsert_after()
Precondition: Nil
Post condition: Linked list with new inserted node
Return: Nil
1. Allocate memory for New node
2. Read New->num
3.if(head==NULL) then
    a.head=New
4.else
a. Read node after which node to be inserted, key
    b. temp=head
    c. do
        i. if(temp->num==key) then
            I. New->next=temp->next;
            II. temp->next=New;
            III. Go to step 5
        ii. else
            I. temp=temp->next;
    e. while(temp!=NULL)
5. Stop

[III] Algorithminsert_head()
Precondition: Nil
Post condition: Linked list with new inserted node
Return: Nil

1. Allocate memory for New node
2. Read New->num
3.if(head==NULL) then
    a.head=New
4.else
    a. temp=head
    b. New->next=temp
    c. head=New
5. return head

[IV] Algorithm Delete()
Precondition: Nil.
Post condition: Linked list with only positive numbers
Return:head node of linked list
1. Allocate memory for prev node
2. Set temp1=head

3. while(temp1!=NULL)
- **a.** if(temp1->num<0)
  - **i.** temp=head;
  - **ii.** while(temp!=NULL)
    - **I.** if(temp->num<0)
      - Go to step iii.
    - **II.** prev=temp
    - **III.** temp=temp->next
    - **IV.** if(temp==NULL) then
      - Display message " Node not found"
    - **V.** else
      - if(temp==head) then
        - Set head=temp->next and temp1=head
      - else
        - Set prev->next=temp->next and temp1=prev->next
    - **VI.** delete temp
- **b.** else
  - temp1=temp1->next

4. return head;

[V] Algorithm Create_sorted()
Precondition: Nil.
Post condition: Linked list with sorted elements
Return: head node of linked list
1. Set temp=NULL and head=NULL
2. do
- **a.** Allocate memory for New node
- **b.** Read New->num
- **c.** if(head==NULL) then
  - **i.** head=New
- **d.** else
  - **i.** Set temp=head and prev=NULL
  - **ii.** while(temp!=NULL)
    - **I.** if(temp->num<New->num)
      - Set prev=temp and temp=temp->next
    - **II.** else go to step iii.
  - **iii.** if(prev==NULL) then
    - **I.** New->next=head
    - **II.** head=New
  - **iv.** else
    - **I.** New->next=prev->next
    - **II.** prev->next=New
- **e.** Read ch

f. while(ch=='y'||ch=='Y') go to step d

3. return head

[VI] Algorithm merge(struct node *first, struct node* second)

Precondition: two sorted linked list

Post condition: Linked list with sorted elements

Return: head node of merged linked list

1. Set third=NULL

2. if(first==NULL) then return(second)

3. else if(second==NULL) then return(first)

4. if(first->num<=second->num) then

a. third=first

b. third->next=merge(first->next,second)

5. else

a. third=second

b. third->next=merge(first,second->next)

6. return(third)

Conclusion:

Thus I have studied concept of linked list. I have also implemented some the operations.

.

Experiment No: 07

Title: Program to check for palindrome and display reverse string using stack.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 7

Title: Program to check for palindrome and display reverse string using stack.

Objective: To understand theuse of stack to reverse the string and check for the palindrome.

Problem Statement:

A palindrome is a string of character that's the same forward and backward. Typically, punctuation, capitalization, and spaces are ignored. For example, ‖Poor Dan is in a droop‖ is a palindrome, as can be seen by examining the characters ―poor danisina droop‖ and observing that they are the same forward and backward. One way to check for a palindrome is to reverse the characters in the string and then compare with them the original-in a palindrome, the sequence will be identical. Write C++ program with functions-
1. to check whether given string is palindrome or not that uses a stack to determine whether a string is a palindrome.
2. to remove spaces and punctuation in string, convert all the Characters to lowercase, and then call above Palindrome checking function to check for a palindrome
3. to print string in reverse order using stack

Outcomes:

1. Message displaying whether the given string is palindrome or not
2. String Reversed

Software Requirement:

9. Linux operating system
10. Eclipse IDE with g++ compiler

Theory:

Stack:

Stack is a LIFO (Last In First Out) data structure. It is an ordered list of the same type of elements. It is a linear list in which all insertions and deletions are performed at the same end called **top.** When elements added to the stack it grows at one end, and when elements are deleted from stack it shrinks at the same end.

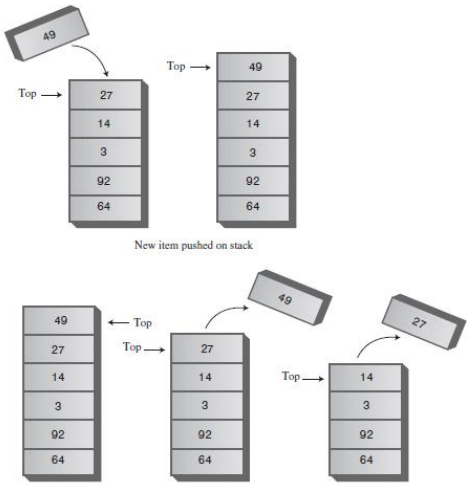


Figure: Insertion and deletion operation on stack

So, A*stack* is an ordered list in which all insertions and deletions are made at one end, called the *top*.
Associated with the stack there are several operations:

INITIALIZE (*S*): which creates *S* as an empty stack;
ADD (*i*,*S*): which inserts the element *i*onto the stack *S* and returns the new stack;
DELETE (*S*): which removes the top element of stack *S* and returns the new stack;
TOP (*S*): which returns the top element of stack *S*;
ISEMPTY (*S*): which returns true if *S* is empty else false;
ISFULL (S): which returns true is S is full else false;
These six functions constitute a working definition of a stack.
The simplest way to represent a stack is by using a one-dimensional array, say STACK(1:*n*), where *n* isthe maximum number of allowable entries. The first or bottom element in the stack will be stored atSTACK(1), the second at STACK(2) and the i-th at STACK(*i*). Associated with the array will be avariable, **top**, which points to the top element in the stack.

