	Uttarakhand Open University, Haldwani School of Computer Science & IT	LABORATORY MANUAL
	PRACTICAL INSTRUCTION SHEET	
DEPTT.: Computer Science& IT	LABORATORY: Data Structure Lab (BCA-06)	SEMESTER: II

INDEX

PREAPEARED BY: Balam Singh Dafouti

Experiment No-1

IMPLEMENTATION OF STACK USING ARRAYS

After performing this experiment students are able to do-
Concept of array-

An array is a collection of elements of same data types. Ordinary variables are capable of holding only one value at a time however there are some situation where we would want to store more then one value at a time in a single variable e.g. we want to arrange the percentage marks of 100 student In ascending order in such case we have two option, construct 100 variables to store percentage or construct one variable capable of holding all 100 variable, such a variable is called array.

Concept of switch statement- C programming language provides a multiple-branch selection statement known as **switch**. Switch statement is a substitute for a series of if ….. Else or else ….. if statement. The basic format for using switch case is sown below-

```
Switch (expression or variable)
{
    case variable equals this :
    do this ;
    break;
    case variable equals this :
    do this ;
    break ;
    case variable equals this :
    do this ;
    break;
        .
   dfault :
   do this
 }
```

A switch statement is used when one out of many course of action has to be selected.

BACKGROUND-

STACK- A **stack** is a limited version of an array. New elements, or **nodes** as they are often called, can be added to a stack and removed from a stack only from one end. For this reason, a stack is referred to as a LIFO structure (Last-In First-Out).
Stacks have many applications. For example, as processor executes a program, when a function call is made, the called function must know how to return back to the program, so the current address of program execution is pushed onto a stack. Once the function is finished, the address that was saved is removed from the stack, and execution of the program resumes. If a series of function calls occur, the successive return values are pushed onto the stack in LIFO order so that each function can return back to calling program. Stacks support recursive function calls in the same manner as conventional nonrecursive calls.
Stacks are also used by compilers in the process of evaluating expressions and generating machine language code. They are also used to store return addresses in a chain of method calls during execution of a program.

Stack - Array Implementation

1. **Implementing a stack with an array:**

   Let's think about how to implement this stack in the C programming language.

   First, if we want to store letters, we can use type char. Next, since a stack usually holds a bunch of items with the same type (e.g., char), we can use an array to hold the contents of the stack.

   Now, consider how we'll use this array of characters, call it contents, to hold the contents of the stack. At some point we'll have to decide how big this array is; keep in mind that a normal array has a fixed size.

   Let's choose the array to be of size 4 for now. So, an array getting **A**, then **B**, will look like:

```
-----------------
| A | B |   |   |
-----------------
  0   1   2   3
```

```
contents
```

Is this array sufficient, or will we need to store more information concerning the stack?

Answer: We need to keep track of the *top* of the stack since not all of the array holds stack elements.

*What **type** of thing will we use to keep track of the top of the stack?*

Answer: One choice is to use an integer, top, which will hold the array index of the element at the top of the stack.

Example:

Again suppose the stack has (A,B) in it already...

```
stack (made up of 'contents' and 'top')
-----------------   -----
| A | B |   |   |   | 1 |
-----------------   -----
  0   1   2   3      top
contents
```

Since **B** is at the top of the stack, the value *top* stores the index of **B** in the array (i.e., 1).

Now, suppose we push something on the stack, Push(stack, 'C'), giving:

```
stack (made up of 'contents' and 'top')
-----------------   -----
| A | B | C |   |   | 2 |
-----------------   -----
  0   1   2   3      top
contents
```

(Note that both the *contents* and *top* part have to change.)

So, a sequence of pops produce the following effects:

```
1.    letter = Pop(stack)
2.    stack (made up of 'contents' and 'top')
3.    -----------------   -----    -----
4.    | A | B |   |   |   | 1 |    | C |
5.    -----------------   -----    -----
6.      0   1   2   3      top     letter
7.    contents
8.    letter = Pop(stack)
9.    stack (made up of 'contents' and 'top')
10.   -----------------   -----    -----
11.   | A |   |   |   |   | 0 |    | B |
12.   -----------------   -----    -----
13.     0   1   2   3      top     letter
14.   contents
15.   letter = Pop(stack)
16.   stack (made up of 'contents' and 'top')
17.   -----------------   -----    -----
18.   |   |   |   |   |   | -1|    | A |
19.   -----------------   -----    -----
20.     0   1   2   3      top     letter
21.   contents
```

so that you can see what value *top* should have when it is empty, i.e., -1.

Let's use this implementation of the stack with **contents** and **top** fields.

What happens if we apply the following set of operations?

22. Push(stack, 'D')
23. Push(stack, 'E')
24. Push(stack, 'F')
25. Push(stack, 'G')

giving:

```
stack (made up of 'contents' and 'top')
-----------------   -----
| D | E | F | G |   | 3 |
-----------------   -----
  0   1   2   3      top
contents
```

*and then try to add **H** with Push(stack, 'H')?*

ALGORITHM:-

Step 1: Start the process.

Step 2: Declare and initialize the variables.

Step 3: Enter the choice to perform PUSH or POP.

Step 4: If choice is PUSH enter the elements to push.

Step 5: If (top > max-2) print stack overflow else VAL[++TOP] = X.

Step 6: Print the stack elements after push.

Step 7: If choice is pop and if (TOP < 0) then print stack underflow.

Step 8: Else X=VAL[TOP] and TOP = TOP -1.

Step 9: Return elements that is popped.

Step 10: Print stack elements after the pop process.

Step 11: Stop the process.

