

# 18CSI301L : Data Structures Using C Lab

LAB MANUAL

**Prepared and Compiled**

**by**

**Dr. Kuldeep Sharma**

**Assoc. Professor**

**&**

**Mr. Md. Zabeeulla**

**Asst. Prof**

**Academic Year: July – December 2019**

**Department of Computer Science and Engineering**

**School of Engineering and Technology**

**JAIN University**

**Jakkasnadra post, Kanakapura**

# 18 CSI301L: DATA STRUCTURES USING C LAB

# 

|  |  |
| --- | --- |
| **LIST OF PROGRAMS** | |
| **Lab**  **Exercise**  **#** | **Program to be covered** |
|  | **Demonstrating Pointers Usage**   1. Printing Memory Addresses : **Write C program to demonstrate the use of pointers by printing memory address** 2. Writing a Swap Function: **Write a C program to swap two numbers using pointers concept** 3. Allocating and Freeing Memory: **Write a C program to demonstrate the use of allocating a memory and freeing** 4. Memory Leaks and Other Problems: **Write a C program to demonstrate the memory leaks when pointers are not used properly.** |
|  | **Demonstrate Strings, User defined data types and Files in C**   1. Reading and Writing Strings : **Write a C program to demonstrate the input and output operations on strings** 2. String operations / Manipulations: **Write a C program to demonstrate the operations on strings – by writing user defined string functions.** 3. Enumerations, Structures and Union: **Write a C program to demonstrate Enumerations, Structures and Union data types.** 4. File operations: **Write a C program to demonstrate the input and output operations on files** |
|  | **Demonstrate the technique of recursion in C**  Recursion – **Write recursive function for**   1. Sum of natural numbers 2. Factorial of a given number 3. Fibonacci sequence |
|  | **Stack ADT** Implement Stack using Arrays. |
|  | **Queue ADT** Implement Queue using Arrays. |
|  | **Singly Linked List** Write a C Program to perform following operations on Singly  Linked List ADT :  i. Create ii. Insert iii. Delete iv. Display |
|  | **Doubly Linked List** Write a C Program to perform following operations on  Doubly Linked List ADT :  i. Create ii. Insert iii. Delete iv. Display |
|  | **Circular Linked List** Write a C Program to perform following operations on  Circular Linked List ADT :  i. Create ii. Insert iii. Delete iv. Display |
|  | Implement Stack using List |
|  | Implement Queue using List |
|  | Implement Binary Search Tree using List.  12. i)Implement a simple heap ii) Implement Priority Queue using heap |
|  |  |
|  |  |

**Course Objective:**

The main objective of this laboratory is to give an insight of Data Structures and demonstrate various different data structures using static and dynamic memory allocation.

**Prerequisite:**

Students should have a prerequisite knowledge of programming concepts in C.

**Outcome of the Course:**

After the completion of this Laboratory course, student will be well versed with the concepts of Data Structures and its usage. The knowledge acquired in laboratory can be applied to develop a mini/major projects.

**Demonstrating Pointers Usage**

1. Printing Memory Addresses : **Write C program to demonstrate the use of pointers by printing memory address**
2. Writing a Swap Function: **Write a C program to swap two numbers using pointers concept**
3. Allocating and Freeing Memory: **Write a C program to demonstrate the use of allocating a memory and freeing**
4. Memory Leaks and Other Problems: **Write a C program to demonstrate the memory leaks when pointers are not used properly.**

**Printing Memory Addresses :** Write C program to demonstrate the use of pointers by

printing memory address

#include<stdio.h>

int main()

{

int \*p;

int a=10;

p=&a;

printf("\n address of a=%p",&a);

printf("\n address of p=%p",&p);

printf("\n value of a=%d",a);

printf("\n value of pointer p=%d",\*p);

printf("\n value at p=%p",p);

return 0;

}

**Writing a Swap Function:** **Write a C program to swap two numbers using pointers concept**

#include<stdio.h>

int main()

{

int a,b;

printf("\n enter the value of a= ");

scanf("%d",&a);

printf("\n enter the value of b=");

scanf("%d",&b);

printf("\n values of a and b before swaping=%d and %d",a,b);

swap(&a,&b);

return 0;

}

void swap(int \*x,int \*y)

{

int temp;

temp=\*x;

\*x=\*y;

\*y=temp;

printf("\n values of a and b after swaping=%d and %d",\*x,\*y);

}

**Allocating and Freeing Memory: Write a C program to demonstrate the use of allocating a memory and freeing.**

#include<stdio.h>

void circle(float k,float \*x,float \*y);

int main()

{

float r,a,p;

printf("\n enter the value for radius");

scanf("%f",&r);

circle(r,&a,&p);

printf("\n area=%f",a);

printf("\n perimeter=%f",p);

return 0;

}

void circle(float k,float \*x,float \*y)

{

\*x=(3.14)\*k\*k;

\*y=2\*3.14\*k;

}

**Memory Leaks and Other Problems:** Write a C program to demonstrate the memory leaks when pointers are not used properly.