Declaration of stack as a structure:

```
structStack
{
        char stack[max];
        int top;
}st;
```

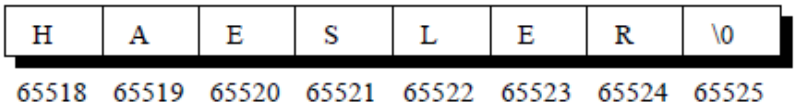
String

The way a group of integers can be stored in an integer array, similarly a group of characters can be stored in a character array. Character arrays are many a time also called strings. Many languages

internally treat strings as character arrays, but somehow conceal this fact from the programmer. Character arrays or strings are used by programming languages to manipulate text such as words and sentences. A string constant is a one-dimensional array of characters terminated by a null ('\0'). For example,

char name[ ] = { 'H', 'A', 'E', 'S', 'L', 'E', 'R', '\0' } ;

Each character in the array occupies one byte of memory and the last character is always '\0'.
The terminating null ('\0') is important, because it is the only waythe functions that work with a string can know where the stringends. In fact, a string not terminated by a '\0' is not really a string,but merely a collection of characters.



The string represented above can be initialized as follows,

char name[ ] = "HAESLER" ;

Note that, in this declaration '\0' is not necessary. C inserts thenull character automatically.

Algorithm:

[I] Algorithm remove_punc(char str[])

Precondition : Accept the string

Postcondition :A string with no spaces, punctuation and all uppercase character concerted to lowercase

**Return : ** char pointer

1.char *p=dst

2.i=0 and j=0

3. while(str[i]!='\0')

- **a.** if(ispunct((unsigned char)str[i])||str[i]==' ')
  - Increment i by 1
- **b.** else if(isupper((unsigned char)str[i]))
  - **i.** dst[j]=tolower((unsigned char)str[i])
  - **ii.** Increment i and j by 1
- **c.** else
  - **i.** dst[j]=str[i]
  - **ii.** Increment i and j by 1

4.dst[j] = '\0'

5. Return p

6. Stop

[II] Algorithm reverse(char str[])

Precondition : Accept the string
Postcondition : Accepted string is reversed
Return : Nil
1.i=0
2. while(str[i]!='\0')
    a. push(str[i])
    b. Increment i by 1
3. Display message **" Reverse string is: "**
4. while(top!=-1)
    a. Display character at the top of the stack by call **pop( )** function
5. Stop

[III] Algorithm palindrome(char str[])
Precondition : Accept the string
Postcondition :A message displaying whether the string is palindrome or not
Return : Nil
1.flag=0
2. i=0
3. while(str[i]!='\0')
    a. push(str[i])
    b. Increment i by 1
4. i=0
5. while(top!=-1)
    a. temp=pop( )
    b. if(str[i]!=temp)
        i. flag=1
        ii. Go to step 6
    c. Increment i by 1
6. if(flag==1)
    a. Display message **"String is not palindrome"**
7. else
    b. Display message **"String is palindrome"**
8. Stop

[IV] Algorithm push(char a)
Precondition: Accept acharacter variable
Postcondition: Stack with newly inserted element
Return: Nil

1.if(top==(max-1))

a. Display message "Stack Overflow"

b. Go to step 4

2. Increment top by 1

3.stack[top]=a

4. Stop

[IV] Algorithm pop()

Precondition : An already created stack

Postcondition : Element at the top of the stack

Return : char

1.if(top==-1)

a. Display message "Stack underflow."

b. Go to step 3

2.return (stack[top--])

3. Stop

Test Cases:

Input:

1. poor danisina droop

Output:

1. **Without removing punctuation, spaces and uppercase to lowercase**
   String is not palindrome
2. **After removing punctuation, spaces and uppercase to lowercase**
   String is palindrome
3. **Reverse**
   Poordanisinadroop

Conclusion:

Thus I have implemented a program to check whether the given string is palindrome or not(with and without punctuation) and also to display reverse string using stack.

Question:

1. What is stack? How it can be represented using array.
2. What are the applications of stack?
3. Write ADT for stack.
4. What are the conditions for empty and full stack?
5. What is palindrome?
6. How string is represented using array.

\

Experiment No: 08

Title: Program to check well formedness of parenthesis.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 8

Title: Program to check well formedness of parenthesis.

Objective: To understand theuse of stack by compiler to check well formedness of parenthesis of infix expression.

Problem Statement:

In any language program mostly syntax error occurs due to unbalancing delimiter such as (),{},[]. Write C++ program using stack to check whether given expression is well parenthesized or not

Outcomes:

Message showing whether given expression is properly parenthesized or not.

Software Requirement:

11. Linux operating system
12. Eclipse IDE with g++ compiler

Theory:

Stack:

Stack is a LIFO (Last In First Out) data structure. It is an ordered list of the same type of elements. It is a linear list in which all insertions and deletions are performed at the same end called **top.** When elements added to the stack it grows at one end, and when elements are deleted from stack it shrinks at the same end.