Program-Array Implementation of a Stack

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200
int stack[MAXSIZE];
int top; //index pointing to the top of stack
void main()
{
void push(int);
int pop();
int will=1,i,num;
clrscr();

while(will ==1)
{
printf("
                    MAIN MENU:
          1. Add element to stack
          2. Delete element from the stack
");
scanf("%d",&will);
switch(will)
{
case 1:
          printf("
Enter the data... ");
          scanf("%d",&num);
          push(num);
          break;
case 2: i=pop();
          printf("
Value returned from pop function is  %d ",i);
          break;
default: printf("Invalid Choice . ");
}

printf(" Do you want to do more operations on Stack ( 1 for yes, any other key to
exit) ");
scanf("%d" , &will);
} //end of  outer while
}              //end of main

void push(int y)
{
if(top>MAXSIZE)
      {
      printf("STACK FULL");
      return;
      }
else
          {
          top++;
          stack[top]=y;
          }
}

int pop()
{
int a;
if(top<=0)
          {
          printf("STACK EMPTY");
          return 0;
          }
else
          {
          a=stack[top];
          top--;
          }
return(a);
}
```

Lab Assignment:

1.Write a program to push 5 elements into stack and then display them using array.

2.Write a program to push 5 integers into the stack and then pop them one by one.

3.Write a program to print the elements of stack in reverse order.

4.Write a program to copy a number of elements from one stack to another stack.

5.Write a program to merge the elements of two stacks into third stack.

Array implementation of a Circular Queue

BACKGROUND-

Queue -Whether it is a railway reservation counter, a movie theatre or print jobs submitted to a network printer there is only one way to bring order to chose─form a queue. If you await your turn patiently there is a more likelihood that you would get a better service.
Queue is a linear data structure that permits insertion of new element at one end and deletion of an element at the other end. The end at which the deletion of an element take place is called **front**, and the end at which insertion of a new element can take place is called **rear**. The deletion or insertion of elements can take place only at the front or rear end of the respectively.

A circular queue- A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full. In other words if we have queue Q of say n elements, then after inserting an element last location of the array the next element will be inserted at the very first location of the array. It is possible to insert new elements, if and only if those locations are empty. We can say that a circular queue is one in which the first element comes just after the last element. It can be viewed as a mess or loop of wire, in which the two ends of the wire are connected together. .

ALGORITHM:-

```
Step 1: Start the process.

Step 2: Declare and initialize the variables.

Step 3: Enter the choice to perform Insertion or Deletion.

Step 4: If choice is Insertion enter the elements to insert.

Step 5: If  (Front = = (Rear + 1) % MAXSIZE)
        Write Queue Overflow and Exit.

         Else : Take the value
                If ( Front = = -1)
                Set Front = Rear = 0
                Rear = ((Rear + 1) % MAXSIZE)
                [Assign Value] Queue[Rear] = elements.
                [End if]

Step 6: Print the queue elements after insertion and exit.

Step 7: If choice is Deletion and if  (Front = = -1)
        Write Queue underflow and Exit.
        Else : element = Queue (Front)
        If (Front = = Rear)
        Set Front = -1
        Rear = -1
Else : Front = (Front + 1) % MAXSIZE
[End if Structure]

Step 8: Print queue elements after the deletion process and exit.
Step 9: Stop the process.
```

Program-Array Implementation of a Circular Queue

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200
int cq[MAXSIZE];
int front,rear;
void main()
{
void add(int,int [],int,int,int);
int del(int [],int ,int ,int );
int will=1,i,num;
front = 1;
rear = 1;
clrscr();
printf("Program for Circular Queue demonstration through array");
while(will ==1)
{
printf("
                MAIN MENU:
        1.Add element to Circular Queue
```

```
            2.Delete element from the Circular Queue
");
scanf("%d",&will);
switch(will)
{
case 1:
            printf("Enter the data... ");
            scanf("%d",&num);
            add(num,cq,MAXSIZE,front,rear);
            break;
case 2: i=del(cq,MAXSIZE,front,rear);
            printf("Value returned from delete function is  %d ",i);
            break;
default: printf("Invalid Choice . ");
}

printf(" Do you want to do more operations on Circular Queue ( 1 for yes, any other key
to exit) ");
scanf("%d" , &will);
} //end of  outer while
}               //end of main

void add(int item,int q[],int MAX,int front,int rear)
{
rear++;
rear= (rear%MAX);
if(front ==rear)
            {
            printf("CIRCULAR QUEUE FULL");
            return;
            }
else
            {
            cq[rear]=item;
            printf("Rear = %d    Front = %d ",rear,front);
            }
}
int del(int q[],int MAX,int front,int rear)
{
```

```
int a;
if(front == rear)
        {
        printf("CIRCULAR QUEUE EMPTY");
        return (0);
        }
else
        {
        front++;
        front = front%MAX;
        a=cq[front];
        return(a);
        printf("
Rear = %d    Front = %d ",rear,front);
        }
}
```

Lab Assignment:

1.Write a program to insert 5 elements into circular queue and then display them.

2.Write a program to insert 5 integers into the circular queue and then delete them.

3.Write a program to check whether a circular queue is full of empty.

4.Write a program to insert a new item at the place of some deleted item.

5.Write a program to display all the items of a circular queue, if REAR < FRONT .