#include<stdio.h>

int main()

{

int n,i,\*ptr;

printf("\n ENTER THE NUMBER OF USN WANT TO ENTER->");

scanf("%d",&n);

ptr=(int \*)malloc(n\*sizeof(int));

if(ptr==NULL)

{

printf("\n YOUR MEMORY IS FULL");

}

printf("\n ENTER THE USN->");

for(i=0;i<n;i++)

{

scanf("%d",ptr+i);

}

printf("\n THE USN ENTERED ARE->");

for(i=0;i<n;i++)

{

printf("%d\n",\*(ptr+i));

}

return 0;

}

**Demonstrate Strings, User defined data types and Files in C**

1. Reading and Writing Strings : **Write a C program to demonstrate the input and output operations on strings**
2. String operations / Manipulations: **Write a C program to demonstrate the operations on strings – by writing user defined string functions.**
3. Enumerations, Structures and Union: **Write a C program to demonstrate Enumerations, Structures and Union data types.**
4. File operations: **Write a C program to demonstrate the input and output operations on files**

A string is generally understood as a [data type](https://en.wikipedia.org/wiki/Data_type) and is often implemented as an [array data structure](https://en.wikipedia.org/wiki/Array_data_structure) of [bytes](https://en.wikipedia.org/wiki/Byte) (or [words](https://en.wikipedia.org/wiki/Word_(computer_architecture))) that stores a sequence of elements, typically characters, using some [character encoding](https://en.wikipedia.org/wiki/Character_encoding). A string may also denote more general [arrays](https://en.wikipedia.org/wiki/Array_data_type) or other sequence (or [list](https://en.wikipedia.org/wiki/List_(abstract_data_type))) data types and structures. Depending on programming language and precise data type used, a [variable](https://en.wikipedia.org/wiki/Variable_(programming)) declared to be a string may either cause storage in memory to be statically allocated for a predetermined maximum length or employ [dynamic allocation](https://en.wikipedia.org/wiki/Dynamic_allocation) to allow it to hold a variable number of elements

**2: Demonstrate Strings, User defined data types and Files in C**

Reading and Writing Strings :

**Write a C program to demonstrate the input and output operations on strings.**

#include<stdio.h>

#include<string.h>

int strl(char a[100])

{

int n=0,i;

for (i=0;a[i]!='\0';i++)

n++;

return n;

}

void strc(char a[100],char b[100])

{

int n,i,j;

n=strlen(a);

for(i=n,j=0;b[j]!='\0';i++,j++)

a[i]=b[j];

a[i]='\0';

puts(a);

}

int strcm(char a[100],char b[100])

{

int i,flag=0;

for(i=0;(a[i]!='\0')&&(b[i]='\0');i++)

{

if(b[i]==a[i]&&a[i]!='\0'&&b[i]!='\0')

flag=1;

else

flag=0;

return flag;

}

}

void strcp(char b[100])

{

char c[100];

int i;

for(i=0;b[i]!='\0';i++)

c[i]=b[i];

c[i]='\0';

puts (c);

}

void strr(char a[100])

{

char b[100];

int i,j,n;

n=strl(a);

for(i=n-1,j=0;i>=0;i--,j++)

b[j]=a[i];

b[j]='\0';

puts (b);

}

int strst(char a[100],char b[100])

{

int i,j=0,n,flag=0;

for(i=0;a[i]!='\0';i++)

{

n=i;

for(j=0;b[j]!='\0';j++)

{

if (a[n]==b[j])

{

flag=1;

n++;

}

else

{

flag=0;

break;

}

if (flag)

break;

}

}

return flag;

}

void main()

{

char a[100],b[100];

printf("enter the string");

gets (a);

gets (b);

printf("string length=%d",strl(a));

printf("string compare=%s\n",strcm(a,b)?"true":"false");

printf("string cat=");

strc(a,b);

printf("string sub string=%d",strst(a,b));

printf("string rev=");

strr(a);