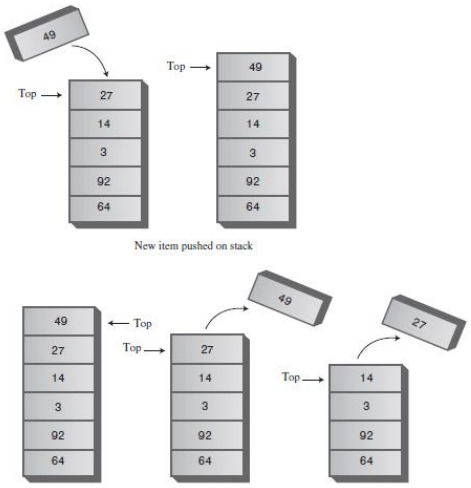


Figure: Insertion and deletion operation on stack

So, A *stack* is an ordered list in which all insertions and deletions are made at one end, called the *top*.
Associated with the stack there are several operations:

INITIALIZE (*S*): which creates *S* as an empty stack;
ADD (*i,S*): which inserts the element *i* onto the stack *S* and returns the new stack;
DELETE (*S*): which removes the top element of stack *S* and returns the new stack;
TOP (*S*): which returns the top element of stack *S*;
ISEMPTY (*S*): which returns true if *S* is empty else false;
ISFULL (S): which returns true is S is full else false;
These six functions constitute a working definition of a stack.
The simplest way to represent a stack is by using a one-dimensional array, say STACK(1:*n*), where *n* isthe maximum number of allowable entries. The first or bottom element in the stack will be stored atSTACK(1), the second at STACK(2) and the i-th at STACK(*i*). Associated with the array will be avariable, **top**, which points to the top element in the stack.

Declaration of stack as a structure:

```
struct Stack
{
        int stack[max];
        int top;
}st;
```

Algorithm:

[I] Algorithm check(char exp[])

Precondition : Accept the expression

Postcondition : Whether the expression is properly parenthesized or not

Return : int (0/1)

1.n=strlen(exp);
2.for(i=0;i<n;i++)
    a.if(exp[i]=='(' || exp[i]=='{' || exp[i]=='[')
        push(exp[i]);
    b.if(exp[i]==')' || exp[i]=='}' || exp[i]==']')
        i.if(st.top==-1)
            return 0
        ii.else
            I. temp=pop( )
            II.if(!match(temp,exp[i]))

return 0

3.if(st.top==-1)

a.return 1

4.else

b.return 0

5.Stop

[II] Algorithm match(char a,char b)

Precondition : Accept two characters(brackets)

Postcondition : Return whether the two brackets are matching or not

Return : int (0/1)

1.if(a=='[' && b==']')

a. return 1

2. if(a=='{' && b=='}')

a.return 1

3. if(a=='(' && b==')')

a. return 1

4. return 0

[III] Algorithm push(char item)

Precondition: Accept acharacter variable

Postcondition: Stack with newly inserted element

Return: Nil

1.if(st.top==(max-1))

a. Display message "Stack Overflow"

b. Go to step 4

2. Increment top by 1

3. st.stack[st.top]=item

4. Stop

[IV] Algorithm pop()

Precondition : An already created stack

Postcondition : Element at the top of the stack

Return : char

1.if(st.top==-1)

a. Display message “Stack underflow."

b. Go to step 3

2.return (st.stack[st.top--])

3. Stop

Test Cases:

Input:

2. (a+b*(c/d+e))
3. (x-y+[z*a]

Output:

1. Expression is well parenthesized
2. Expression is not well parenthesized

Conclusion:

Thus I have implemented a program to check well formedness of parenthesis using stack.

Experiment No: 09

Title: Program to job scheduling using queue

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 9

Title: Program to job scheduling using queue.

Objective: To understand thehow operating system uses queue to schedule job if no priority is associated with them.

Problem Statement:

Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job and delete job from queue.

Outcomes:

Job simulation using queue.

Software Requirement:

13. Linux operating system
14. Eclipse IDE with g++ compiler

Theory:

Queue:

The word *queue* is British for *line* (the kind you wait in). In Britain, to "queue up"means to get in line. In computer science a queue is a data structure that is somewhatlike a stack, except that in a queue the first item inserted is the first to beremoved (First-In-First-Out, FIFO), while in a stack, as we've seen, the last iteminserted is the first to be removed (LIFO). A queue works like the line at the movies:The first person to join the rear of the line is the first person to reach the front ofthe line and buy a ticket. The last person to line up is the last person to buy a ticket(or—if the show is sold out—to fail to buy a ticket). They're also used to model real-world situationssuch as people waiting in line at a bank, airplanes waiting to take off, or datapackets waiting to be transmitted over the Internet.There are various queues quietly doing their job in your computer's (or thenetwork's) operating system. There's a printer queue where print jobs wait for theprinter to be available. A queue also stores keystroke data as you type at thekeyboard. This way, if you're using a word processor but the computer is brieflydoing something else when you hit a key, the keystroke won't be lost; it waits in thequeue until the word processor has time to read it. Using a queue guarantees thekeystrokes stay in order until they can be processed.

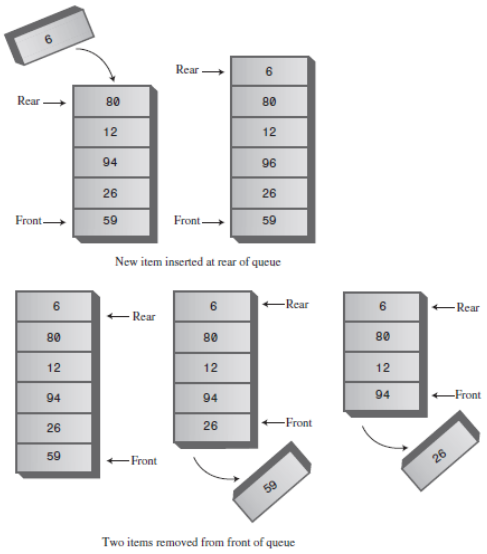


Figure: Insertion and deletion operation on queue

Associated with the queue there are several operations:

INITIALIZE (*Q*): which creates *Q*as an empty queue;
ADD (*i*,*Q*): which inserts the element *i*in the queue*Q*from front and returns the new queue;
DELETE (Q): which removes the front element of queue*Q*and returns the new queue;
ISEMPTY (Q): which returns true if *Q*is empty else false;
ISFULL (Q): which returns true is *Q* is full else false;
These five functions constitute a working definition of a queue.
The simplest way to represent a queue is by using a one-dimensional array, say QUEUE(1:*n*), where *n* isthe maximum number of allowable entries. The first or bottom element in the queue will be stored atQUEUE(1), the second at QUEUE(2) and the i-th at QUEUE(*i*). Associated with the array will be twovariables, **front,**which points to the first element, and **rear**, which points to the last element, in the queue.