Experiment No-3

	Uttarakhand Open University, Haldwani **School of Computer Science & IT**	LABORATORY MANUAL
	PRACTICAL INSTRUCTION SHEET	
DEPTT.: Computer Science& IT	LABORATORY: Data Structure Lab (BCA-06)	SEMESTER: II

OPERATION IMPLEMANTATION OF STACK

BACKGROUND-

Operations

An abstract data type (ADT) consists of a data structure and a set of **primitive operations**. The main primitives of a stack are known as:

Software Development 2 Bell College

12. The STACK Data Structure page 2

Push adds a new node

Pop removes a node

Additional primitives can be defined:

IsEmpty reports whether the stack is empty

IsFull reports whether the stack is full

Initialise creates/initialises the stack

Destroy deletes the contents of the stack (may be implemented by re-initialising the stack)

Initialise

Creates the structure – i.e. ensures that the structure exists but contains no elements

e.g. ***Initialise(S)*** creates a new empty stack named S

Push

e.g. ***Push(X,S)*** adds the value X to the TOP of stack S

Pop

e.g. ***Pop(S)*** removes the TOP node and returns its value

S

X

S

S

Software Development 2 Bell College

12. The STACK Data Structure page 3

Examples-

C

B B B

A A A A

s.push(‘A’); s.push(‘B’); s.push(‘C’);

s.pop();

returns C

F

B B
A A A
s.push('F'); s.pop();
returns F
s.pop();
returns B
s.pop();
returns A
We could try the same example with actual values for A, B and C.
A = 1 B = 2 C = 3

1. **StackPush():**

   Now, pushing onto the stack requires the stack itself as well as *something to push*. So, its prototype will look like:

   ```
   void StackPush(stackT *stackP, stackElementT element);
   ```

   The function should place an element at the correct position in the *contents* array and update the *top*. However, before the element is placed in the array, we should make sure the array is not already full...Here is the body of the function:

   ```
   void StackPush(stackT *stackP, stackElementT element)
   {
     if (StackIsFull(stackP)) {
       fprintf(stderr, "Can't push element on stack: stack is full.\n");
       exit(1);  /* Exit, returning error code. */
     }

     /* Put information in array; update top. */

     stackP->contents[++stackP->top] = element;
   }
   ```

   Note how we used the *prefix* ++ operator. It increments the *top* index **before** it is used as an index in the array (i.e., where to place the new element).

   Also note how we just reuse the StackIsFull() function to test for fullness.

2. **StackPop():**

Finally, popping from a stack only requires a stack parameter, but the value popped is typically returned. So, its prototype will look like:

```
stackElementT StackPop(stackT *stackP);
```

The function should return the element at the top and update the *top*. Again, before an element is removed, we should make sure the array is not empty....Here is the body of the function:

```
stackElementT StackPop(stackT *stackP)
{
  if (StackIsEmpty(stackP)) {
    fprintf(stderr, "Can't pop element from stack: stack is empty.\n");
    exit(1);  /* Exit, returning error code. */
  }

  return stackP->contents[stackP->top--];
}
```

Note how we had the sticky problem that we had to update the *top* before the function returns, but we need the *current value of top* to return the correct array element. This is accomplished easily using the *postfix* -- operator, which allows us to use the current value of *top* before it is decremented.

ALGORITHM:-

Step 1: Start the process.

Step 2: Declare and initialize the variables.

Step 3: Enter the choice to perform PUSH or POP.

Step 4: If choice is PUSH enter the elements to push.

Step 5: If (top > max-2) print stack overflow else VAL[++TOP] = X.

Step 6: Print the stack elements after push.

Step 7: If choice is POP and if (TOP < 0) then print stack underflow.

Step 8: Else X=VAL[TOP] and TOP = TOP -1.

Step 9: Return elements that is popped.

Step 10: Print stack elements after the pop process.

Step 11: Stop the process.

PROGRAM OF STACK IMPLIMENTATION-

```
#include<stdio.h>
#include<conio.h>
#define size 2
int stack[size],top=-1,b,res;
void push();
void pop();
void display();
void main()
{
int c;
clrscr();
printf("1.push\n");
printf("2.pop\n");
printf("3.display\n");
do
{
printf("\n enter your choice");
scanf("%d",&c);
switch(c)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
printf("\n contents of stack\n");
display();
break;
default:
printf("invalid choice");
exit(0);
}
}
while(c<4);
getch();
}
void push()
{
if(top>=size)
{
printf("stack is overflow");
return;
}
else
{
printf("enters the number to be pushed\n");
scanf("%d",&b);
top++;
stack[top]=b;
printf("number pushed:%d",stack[top]);
return;
}
}
void pop()
{
if (top==-1)
{
printf("stack is overflow");
return;
}
else
{
res=stack[top];
top--;
printf("deleted one is %d",res);
return;
```

```
}
}
void display()
{
int i;
if(top==-1)
{
printf("stack is overflow");
return;
}
for(i=top;i>=0;i--)
{
printf("%d\n",stack[i]);
}
}\
```

Lab Assignment:

1. Explain the concept of PUSH & POP for the stack.

2. How will you copy the items of one stack into another stack.

3.Write a program to arrange the items of a stack into ascending order.

<u>Experiment No-4</u>

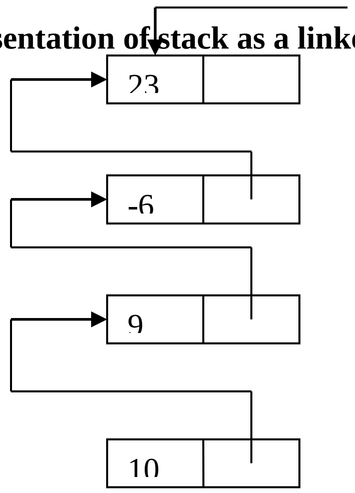
IMPLEMENTATION OF STACK USING LINKED LIST

BACKGROUND-

Stack as linked list- When implemented as an array **it** suffers from the basic limitation of an array – that its size can not be increased or decreased once it is declared. As a result , one ends up reserving either too much space or too less space for an array and in turn for a stack. This problem can be overcome if we implement a stack using a linked list. In case of linked stack we shall push and pop nodes from one end of a linked list.
The stack as linked list is represented as a singly connected list. Each node in the linked list contains the data and a pointer that gives location of the next node in the list. The node in the list is a structure as shown below:

```
Struct node
{
  <data type> data;
  node *link;
}
```

Representation of stack as a linked list



<u>**ALGORITHM:-**</u>

Step 1: Start the process.

Step 2: Initialize and declare the variables.