}

String operations / Manipulations:

**Write a C program to demonstrate the operations on strings – by writing user defined string functions.**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main()

{

char a[10],b[10],s[10];

int x,y,z;

printf("\n enter the 1st string");

gets(a);

printf("\n enter the second string");

gets(b);

x=strlen(a);

y=strlen(b);

z=strcmp(a,b);

strcpy(a,b);

strrev(b);

printf("\n string length =%d",x);

printf("\n string length =%d",y);

printf("\n after comparing =%d",z);

printf("\n string rev =%s",a);

return 0;

}

Enumerations, Structures and Union :

**Write a C program to demonstrate Enumerations,Structures and Union data types.**

**Using structure**

#include<stdio.h>

struct student

{

float p,c,m;

};

int main()

{

int n,i;

struct student s1[100];

printf("\n enter number of student");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

printf("\n enter the marks for physics=");

scanf("%f",&s1[i].p);

printf("\n enter the marks for chemistry=");

scanf("%f",&s1[i].c);

printf("\n enter the marks for maths=");

scanf("%f",&s1[i].m);

}

printf("\n marks are");

for(i=1;i<=n;i++)

printf("%f\n%f\n%f\n",s1[i].p,s1[i].c,s1[i].m);

return 0;

}

**Using UNION**

#include<stdio.h>

#include<string.h>

union book

{

int a;

char name ;

//char author[20];

float price;

};

int main()

{

union book b1;

printf("\n enter the integer value");

scanf("%d",&b1.a);

printf("\n%d",b1.a);

printf("\n enter the name of book");

scanf("%c",&b1.name);

printf(" %c",b1.name);

printf("\n enter the price");

scanf(" %f",&b1.price);

printf("\n %f",b1.price);

return 0;

}

**File operations:** Write a C program to demonstrate the input and output operations on files.

#include<stdio.h>

int n;

struct student

{

int usn;

char name[10];

int sem;

};

int main()

{

struct student s1[n];

FILE \*fp;

fp=fopen("student.txt","w");

int i;

printf("\n enter the number of student");

scanf("\n %d",&n);

for(i=0;i<n;i++)

{

printf("\n enter the details");

scanf("%d%s%d",&s1[i].usn,s1[i].name,&s1[i].sem);

fprintf(fp,"%d%s%d",s1[i].usn,s1[i].name,s1[i].sem);

}

fclose(fp);

fp=fopen("student.txt","r");

for(i=0;i<n;i++)

{

fscanf(fp,"%d%s%d",&s1[i].usn,s1[i].name,&s1[i].sem);

printf("%d%s%d",s1[i].usn,s1[i].name,s1[i].sem);

}

fclose(fp);

return 0;

}

**Demonstrate the technique of recursion in C**

Recursion – **Write recursive function for**

1. Sum of natural numbers
2. Factorial of a given number
3. Fibonacci sequence

Many programming languages implement recursion by means of **stacks**. Generally, whenever a function (**caller**) calls another function (**callee**) or itself as callee, the caller function transfers execution control to the callee. This transfer process may also involve some data to be passed from the caller to the callee.

This implies, the caller function has to suspend its execution temporarily and resume later when the execution control returns from the callee function. Here, the caller function needs to start exactly from the point of execution where it puts itself on hold. It also needs the exact same data values it was working on. For this purpose, an activation record (or stack frame) is created for the caller function.

#include<stdio.h>

#include<conio.h>

int fib(int n)

{

int x;

if (n==1)

return 0;

else if (n==2)

return 1;

x=fib(n-1) + fib(n-2);

return (x);

}

int sum(int n)

{

if (n<=1)

return 1;

return n+sum(n-1);

}

int factorial(int n)

{

if (n<=1)

return 1;

return n\*factorial(n-1);

}

void main()

{

int a[20],n, i, sum1, fact;

printf("enter the value for n:\n");

scanf("%d", &n);

for (i=1 ;i<= n;i++)

a[i]=fib(i);

printf("\nfibonacci series\n ");

for (i=1;i<= n;i++)

printf (" -->%d", a[i]);

printf("\n sum of natural numbers = ");

sum1 = sum(n);

printf("%d", sum1);

printf("\n factorial of %d = ",n);

fact = factorial(n);

printf("%d", fact);

getch();