Declaration of queue as a structure:

```
structqueue
{
        intque[max];
        intfront, rear;
}q;
```

Algorithm:

[I] Algorithm insert(int item)

Precondition: Accept ainteger variable, id of job
Postcondition: Queue with newly inserted job and modified value of rear
Return: int
1.if(q.front == -1)
a. Increment front by 1
2. Increment rear by 1
3.q.que[q.rear]=item
4. Return new value of rear index
5. Stop

[II] Algorithm delet()
Precondition : An already created queue
Postcondition : Job at the front of the queue
Return : int
1.item = q.que[q.front]
2. Increment front by 1
3. Display deleted job
4.returnnew value of front
5. Stop

Test Cases:

Input:
Output:

Conclusion:

Thus I have implemented a program to simulate job scheduling in operating system using queue.

Question:

7. What is queue? How it can be represented using array.
8. What are the applications of queue?
9. Write ADT for queue.
10. What are the conditions for empty and full queue?

Experiment No: 10

Title: Program to simulate operations on double ended queue.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 10

Title: Program to simulate operations on double ended queue.

Objective: To understand thedouble ended queue using array and implementation of insertion and deletion using both ends.

Problem Statement:

A double-ended queue(deque) is a linear list in which additions and deletions may be made at either end. Obtain a data representation mapping a deque into a one-dimensional array. Write C++ program to simulate deque with functions to add and delete elements from either end of the deque.

Outcomes:

Double ended queue with elements inserted and deleted from either end.

Software Requirement:

15. Linux operating system
16. Eclipse IDE with g++ compiler

Theory:

Double Ended Queue:

The short form of double ended queue is **dequeue.** It is a general representation for both queue and stack and can be used as stack and queue. In a dequeue, insertion as well as deletion can be carried out either at the rear or front.

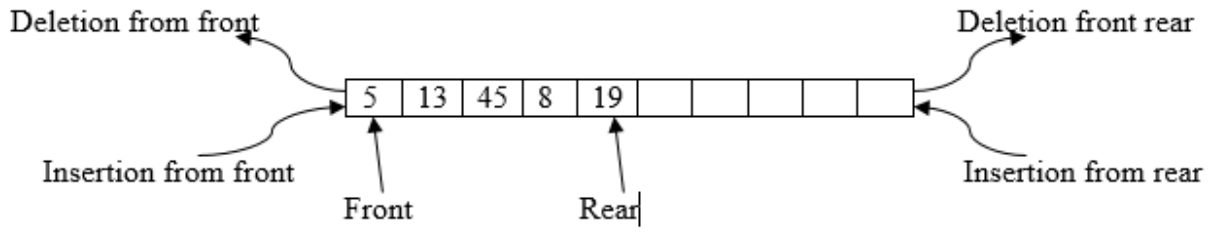


Figure: Double Ended Queue

Declaration of dequeue as a structure:

```
structdequeue
{
```

```
    intdeque[max];
    intfront, rear;
}q;
```

Operation on a Dequeue:

There various operations that can be performed on dequeue.

1. Initialize( ): Make the dequeue empty
2. IsEmpty( ): Determine if dequeue is empty or not
3. IsFull( ): Determine if dequeue is full or not
4. Insert_front( ):Insert an element at the front end of the dequeue
5. Insert_rear( ): Insert an element at the rear end of the dequeue
6. Delete_front( ): Delete an element from the front end of the dequeue
7. Delete_rear( ): Delete an element from the rear end of the dequeue
8. Display( ): Display contents of the dequeue from front to rear end

Algorithm:

[I] Algorithm insert_rear(int item)

Precondition: Accept aninteger variable.

Postcondition: Queue with newly inserted value and modified value of rear

Return: int

1.if(q.front == -1&&q.rear==-1)

a. Increment front by 1

2. Increment rear by 1

3.q.que[q.rear]=item

4. Return new value of rear index

5. Stop

[II] Algorithm delete_front()

Precondition : An already created dequeue

Postcondition : Dequeue withdeleted item.

Return : int

1.item = q.que[q.front]

2.q.que[q.front] = -1

3.Increment front by 1

4.Return item

5. Stop

[III] Algorithm insert_front(int item)

Precondition: Accept aninteger variable.

Postcondition: Queue with newly inserted value and modified value of front

Return: int
1. if(q.front==-1)
    a. Increment front by 1
2. i=q.front-1
3. while(i>=0)
    a. q.que[i+1]=q.que[i]
    b. Decrement i by 1
4. j=q.rear
5. while(j>=q.front)
    a. q.que[j+1]=q.que[j]
    b. Decrement j by 1
6. Increment j by 1
7. q.que[q.front]=item
8. Return new value of front index
9. Stop

[IV] Algorithmdelete_rear()
Precondition : An already created dequeue
Postcondition : Dequeue with deleted item.
Return : int
1. item=q.que[q.rear]
2. q.que[q.rear] = -1
3. Decrement rear by 1
4. return item
5. Stop

Test Cases:

Input:
In empty queue
1. Insert from rear value 10
2. Insert from rear value 30
3. Insert from front value 17
4. Insert from front value 98
5. Delete from rear
6. Delete from front

Output:
1. 10
2. 10, 30
3. 17, 10, 30
4. 98, 17, 10, 30

5. 98, 17, 10
6. 17, 10

Conclusion:

Thus I have implemented a program to represent and simulate operations of double ended queue using array.