Step 3: Enter the choice PUSH or POP.

Step 4: If choice is PUSH then

a) Check the condition (TOP = MAX) and display stack is full if true.

b) Else get a new node and enter the item in the data field and adjust the link field so that top points to the new node.

Step 5: If choice = POP then

a) Check the condition (TOP = 0) and display stack is empty if true.

b) Else set ITEM = DATA(TOP) , delete the node pointed to by TOP and adjust the link fields so that TOP points to the top of the stack.

Step 6: Print the stack elements after PUSH/POP process.
Step 7: Stop the process.

Lab Assignment:

1.Write a program to push 10 elements into stack and then display them using linked list.

2.Write a program to push 5 integers into the stack and then pop them using linked list.

3.Write a program to print the elements of stack in reverse order using linked list.

4.Write a program to copy a number of elements from one stack to another stack using linked list.

5.Write a program to merge the elements of two stacks into third stack using linked list.

<u>ExperimentNo-5</u>

LINKEDLIST IMPLEMENTATION OF QUEUE

After performing this experiment students are able to do-

Concept of structure-

structure definition:
general format:

```
struct tag_name
{
data type member1;
data type member2;
…
…
}
```

Example:

```
struct lib_books
{
char title[20];
char author[15];
int pages;
float price;
};
```

the keyword struct declares a structure to holds the details of four fields namely title, author pages and price. These are members of the structures. Each member may belong to different or same data type. The tag name can be used to define objects that have the tag names structure. The structure we just declared is not a variable by itself but a template for the structure.
We can declare structure variables using the tag name any where in the program. For example the statement-

```
struct lib_books book1,book2,book3;
```

declares book1,book2,book3 as variables of type struct lib_books each declaration has four elements of the structure lib_books. The complete structure declaration might look like this

```
struct lib_books
{
char title[20];
char author[15];
int pages;
float price;
};
```

struct lib_books, book1, book2, book3;

structures do not occupy any memory until it is associated with the structure variable such as book1. the template is terminated with a semicolon. While the entire declaration is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template. The tag name such as lib_books can be used to declare structure variables of its data type later in the program.

We can also combine both template declaration and variables declaration in one statement, the declaration

```
struct lib_books
{
char title[20];
char author[15];
int pages;
float price;
} book1,book2,book3;
```

is valid. The use of tag name is optional for example

```
struct
{
…
…
…
}
```

book1, book2, book3 declares book1,book2,book3 as structure variables representing 3 books but does not include a tag name for use in the declaration.

A structure is usually defines before main along with macro definitions. In such cases the structure assumes global status and all the functions can access the structure.

BACKGROUND- As a dynamic list will implement the Queue, we won't check the 'queue overflow' condition here. Here, we have two pointers -- front and rear, pointing to beginning and end of the Queue. When 'front' and 'rear' both point to NULL, Queue is empty. Every time we add an element to the queue, the 'rear' pointer shifts forward to point to that newly added element.

ALGORITHM:-

```
Step 1: Start the process.

Step 2: Initialize and declare the variables.

Step 3: Enter the choice INSERTION or DELETION.

Step 4: If choice is INSERTION then

        b)  Check the condition (Rear = MAX) and display queue is full if true.

        b)  Else get a new node and enter the item in the data field and adjust the link field
    so that rear points to the new node.

Step 5: If choice = DELETION then

        c)  Check the condition (Front = -1) and display queue is empty if true.

        d)  Else if (Front = Rear), delete the node pointed to by Front and Front's link is
            NULL.

        e)  Else Front =  Front→link

Step 6: Print the queue elements after INSERTION or DELETION process.
Step 7: Stop the process.
```

PROGRAM :

```
#include<stdio.h>
#include<conio.h>
void ins();
void del();
void dis();

struct node
{
int data;
struct node *next;
}*f=NULL,*r=NULL;

void main()
{int ch;
clrscr();
printf("\n1.insertion");
printf("\n2.deletion");
printf("\n3.display");
do
{
printf("\n enter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1:ins();break;
case 2:del();break;
case 3:dis();break;
default:
printf("\n invalid choice");
break;
}}
while(ch<4);
}

void ins()
{
int x;
struct node *newnode;
newnode=malloc(sizeof(struct node));
printf("\n enter the number");
scanf("%d",&x);
newnode->data=x;
```

```
newnode->next=NULL;
if(r==NULL)
{
f=newnode;
r=newnode;
}
else
{
r->next=newnode;
r=newnode;
}
printf("\nthe element %d is inserted",x);
getch();
}

void del()
{struct node *t;
if(f==NULL)
{
printf("\n queue is empty");
return;
}
t=f;
if(f==r)
f=r=NULL;
else
f=f->next;
printf("\n the element %d is deleted",t->data);
free(t);
getch();
}
void dis()
{
struct node *t;
if(f==NULL)
{
printf("\nQueue is empty");
return;
}
```

```
t=f;
printf("\ncontents of the queue");
while(t!=NULL)
{
printf("%d",t->data);
t=t->next;
}
getch();
}

#include<stdio.h>
#include<conio.h>
void ins();
void del();
void dis();

struct node
{
int data;
struct node *next;
}*f=NULL,*r=NULL;

void main()
{int ch;
clrscr();
printf("\n1.insertion");
printf("\n2.deletion");
printf("\n3.display");
do
{
printf("\n enter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1:ins();break;
case 2:del();break;
case 3:dis();break;
default:
printf("\n invalid choice");
break;
}}
while(ch<4);
}

void ins()
{
int x;
struct node *newnode;
newnode=malloc(sizeof(struct node));
printf("\n enter the number");
scanf("%d",&x);
newnode->data=x;
newnode->next=NULL;
if(r==NULL)
{
f=newnode;
r=newnode;
}
else
{
r->next=newnode;
r=newnode;
}
printf("\nthe element %d is inserted",x);
getch();
}

void del()
{struct node *t;
if(f==NULL)
{
printf("\n queue is empty");
return;
}
t=f;
if(f==r)
f=r=NULL;
else
```

```
f=f->next;
printf("\n the element %d is deleted",t->data);
free(t);
getch();
}
void dis()
{
struct node *t;
if(f==NULL)
{
printf("\nQueue is empty");
return;
}
t=f;
printf("\ncontents of the queue");
while(t!=NULL)
{
printf("%d",t->data);
t=t->next;
}
getch();
}
```

Lab Assignment:

1.Write a program to insert 5 elements into linear queue and then display them using linked list.