}

**Stack ADT** Implement Stack using Arrays.

A stack is a linear data structure in which an element can be inserted or deleted only at one end of the list. A stack works on the principle of last in first out and is also known as a Last-In-First-Out (LIFO) list. Operations on a stack

* push
  + Insert , a new object on top, stack
  + Overflow Condition
* Pop
  + Delete, top object, from the stack
  + Underflow Condition
* A stack is an abstract data type (ADT) that supports
* Two main methods:
  + push(*element): Inserts object o onto top of stack*
    - *Input: Object; Output: none*
  + pop(): Removes the top object of stack and returns it; if stack is empty an error occurs
    - *Input: none; Output: Object*
* When stacks are represented as arrays, a variable named **Top** is used to point to the top element of the stack with **MAX** is the maximum size of the stack
* Initially the value of Top = -1 ( or 0) to indicate an empty stack.
* Push an element onto the stack,
  + Top is incremented by 1 and the element is pushed at that position.
  + When Top reaches MAX-1 and an attempt is made to push a new element, then stack overflows.
* Pop (or remove) an element from the stack,
  + the element on the top of the stack is assigned to a local variable and then Top is decremented by 1.
  + When the value of Top is equal to -1 and an attempt is made to pop an element, the stack underflows.
* Before inserting/deleting any element onto/from the stack, it is necessary to test the condition of overflow/underflow

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#define MAX 6

int menu()

{

int ch;

printf("\n -----------\*\*\*\*\*\*\*\*\*\*-------------\n");

printf("\n ....Stack operations using ARRAY.... ");

printf("\n -----------\*\*\*\*\*\*\*\*\*\*-------------\n");

printf("\n 1. Push ");

printf("\n 2. Pop ");

printf("\n 3. Display");

printf("\n 4. Quit ");

printf("\n Enter your choice: ");

scanf("%d", &ch);

return ch;

}

void display(int stack[],int \*top)

{

int i;

if((\*top) == 0)

{

printf("\n\nStack empty..");

return;

}

else

{

printf("\n\nElements in stack:");

for(i = (\*top)-1; i >= 0 ; i--)

printf("\n%d", stack[i]);

}

}

void pop(int stack[],int \*top)

{

if(\*top == 0)

{

printf("\n\nStack Underflow..");

return;

}

else

printf("\n\npopped element is: %d ", stack[--(\*top)]);

}

void push(int stack[],int \*top)

{

int data;

if(\*top == MAX)

{

printf("\n\nStack Overflow..");

return;

}

else

{

printf("\n\nEnter data: ");

scanf("%d", &data);

stack[(\*top)] = data;

\*top = \*top + 1;

printf("\n\nData Pushed into the stack");

}

}

void main()

{

int ch;

int stack[MAX];

int (\*top) = 0;

do

{

ch = menu();

switch(ch)

{

case 1:

push(stack,&top);

break;

case 2:

pop(stack,&top);

break;

case 3:

display(stack,&top);

break;

case 4:

exit(0);

}

getch();

} while(1);

}

**Queue ADT** Implement Queue using Arrays.

A queue is another special kind of list, where items are

* Inserted at one end called the rear
* Deleted at the other end called the front.

Operations on queues:

* enqueue: which inserts an element at the end of the queue.
* dequeue: which deletes an element at the start of the queue

insertQ(): inserts an element at the end of queue Q.

deleteQ(): deletes the first element of Q.

displayQ(): displays the elements in the queue.

#include <stdio.h>

#include <conio.h>

# define MAX 6

void insertQ(int Q[], int \*front, int \*rear)

{

int data;

if((\*rear) == MAX)

{

printf("\n Linear Queue is full");

return;

}

printf("\n Enter data: ");

scanf("%d", &data);

Q[\*rear] = data;

(\*rear)++;

printf("\n Data Inserted in the Queue ");

}

void deleteQ(int Q[], int \*front, int \*rear)

{

if(\*rear == \*front)

{

printf("\n\n Queue is Empty..");

\*rear = \*front = 0;

return;

}

else

{

printf("\n Deleted element from Queue is %d", Q[\*front]);

(\*front)++;

if( \*front == \*rear)

\*front = \*rear =0;

}

}

void displayQ(int Q[], int \*front, int \*rear)

{

int i;

if(\*front == \*rear)

{

printf("\n\n\t Queue is Empty");

return;

}

else

{

printf("\n Elements in Queue are: ");

for(i = \*front; i < \*rear; i++)

{

printf("%d\t", Q[i]);