Question:

11. What is double ended queue? How it can be represented using array.
12. What are the various ways to represent the dequeue
13. Write ADT for dequeue.
14. What are the conditions for empty and full queue?

Experiment No: 11

Title: Program for searching roll number using various searching techniques form a random and a sorted list.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

Experiment No: 11

Title: Program for searching roll number using various searching techniques form a random and a sorted list.

Objective: To understand all searching techniques and their implementation.

Problem Statement:

a) Write C++ program to store roll numbers of student in array who attended training program in random order. Write function for searching whether particular student attended training program or not using linear search and sentinel search.
b) Write C++ program to store roll numbers of student array who attended training program in sorted order. Write function for searching whether particular student attended training program or not using binary search and Fibonacci search.

Outcomes:

Message showing if the entered roll number is present in the given list.

Software Requirement:

17. Linux operating system
18. Eclipse IDE with g++ compiler

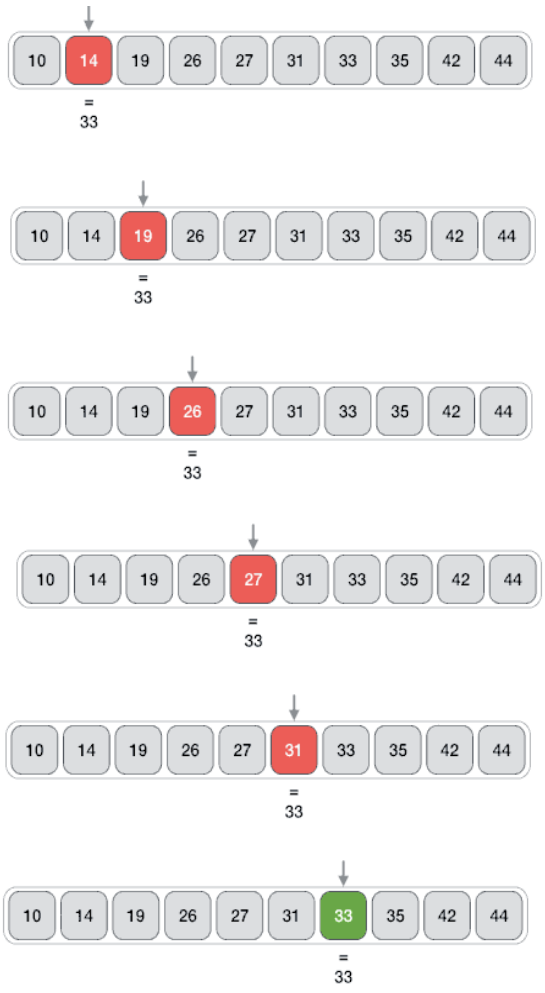
Theory:

Search:

Searching is a process of finding the target value in a given list and on successful search return its position. There are various types of searching techniques as given below:

1.Linear Search:

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every items is checked and if a match founds then that particular item is returned otherwise search continues till the end of the data collection.



Analysis:

Searching an element requires number of comparisons. For example if the target value is at 1st position in list that it requires 1 comparison, same way 2nd element requires 2 comparisons, Nth element requires N comparison. So on average **(1+2+…..+N)/N** comparisons are required. This can be represented as follows:

$$\frac{1}{N}\sum_{i=1}^{N} i = \frac{1}{N} * \frac{N^2+N}{2} = \frac{N+1}{2} = O(N)$$

Linear search performs one more comparison or check i.e.the loop index to terminate the loop if all elements are searched. That takes (N+1) comparison, which adds extra time complexity.

2.Sentinel Search:

The sentinel search reduces the time by eliminating the loop index check.This can be done by inserting the desired item itself as a sentinel value at the far end of the list, as in this pseudocode:

```
Set A[n + 1] to x.
Set i to 1.
Repeat this loop:
   If A[i] = x, then exit the loop.
   Set i to i + 1.
Return i.
```

With this stratagem, it is not necessary to check the value of *i* against the list length *n*: even if *x* was not in *A* to begin with, the loop will terminate when $i = n + 1$. However, this method is possible only if the array slot $A[n + 1]$ exists but is not being otherwise used. Similar arrangements could be made if the array were to be searched in reverse order, and element $A[0]$ were available.

Although the effort avoided by these ploys is tiny, it is still a significant component of the overhead of performing each step of the search, which is small. Only if many elements are likely to be compared will it be worthwhile considering methods that make fewer comparisons but impose other requirements.

3.Binary Search:

It is a search algorithm that finds the position of a target value within a sorted array. It compares the target value to the middle element of the array; if they are unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful.

Binary search works on sorted arrays. A binary search begins by comparing the middle element of the array with the target value. If the target value matches the middle element, its position in the array is returned. If the target value is less than or greater than the middle element, the search continues in the lower or upper half of the array, respectively, eliminating the other half from consideration

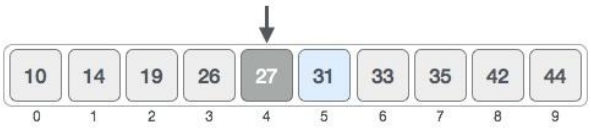
How binary search works:

The below given is our sorted array and assume that we need to search location of value 31 using binary search.

First, we shall determine the half of the array by using this formula −

mid = low + (high - low) / 2

Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5). So 4 is the mid of array.



Now we compare the value stored at location 4, with the value being searched i.e. 31. We find that value at location 4 is 27, which is not a match. Because value is greater than 27 and we have a sorted array so we also know that target value must be in upper portion of the array.