2.Write a program to insert 10 integers into the linear queue and then delete them using linked list.

3.Write a program to check whether a linear queue is full of empty using linked list.

4.Write a program to arrange the items of a linear queue in descending order.

	Uttarakhand Open University, Haldwani **School of Computer Science & IT**	LABORATORY MANUAL
	PRACTICAL INSTRUCTION SHEET	
DEPTT.: Computer Science& IT	LABORATORY: Data Structure Lab (BCA-06)	SEMESTER: II

5.Write a program to display all the items of a linear queue using linked list.

Experiment No-6

IMPLEMENTATION OF TREE TRAVERSALS USING LINKED LISTS

Backgrounds:
Like a linked list, elements will be stored in *nodes*. Furthermore, a node will have to keep track of its element's immediate children. Like a linked list, nodes will point to one another in the tree--each node will point to the left and right child's node.

When there is no left or right child, of course, we'll make the corresponding pointers NULL.

Now, let's return to our original tree, but view it as if it was made up of these C treeNodeTs...

```
        -----
        |j  |
        |5  |
        |---|
        | | |
        /---\
       v     v
    -----     -----
    |f  |     |k  |
    |30 |     |13 |
    |---|     |---|
    | | |     |0| |
    /---\     ----\
   v     v        v
-----   -----      -----
|a  |   |h  |      |z  |
|100|   |50 |      |1  |
|---|   |---|      |---|
|0|0|   |0|0|      |0|0|
-----   -----      -----
```

PREAPEARED BY: Balam Singh Dafouti

While these nodes suffice to keep track of all the elements.We need a pointer to the root of the tree!

ALGORITHM:-

```
Step 1: Start the process.
Step 2: Initialize and declare variables.
Step 3: Enter the choice. Inorder / Preorder / Postorder.

Step 4: If choice is  In-order then

        a)  Traverse the left subtree in inorder.
        b)  Process the root node.
        c)  Traverse the right subtree in inorder.

Step 5: If choice is Pre-order then

        a)  Process the root node.
        b)  Traverse the left subtree in preorder.
        c)  Traverse the right subtree in preorder.

Step 6: If choice is Post-order then

        a)  Traverse the left subtree in preorder.
        b)  Traverse the right subtree in preorder.
        c)  Process the root node.
```

PROGRAM :

```
void inorder(struct treenode *sr)
{
if(sr!=NULL)
{
in(sr->lchild);
printf("\t%d",sr->da);
in(sr->rchild);
}
else
return;
}
```

```
void preorder(struct treenode *sr)
{
if(sr!=NULL)
{
printf("\t%d",sr->da);
pr(sr->lchild);
pr(sr->rchild);
}
else
return;
}

void postorder(struct treenode *sr)
{
if(sr!=NULL)
{
po(sr->lchild);
po(sr->rchild);
printf("\t%d",sr->da);
}
else
return;
}
```

Lab Assignment:

1.Write a program to print all the items of a binary tree using in-order traversal.

2. Write a program to print all the items of a binary tree using pre-order traversal.

3. Write a program to print all the items of a binary tree using post-order traversal..

4. Write a program to print the items of a binary tree in ascending order.

5.Write a program to print the greatest element of a binary tree.

Experiment No-7

QUEUE USING ARRAY

```
#include<stdio.h>
#include<conio.h>
void ins();
void del();
void dis();
int f,l,count=0;
void main()
{
int queue[10],ele,quit;
char c;
printf("\n\tprogram of queue with array");
f=l=quit=0;
do
{
printf("\n\toptions\t\tchoice\n\tinsert\t\ti\n\tdelete\t\td\n\tview\t\tv\n\texit\t\te\n\tenter the
choice:");
scanf("%c",&c);
switch(c)
{
case'i':
        printf("\n\tenter the element to be inserted:");
        scanf("%d",&ele);
        ins(queue,ele);
        break;
case'd':
        del(queue);
        break;
case'v':

        printf("\n\t*****queue*****\n\t");
        dis(queue);
        break;
case'e':
        quit=1;
```

```
}
}
while(!quit);
}
void ins(int queue[10],int ele)
{
if(l<10)
{
l++;
queue[l]=ele;
if(!f)
f=1;
count++;
}
else
printf("\n\t**queue overflow....don't insert\n");
return;
}
void dis(int queue[10])
{
int c;
for(c=f;c<l+1;c++)
printf("\t%6d",queue[c]);
printf("\n");
return;
}
void del(int queue[10])
{
int ele;
if(f)
{
ele=queue[f];
count--;
printf("\n\telement deleted=%d\n",ele);
if(f==l)
{
f=0;
l=0;
printf("\t****empty****");
```

```
}
else
f++;
}
else
{
printf("\n\tqueue underflow...don't delete");
}
return;
}
list.ele[p-1]=x;
list.la=list.la+1;
}
void pr()
{
printf("NULL-->");

for(i=0;i<list.la;i++)
printf("%d-->",list.ele[i]);
printf("NULL");

}
void del(int p)
{
int q;
if(p<0||p>list.la)
printf("\n\tposition out of range\n");
else
{
for(q=p;q<=list.la;q++)
{
list.ele[q-1]=list.ele[q];
}
list.la=list.la-1;
}
}
```

Lab Assignment:

1.Write a program to insert 5 elements into linear queue and then display them using array.