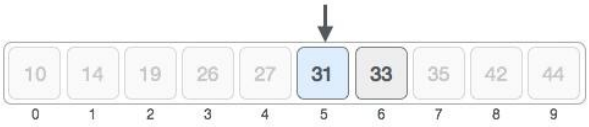
We change our low to mid + 1 and find the new mid value again.

low = mid + 1
mid = low + (high - low) / 2

Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.

The value stored at location 7 is not a match, rather it is less that what we are looking for. So the value must be in lower part from this location.

So we calculate the mid again. This time it is 5.



We compare the value stored ad location 5 with our target value. We find that it is a match.

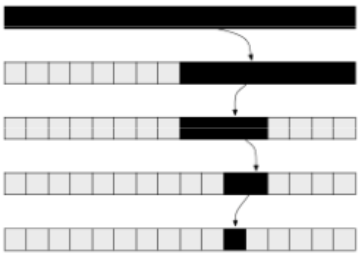
We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Analysis:
With every call to binsearch (or every while loop), we reduce the size of sub-array by 50%. In every call to binsearch ,we only do constant work. Thus, we call binsearch once with n, with n/2, with n/4, ...

log(n+1) times

Binsearch has worst-case complexity O(log(n)). Average case is onlymarginally better



4.<u>Fibonacci Search:</u>

Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.

Similarities with Binary Search:

1. Works for sorted arrays
2. A Divide and Conquer Algorithm.
3. Has Log n time complexity.
4.

Differences with Binary Search:

1. Fibonacci Search divides given array in unequal parts
2. Binary Search uses division operator to divide range. Fibonacci Search doesn't use /, but uses + and -. The division operator may be costly on some CPUs.
3. Fibonacci Search examines relatively closer elements in subsequent steps. So when input array is big that cannot fit in CPU cache or even in RAM, Fibonacci Search can be useful.

Background:

Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1. First few Fibinacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Observations:

Below observation is used for range elimination, and hence for the O(log(n)) complexity.

F(n - 2) ≈ (1/3)*F(n) and
F(n - 1) ≈ (2/3)*F(n).

Algorithm:

Let the searched element be x.

The idea it to first find the smallest Fibonacci number that is greater than or equal to length of given array. Let the found fibonacci number be fib (m'thfibonacci number). We use (m-2)'th Fibonacci number as index (If it is a valid index). Let (m-2)'th Fibonacci Number be i, we compare arr[i] with x, if x is same, we return i. Else if x is greater, we recur for subarray after i, else we recur for subarray before i.

Below is complete algorithm

Let arr[0..n-1] be th input array and element to be searched be x.

1. Find the smallest Fibonacci Number greater than or equal n. Let this number be fibM [m'th Fibonacci Number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)'th Fibonacci Number and fibMm2 [(m-2)'th Fibonacci Number./li>
2. While the array has elements to be inspected:
    a. Compare x with the last element of the range covered by fibMm2
    b. **If** x matches, return index
    c. **Else If** x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.

d. **Else** x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate elimination of approximately front one-third of the remaining array.

Since there might be a single element remaining for comparison, check if fibMm1 is 1. If Yes, compare x with that remaining element. If match, return index.

Illustration:

Let us understand the algorithm with below example:

i	1	2	3	4	5	6	7	8	9	10	11	12	13
ar[i]	**10**	**22**	**35**	**40**	**45**	**50**	**80**	**82**	**85**	**90**	**100**	-	-

Illustration assumption: 1-based indexing. Target element x is 85. Length of array n = 11.

Smallest Fibonacci number greate than or equal to 11 is 13. As per our illustration, fibMm2 = 5, fibMm2 = 8, and fibM = 13.

Another implementation detail is the offset variable (zero initialized). It marks the range that has been eliminated, starting from the front. We will update it time to time.

Now since the offset value is an index and all indices inclusive it and below it have been eliminated, it only makes sense to add something to it. Since fibMm2 marks approximately one-third of our array, as well as the indices it marks are sure to be valid ones, we can add fibMm2 to offset and check the element at index i = min(offset + fibMm2, n).

fibMm2	*fibMm1*	*fibM*	*offset*	i= min(offset+fibL n)	*arr[i]*	*Consequence*
5	8	13	0	5	45	Move one down, reset offset
3	5	8	5	8	82	Move one down, reset offset
2	3	5	8	10	90	Move two down
1	1	2	8	9	85	Return i

Analysis:

- Let's assume we can always compute x ~ l+2*(r-l)/3 using only integer additions and subtractions
- In the worst-case, we always have c in the larger (2/3) fraction of the array
  - We call once for n, once for 2n/3, once for 4n/9, …, 1
- I.e., we look at arrays of size fib(n-1), fib(n-2), fib(n-3), …
- Consider that

$$fib(n) = \left[\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n \right] \sim c*1.62^n$$

- Thus, for $n \sim c*1{,}62^{n'}$ (for some n') we make O(n') comparisons
- We thus need $1/c*\log_{1,62}(n)$=O(log(n)) comparisons

Algorithm:

[I] Algorithm SentinelSearch(int A[N+1],intmax,int key)

Precondition : Accept the Array, its size and key to be searched

Postcondition : Search result

Return : int

1.A[max]=key, where max is number of elements in array
2.i=0
3.while(A[i]!=key)
    Increment i by 1
4.A[max]=-1
5.if(i!=max)
    return 1
6.else
    return 0
7. Stop

[II] AlgorithmLinearSearch(int A[N],intmax,int key)

Precondition : Accept the Array, its size and key to be searched

Postcondition : Search result

Return : int

1.fori=0 to max(i.e. number of elements in array) , step up (i++)
    a.if(A[i]==key)
        i.return 1
2.return 0;
3.Stop