2.Write a program to insert 10 integers into the linear queue and then delete them using array.

3.Write a program to check whether a linear queue is full of empty using array.

4.Write a program to arrange the items of a linear queue in descending order using array.

5.Write a program to display all the items of a linear queue using array.

TRAVERSAL OF TREES

Backgrounds:when we want to visit each and every element of a tree, there are three different ways to do the traversal -- preorder, inorder and postorder. Remember, these traversal methods are limited to Binary trees only and not for any other tree.

Algorithm:-

```
Step-1: For the current node check whether it has a left child. If it has then go to step-2 or
          else step-3
Step-2: Repeat step-1 for this left child
Step-3: Visit (i.e printing in our case) the current node
Step-4: For the current node check whether it has a right child. If it has then go to step-5
Step-5: Repeat step-1 for this right child
```

PROGRAM FOR TRAVERSAL OF TREES

```
#include<stdio.h>
#include<alloc.h>
struct treenode
{
struct treenode *lchild;
int da;
struct treenode *rchild;
};
void ins(struct treenode **,int);
void in(struct treenode *);
void pr(struct treenode *);
void po(struct treenode *);
void main()
{
struct treenode *d;
int r,i=1,n,quit=0;
char c;
```

```
d=NULL;
printf("\n\n\tspecify the numbers of items to be inserted:");
scanf("%d",&r);
do
{
printf("\n\toptions\t\t\tchoice\n\tinsert\t\t\ti\n\tinorder\t\t\tn\n\tpreorder\t\tr\n\tpostorder\t\t
o\n\texit\t\t\te");
printf("\n\n\tenter your chioce:");
scanf("%c",&c);
switch(c)
{
case'i':
        while(i++<=r)
        {
        printf("\n\n\tenter the data:");
        scanf("%d",&n);
        ins(&d,n);
        }
        break;
case'n':
        printf("\n\tinorder traversal:\n");
        in(d);
        break;
case'r':
        printf("\n\n\tpreorder traversal:\n");
        pr(d);
        break;
case'o':
        printf("\n\n\tpostorder traversal:\n");
        po(d);
        break;
case'e':
        quit=1;
}
}
while(!quit);
}
void ins(struct treenode **sr,int n)
{
```

```
if(*sr==NULL)
{
*sr=malloc(sizeof(struct treenode));
(*sr)->lchild=NULL;
(*sr)->da=n;
(*sr)->rchild=NULL;
return;
}
else
{
if(n<(*sr)->da)
ins(&((*sr)->lchild),n);
else
ins(&((*sr)->rchild),n);
}
return;
}
void in(struct treenode *sr)
{
if(sr!=NULL)
{
in(sr->lchild);
printf("\t%d",sr->da);
in(sr->rchild);
}
else
return;
}
void pr(struct treenode *sr)
{
if(sr!=NULL)
{
printf("\t%d",sr->da);
pr(sr->lchild);
pr(sr->rchild);
}
else
return;
}
```

```
void po(struct treenode *sr)
{
if(sr!=NULL)
{
po(sr->lchild);
po(sr->rchild);
printf("\t%d",sr->da);
}
else
return;
}
```

Lab Assignment:

1.Write a program to print all the items of a binary tree using in-order traversal.

2. Write a program to print all the items of a binary tree using pre-order traversal.

3. Write a program to print all the items of a binary tree using post-order traversal..

4. Write a program to print the items of a binary tree in ascending order.

5.Write a program to print the greatest element of a binary tree.

<u>Experiment No-9</u>

IMPLEMENTATION OF HEAP SORT

Backgrounds: In this method, we will interpret thee array to be sorted as a binary tree, in a sequential representation of the binary tree, we shall have

1. The father node at location (i-2)/2 if i is not equal to zero.
2. 2. The left child at location2i+1.
3. 3. The right child at location2i2.

As the subscripts in C start from 0 to (MAXSIZE-1).

PROGRAM :

```
#include<stdio.h>
#include<conio.h>
void heap(int a[],int n);
void create_heap(int a[],int n);
void main()
{
int a[25],i,n;
clrscr();
printf("\n enter how many number:");
scanf("%d",&n);
printf("\n enter the numbers:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
heap(a,n);
printf("the sorted list is");
for(i=0;i<n;i++)
printf("\t%d",a[i]);
getch();
}
void create_heap(int a[],int n)
{
int i,j,q,key;
for(q=1;q<n;q++)
{
i=q;
key=a[q];
j=(int)(i/2);
while((i>0)&&(key>a[j]))
{
a[i]=a[j];
i=j;
j=(int)(i/2);
if(j<0)
j=0;
}
a[i]=key;
}}
void heap(int a[],int n)
{
int i,j,q,key,temp;
create_heap(a,n);
for(q=n-1;q>=1;q--)
{
temp=a[0];
a[0]=a[q];
a[q]=temp;
i=0;
key=a[0];
j=1;
if((j+1)<q)
if(a[j+1]>a[j])
j=j+1;
while((j<=(q-1))&&(a[j]>key))
{
a[i]=a[j];
i=j;
j=2*i;
if((j+1)<q)
if(a[j+1]>a[j])
j=j+1;
else if(j>n-1)
j=n-1;
a[i]=key;  }} }
```

Lab Assignment:

1.Write a program to build a MAX HEAP, and then traverse it in pre-order.

2. Write a program sort the items in ascending order using heap sort method.

3. Write a program sort the items in descending order using heap sort method.

4. Write a program to convert a MAX HEAP into MIN HEAP.

5.Write a program to print the greatest and smallest element of a HEAP.

Experiment No-10

IMPLEMENTATION OF QUICK SORT

Backgrounds: The quicksort algorithm works by partitioning the array to be sorted. And each partition is in turn sorted recursively. In partition, one of the array elements is chosen as a key value. This key value can be the first elements of an array. That is, if a is

an array then key=a[0And rest of the array elements are grouped into two partitions such that
1.One partition contains elements smaller than the key value.
2. Another partition contains element larger than the key value.

ALGORITHM:-

Quick_Sort(a,l,h)
Where
a→Represents the list of elements.
l→Represents the position of the first element in the list (only at the starting point, it's value change during the execution of the function).
h→Represents the position of the last element in the list (only at starting point the value of it's changes during the execution of the function).

```
Step 1: [Initally]
        Low=l
        High=h
        Key=a[(l+h)/2] [Middle element of the element of the list]

Step 2: Repeat through step 7 while (low<=high)
Step 3: Repeat step 4 while (a([low]<key))
Step 4: low =low+1
Step 5: Repeat step 6 while (a[high]<key)
Step 6: high =high-1
Step 7: if (low<=high)
                (2) tenp = a[low]
                (3) a[low] = a[high]
                (4) a[high]=temp
                (5) low=low+1
                (6) high=high -1

Step 8: if (l<high) Quick_Sort (a,l,high)
Step 9: if (low<h) Quick_Sort (a,low,h)
Step10: Exit.
```

PROGRAM :

```
#include<stdio.h>
#include<alloc.h>
void quicksort(int *,int,int);
int split(int *,int,int);
```

```
void main()
{
int n,i,arr[10],quit=0;
char c;
printf("\n\tenter the range of the array:");
scanf("%d",&n);
do
{
printf("\n\toption\t\tchoice\n\tinsert\t\ti\n\tquick sort\tq\n\tview\t\tv\n\texit\t\te");
printf("\n\tenter your choice:");
scanf("%c",&c);
switch(c)
{
case'i':
        for(i=0;i<n;i++)
        {
        printf("\n\tenter the element:");
        scanf("%d",&arr[i]);
        }
        break;
case'q':
        quicksort(arr,0,n-1);
        break;
case'v':
        for(i=0;i<n;i++)
        printf("%d\t",arr[i]);
        break;
case'e':
        quit=1;
}
}
while(!quit);
}
void quicksort(int a[],int lower,int upper)
{
int i;
if(upper>lower)
{
i=split(a,lower,upper);
quicksort(a,lower,i-1);
quicksort(a,i+1,upper);
}
}
int split(int a[],int lower,int upper)
{
int i,p,q,r;
p=lower+1;
q=upper;
i=a[lower];
while(q>=p)
{
while(a[p]<i)
        p++;
while(a[q]>i)
        q--;
if(q>p)
{
r=a[p];
a[p]=a[q];
a[q]=r;
}}
r=a[lower];
a[lower]=a[q];
a[q]=r;
return q;
}
```

Lab Assignment:

1.Find out the complexity of quick sort, if there are 10 item to be sorted.

2. Write a program sort the items in ascending order using quick sort method.

3. Write a program sort the items in descending order using quick sort method.

4. Explain the merits and demerits of quick sort method.

5.Explain the process of sorting the elements using quick sort method.

Experiment No-11

Graph Implementation

Backgrounds: Depth first traversal follows first a path from the starting Node to an ending Node, then another path from the start to the end, and so forth until all nodes have been visited.

Breadth First Search

Search a graph (directed or not) in breadth first; this is done by using a queue where the vertices found are stored.

Here is a brief description of the BFS algorithm:

```
bfs (Graph G)
{
    all vertices of G are first painted white

    the graph root is painted gray and put in a queue

    while the queue is not empty
    {
        a vertex u is removed from the queue

        for all white successors v of u
        {
            v is painted gray
                v is added to the queue
        }

        u is painted black
    }
}
```

And now watch it run -- click the applet to start/stop the search.

Depth First Search

The general idea is the same, but we now use a stack instead of a queue. With recursion of course, so the stack management is all done by Java.

Here is a brief description of the DFS algorithm:

```
dfs-visit (Graph G, Vertex u)
{
    the vertex u is painted gray

    for all white successors v of u
    {
        dfs-visit(G, v)
        }

        u is painted black
}


dfs (Graph G)
{
        all vertices of G are first painted white

        dfs-visit(G, root of G)
}
```

PROGRAM FOR DFS & BFS :

```
#include<stdio.h>
int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();
main()
{
int n,i,s,ch,j;
char c,dummy;
printf("ENTER THE NUMBER VERTICES ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);
scanf("%d",&a[i][j]);
}
}
printf("THE ADJACENCY MATRIX IS\n");
```

```
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf(" %d",a[i][j]);
}
printf("\n");
}

do
{
for(i=1;i<=n;i++)
vis[i]=0;
printf("\nMENU");
printf("\n1.B.F.S");
printf("\n2.D.F.S");
printf("\nENTER YOUR CHOICE");
scanf("%d",&ch);
printf("ENTER THE SOURCE VERTEX :");
scanf("%d",&s);

switch(ch)
{
case 1:bfs(s,n);
break;
case 2:
dfs(s,n);
break;
}
printf("DO U WANT TO CONTINUE(Y/N) ? ");
scanf("%c",&dummy);
scanf("%c",&c);
}while((c=='y')||(c=='Y'));
}
void bfs(int s,int n)
{
int p,i;
add(s);
vis[s]=1;
```

```
p=delete();
if(p!=0)
printf(" %d",p);
while(p!=0)
{
for(i=1;i<=n;i++)
if((a[p][i]!=0)&&(vis[i]==0))
{
add(i);
vis[i]=1;
}
p=delete();
if(p!=0)
printf(" %d ",p);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
bfs(i,n);
}
void add(int item)
{
if(rear==19)
printf("QUEUE FULL");
else
{
if(rear==-1)
{
q[++rear]=item;
front++;
}
else
q[++rear]=item;
}
}
int delete()
{
int k;
if((front>rear)||(front==-1))
return(0);
```

```
else
{
k=q[front++];
return(k);
}
}

void dfs(int s,int n)
{
int i,k;
push(s);
vis[s]=1;
k=pop();
if(k!=0)
printf(" %d ",k);
while(k!=0)
{
for(i=1;i<=n;i++)
if((a[k][i]!=0)&&(vis[i]==0))
{
push(i);
vis[i]=1;
}
k=pop();
if(k!=0)
printf(" %d ",k);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
dfs(i,n);
}
void push(int item)
{
if(top==19)
printf("Stack overflow ");
else
stack[++top]=item;
}
int pop()
```

```
{
int k;
if(top==-1)
return(0);
else
{
k=stack[top--];
return(k);
}
}
```