[III] Algorithm BinarySearch(int A[N],intkey,intlow,int high)
Precondition : Accept the Array, key to be searched, low and high index
Postcondition : Search result
Return : int
1.if(low>high)
return -1
2.mid=(low+high)/2
3.if(key>A[mid])
returnBinarySearch(A,key,mid+1,high)
4.else if(key<A[mid])
returnBinarySearch(A,key,low,mid-1)
5.else
return mid
6. Stop

[IV] Algorithm Fib(int n)
Precondition : Accept the number of elements in the array
Postcondition :value of a
Return : int
1.if(n<1)
return n
2.a=0
3.b=1
4.while(b<n)
a. Set f=a+b
b. Set a=b
c. Set b=f
5.return new value of a
6.Stop

[V] Algorithm FibSearch(int A[N],intn,intkey,intf,intb,int a)
Precondition : Accept the Array, its size, key to be searched and values of f, b and a
Postcondition : Search result
Return : Nil
1.if(f<1||f>n)
Display message **"Student has not attended training."**
2.else if(key<A[f-1])
a.if(a<=0)
Display message **"Student has not attended training."**
b.else
Recursively call itself i.e. FibSearch(A,n,key,f-a,a,b-a)
3.else if(key>A[f])

a.if(b<=1)
Display message **"Student has not attended training."**
b.else
Recursively call itself i.e.FibSearch(A,n,key,f+a,b-a,a-b)

4.else
Display message **"Student has attended training."**

5.Stop

Test Cases:

Input:

Enter total number of students: 10

1. **Enter roll numbers randomly**
   56
   93
   49
   63
   57
   90
   81
   48
   34
   70
2. **Enter roll numbers in ascending order**
   34
   48
   49
   56
   57
   63
   70
   81
   90
   93

Output:

1. Linear and Sentinel Search
   Roll number to be searched: 34
   Student has attended training program
   Number of comparisons: 9
2. Binary and Fibonacci search

Roll number to be searched: 34
Student has attended training program
Number of comparisons: 4

Conclusion:

Thus I have implemented linear and sentinel search algorithms for searching a key from random list and binary and Fibonacci search algorithms for searching a key from sorted list.

Question:

15. What is complexity of Searching?
16. What is sequential search and what are its variants?
17. Explain Fibonacci and Binary search with the help of example?
18. What is recursion and recursive algorithm?
19. What is average case complexity of all searching methods?

Experiment No: 12

Title: Program for sorting floating point numbers using bubble sort and selection sort.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 12

Title: Program for sorting floating point numbers using bubble sort and selection sort.

Objective: To understand themost basic sorting method i.e. bubble sort and selection sort and its implementation.

Problem Statement:

Write C++ program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using
a) Selection Sort
b) Bubble sort and display top five scores.

Outcomes:

Top five scores.

Software Requirement:

19. Linux operating system
20. Eclipse IDE with g++ compiler

Theory:

Sorting:

When you rearrange data and put it into a certain order, you are **sorting** the data. You can sortdata alphabetically, numerically, and in other ways. Often you need to sort data before you usesearching algorithms to find a particular piece of data.

1.Bubble Sort

The idea of bubble sort is to compare two adjacent elements. If they are not in the right order,switch them. Do this comparing and switching (if necessary) until the end of the array is reached.Repeat this process from the beginning of the array n times.

Bubble Sort Example

Here we want to sort an array containing [8, 5, 1]. The following figure shows how we can sortthis array using bubble sort. The elements in consideration are shown in **bold.**

8, 5, 1	Switch 8 and 5
5, **8, 1**	Switch 8 and 1
5, 1, 8	Reached end start again.
5, 1, 8	Switch 5 and 1
1, **5, 8**	No Switch for 5 and 8

1, 5, 8 Reached end start again.
1, 5, 8 No switch for 1, 5
1, **5, 8** No switch for 5, 8
1, 5, 8 Reached end.

But do not start again since this is the nth iteration of same process

2.Selection Sort

The name of Selection Sort comes from the idea of selecting the smallest element from thoseelements not yet sorted. The smallest element is then swapped with the first unsorted element.Here is the basic process of sorting an n-element array, A, using selection sort:

1. Find the smallest element from A[0]…A[n]
2. Swap that smallest element with A[0]
3. Find the smallest element from A[1]…A[n]
4. Swap that smallest element with A[1]
5. Find the smallest element from A[2]…A[n]
6. Swap that smallest element with A[2]

.
.
.

Continue this process until the last element in the array.

Selection Sort Example

Here we are sorting an array containing the following numbers:

8, 27, 33, 2, 20, 12, 19, 5

In the following figure:

- The already sorted part is shown in *italics*
- The first unsorted element is shown underlined
- The minimum element of the unsorted part is shown in **bold**

8, 27, 33, **2**, 20, 12, 19, 5
2, 27, 33, 8, 20, 12, 19, **5**
2, *5*, 33, **8**, 20, 12, 19, 27
2, *5*, *8*, 33, 20, **12**, 19, 27
2, *5*, *8*, *12*, 20, 33, **19**, 27
2, *5*, *8*, *12*, *19*, 33, **20**, 27
2, *5*, *8*, *12*, *19*, *20*, 33, **27**
2, *5*, *8*, *12*, *19*, *20*, *27*, **33**
2, *5*, *8*, *12*, *19*, *20*, *27*, *33*

Resulting sorted array

Analysis:

To sort n elements, both sorts perform**n - 1** passes:
on 1st pass, it performs n - 1 comparisons
on 2nd pass, it performs n - 2 comparisons
…
on the (n-1)st pass, it performs 1 comparison

Adding up the comparisons for each pass, we get:

C(n) = 1 + 2 + … + (n - 2) + (n - 1)

The resulting formula for C(n) is the sum of an arithmeticsequence:

$$C(n) = 1 + 2 + \ldots + (n - 2) + (n - 1) = \sum_{i=1}^{n-1} i$$

Formula for the sum of this type of arithmetic sequence:

$$\sum_{i=1}^{m} i = \frac{m(m+1)}{2}$$

Thus, we can simplify our expression for C(n) as follows:

$$C(n) = \sum_{i=1}^{n-1} i$$
$$= \frac{(n-1)((n-1)+1)}{2}$$
$$= \frac{n(n-1)}{2}$$
$$= \frac{n^2}{2} - \frac{n}{2}$$

So we can say that $C(n) = \frac{n^2}{2} - \frac{n}{2}$ is $O(n^2)$

Algorithm:

[I] Algorithm SelectionSort()

Precondition : Accept the Array to be sorted
Postcondition : Sorted array
Return : Nil

1.for i=0 to n, step up(i++)
- **a.** for j=i+1 to n, step up(j++)
  - **i.** if(a[i] > a[j]) then swap them i.e.
    - **I.** temp=a[i]
    - **II.** a[i]=a[j]
    - **III.** a[j]=temp

2.Stop

[I] Algorithm BubbleSort()

Precondition : Accept the Array to be sorted
Postcondition : Sorted array
Return : Nil

1.for i=0 to n, step up(i++)
- **a.** for j=0 to n-i-1, step up(j++)
  - **i.** if(a[j] > a[j+1]) then swap them i.e.
    - **I.** temp=a[j]
    - **II.** a[j]=a[j+1]

III. a[j+1]=temp

2.Stop

Test Cases:

Input:

Enter total number of students: 10
Enter percentage marks of students...
56
93
49
63
57
90
81
48
34
70

Output:

34
48
49
56
57
63
70
81
90
93

Conclusion:

Thus I have implemented bubble sort and selection sort algorithm to sort randomly entered percentage of students.

Question:

20. What is sorting?
21. Explain various sorting methods.
22. What is time complexity of bubble sort and selection sort algorithm?

Experiment No: 13

Title: Program for sorting floating point numbers using quick sort.

Roll No:_____ Class:_____ Batch:_____

Date of Performance: ___ /___/_____

Date of Assessment : ___ /___/_____

ASSIGNMENT NO. 13

Title: Program for sorting floating point numbers using quick sort.

Objective: To understand the fastersorting method i.e. quick sort and its implementation.

Problem Statement:

Write C++ program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using quick sort and display top five scores.

Outcomes:

Top five scores.

Software Requirement:

21. Linux operating system
22. Eclipse IDE with g++ compiler

Theory:

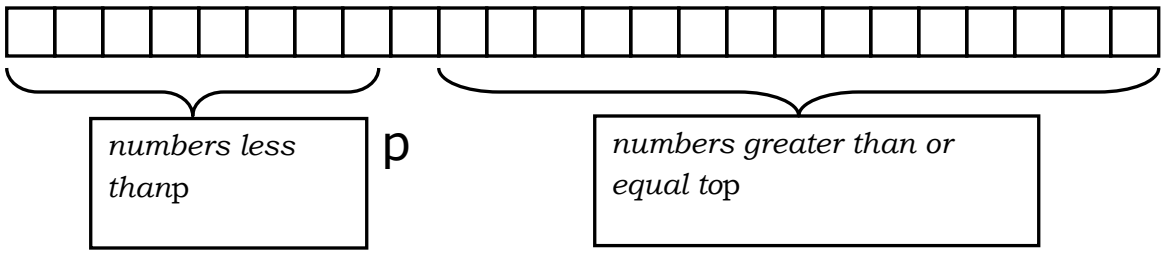
Quick Sort

Quick Sort also known as partition exchange sort.
Original algorithm was developed by C.A.R. Hoare in 1962.

Quick sort sorts by employing a divide and conquer strategy to divide a list into two sub-lists. It is one of the fastest sorting algorithms available. Quick Sort is especially convenient with large arrays that contain elements in random order.

The steps are:

1. Pick an element, called a *pivot*, from the list.
2. Reorder the list so that all elements which are less than the pivot come before the pivot and so that all elements greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively sort the sub-list of smaller elements and the sub-list of greater elements.



Efficiency/Complexity:

The worst-case performance of Quicksort is $\Theta(n^2)$.

The best cases are when the array is split half and half.The best case running time is $\Theta(nlogn)$
The average case performance of QuickSort is $\Theta(nlogn)$.

Algorithm:

[I] Algorithm quicksort(int p, int q,)

Precondition : Accept the Array to be sorted
Postcondition : Sorted array
Return : Nil

2. if(p<q)
    - **a.** j=partition(p,q)
    - **b.** call function, quicksort(p,j-1)
    - **c.** call function, quicksort(j+1,q)
3. End if

[II] Algorithm Partition(int low, int high)

1. i =low+1
2. j =high-1
3. pivot = arr[low]
4. while i < j do
    - **a.** while arr[i] < pivot, increment i by 1 (i++)
    - **b.** while arr[j] > pivot, decrement j by 1 (j--)
    - **c.** if i < j then
        - **i.** temp = arr[i]
        - **j.** arr[i] = arr[j]
        - **k.** arr[j] = temp

5. temp = arr[j]
6. arr[j] = arr[low]

7. arr[low]= temp
8. return j
9. Stop

Test Cases:

Input:

Enter total number of students: 10
Enter percentage marks of students...
56
93
49
63
57
90
81
48
34
70

Output:

Quick Sort .....
34
48
49
56
57
63
70
81
90
93

Conclusion:

Thus I have implemented quick sort algorithm recursively to sort randomly entered percentage of students.

Question:

1. What is complexity of quick sort?
2. Which element in list is generally chosen as pivot?
3. What is recursion and recursive algorithm?
4. Which algorithmic strategy is used in quick sort?