Lab Assignment:

1.Consider a directed graph, and traverse it using DEPTH FIRST SEARCH(DFS).

2. Consider a undirected graph, and traverse it using BREATH FIRST SEARCH(BFS).

3. Consider a directed graph, and find out it's SPPANING TREE by KRUSKAL's algo.

4. Consider a weighted & directed graph, and find out SHORTEST PATHs from any node to any other nodes by DIJKASTRA's algo.

5.Explain the necessity of minimum cost spanning tree.

Experiment No-12

DELETION IN BINARY SEARCH TREE

Lab Objectives:
After performing this lab, the students should be able to create a binary search tree.

Backgrounds:

There are basically four cases to deal with as regards the node (holding the value) to be deleted:

1. A **leaf node**
2. A **non-leaf node** with an **empty left subtree**
3. A **non-leaf node** with an **empty right subtree**
4. A **non-leaf node** with **neither of its subtrees empty**

Pre-lab:
1. Study / review how to create a tree in C.
2. How to add the node in the tree.
Experiments in this lab

Search tree in C language.
Deletion in binary search tree.

```
Node Delete(Node root, Key k)
1  if (root == null) // failed search
2    return null;
3  if (k == root.key) // successful search
4    return DeleteThis(root);
5  if (k < root.key) // k in the left branch
6    root.left = Delete(root.left, k);
7  else // k > root.key, i.e., k in the right branch
8    root.right = Delete(root.right, k);
9  return root;
Node DeleteThis(Node root)
1  if root has two children
2    p = Largest(root.left); // replace root with its immediate predecessor p
3    root.key = p.key;
4    root.left = Delete(root.left, p)
5    return root;
6  if root has only left child
7    return root.left
8  if root has only right child
9    return root.right
10  else root has no children
11    return null
Node Largest(Node root)
1  if root has no right child
2    return root
3  return Largest(root.right)
```

Lab Assignment:

1. If elements of a BST are : 10 , 20 , 5 , 8 , 20 , 30 , 60 , 8 , 2 , 3 , 90, Re-build a BST after deletion of element 10.

2. Write a program to delete a node of a BST that have no child.

3. Write a program to delete a node of a BST that have only left child.

4. Write a program to delete a node of a BST that have only right child.

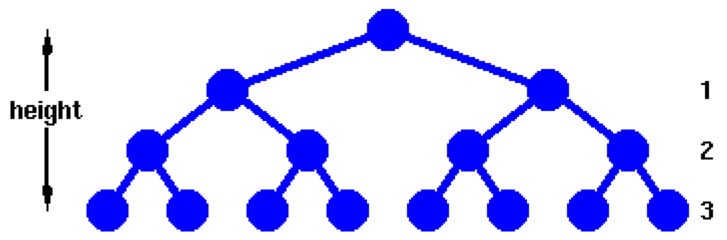
5. Write a program to delete a node of a BST that have both the child.

<u>Experiment No-13</u>

INSERTION IN BINARY SEARCH TREE

Lab Objectives:
After performing this lab, the students should be able to arrange data in tree form.

Backgrounds:
□



Pre-lab:
1. Study / review how to create a tree in C.
2. How to add the node in the tree and how to delete it.
Experiments in this lab

- Binary Search tree in C language.
- Insertion in binary search tree.

ALGORITHM:-

```
1.Struct rec*insert (struct rec * tree, long digit) Repeat steps from 2 to 14
2. if (tree = = MULL) Step from 3 to 6
3. Tree = (struct rec *) malloc (sizeof(struct rec));
4. Tree → left =tree→ right= NULL
5. Tree→ num=digit;
6. Else Step 7
7. if (digit<tree→num) tree→left= insert (tree→left, digit);
8. Else Step 9
9. if (digit< tree→num)tree→right= insert (tree→right, digit);
10. Else Step 11
11. if (digit == tree →num) step 12 to 13
12. Puts("Duplicate nodes:Program Exited");
13. Exit(0);
14. Return(tree);
15. End
```

PROGRAM FOR INSERTION IN BINARY SEARCH TREE

```
void BST::InsertNode(int x)
{
        TreeNode *parent = NULL;
        TreeNode *child;
        child = root;
        while(child != NULL)
        {
                parent = child;
                if(x <= child->getItem())
                        child = child->getLeftChild();
                else
                        child = child->getRightChild();
        }

        if(parent == NULL)
        {
                child = new TreeNode(x);
                root = child;
        }
        else if( x <= child->getItem())
        {
                child = new TreeNode(x);
                parent->setLeftChild(child);
        }
        else
        {
                child = new TreeNode(x);
                parent->setRightChild(child);
        }}
```

Lab Assignment:

1. Build a BST , if elements are : 10 , 20 , 5 , 8 , 10 , 30 , 60 , 8 , 2 , 3 , 90

2. Write a program to print the elements of a BST in ascending order.

3. Write a program to print the elements of a BST in ascending order.

4. Write a program to insert a new item into BST

5. What is the importance of binary search tree ?