}

}

}

int menu()

{

int ch;

printf("\n \tQueue operations using ARRAY..");

printf("\n -----------\*\*\*\*\*\*\*\*\*\*-------------\n");

printf("\n 1. Insert ");

printf("\n 2. Delete ");

printf("\n 3. Display");

printf("\n 4. Quit ");

printf("\n Enter your choice: ");

scanf("%d", &ch);

return ch;

}

void main()

{

int ch;

int Q[MAX];

int \*front=0, \*rear=0;

do

{

ch = menu();

switch(ch)

{

case 1:

insertQ(Q, &front, &rear);

break;

case 2:

deleteQ(Q, &front, &rear);

break;

case 3:

displayQ(Q, &front, &rear);

break;

case 4: exit(0);

}

} while(1);

}

**Linked list**

The size requirement need not be known at compile time. A linked list is a data structure that is used to model such a dynamic list of data items

ARRAY is sequential mapping, elements are fixed distance apart. makes insertion or deletion at any arbitrary position in an array a costly operation

Linked List not necessary that the elements be at a fixed distance apart an element is required to be linked with a previous element of the list done by storing the address of the next element

A linked list is a linear collection of data elements called nodes in which linear representation is given by links from one node to the next node.

Different from array, data elements of linked list are generally not lined in consecutive memory space; instead they are dispersed in various locations

Linked list element (node) is user defined structure data type, typically contains two parts

* Information/data field
* One/Two pointers, holding the address of next

Types of linked list:

1. Singly linked list
2. Circular linked list
3. Doubly linked list

**Singly Linked List** Write a C Program to perform following operations on Singly

Linked List ADT :

i. Create ii. Insert iii. Delete iv. Display

# include <stdio.h>

# include <conio.h>

# include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

typedef struct node NODE;

NODE \*start = NULL;

int menu()

{

int ch;

system("cls");

printf("\n 1.Create a list ");

printf("\n--------------------------");

printf("\n 2.Insert a node at beginning ");

printf("\n 3.Insert a node at end");

printf("\n--------------------------");

printf("\n 4.Delete a node from beginning");

printf("\n 5.Delete a node from Last");

printf("\n--------------------------");

printf("\n 6.Displaying the list");

printf("\n--------------------------");

printf("\n 7.Exit ");

printf("\n\n Enter your choice: ");

scanf("%d",&ch);

return ch;

}

NODE\* getnode()

{

NODE \* newnode;

newnode = (NODE \*) malloc(sizeof(NODE));

printf("\n Enter data: ");

scanf("%d", &newnode -> data);

newnode -> next = NULL;

return newnode;

}

void createlist(int n)

{

int i;

NODE \*newnode, \*temp;

for(i = 0; i < n; i++)

{

newnode = getnode();

if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL)

temp = temp -> next;

temp -> next = newnode;

}

}

}

void display()

{

NODE \*temp;

temp = start;

printf("\n The contents of List (Left to Right): \n");

if(start == NULL)

{

printf("\n Empty List");

return;

}

else

{

while(temp != NULL)

{

printf("%d-->", temp -> data);

temp = temp -> next;

}

}

}

void insert\_at\_beg()

{

NODE \*newnode;

newnode = getnode();

if(start == NULL)

{

start = newnode;

}

else

{

newnode -> next = start;

start = newnode;

}

}

void insert\_at\_end()

{

NODE \*newnode, \*temp;

newnode = getnode();

if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL)

temp = temp -> next;

temp -> next = newnode;

}

}

void delete\_at\_beg()

{

NODE \*temp;

if(start == NULL)

{

printf("\n No nodes are exist..");

return ;

}

else

{

temp = start;

start = temp -> next;

printf("\n Node deleted %d", temp->data);

free(temp);

}

}

void delete\_at\_last()

{

NODE \*temp, \*prev;

if(start == NULL)

{

printf("\n Empty List..");

return ;

}

else

{

temp = start;

prev = start;

while(temp -> next != NULL)

{

prev = temp;

temp = temp -> next;

}

prev -> next = NULL;

printf("\n Node deleted %d", temp->data);

free(temp);

}

}

void main(void)

{

int ch, n;

while(1)

{

ch = menu();

switch(ch)

{

case 1:

if(start == NULL)

{

printf("\n Number of nodes you want to create: ");

scanf("%d", &n);

createlist(n);

printf("\n List created..");

}

else

printf("\n List is already created..");

break;

case 2:

insert\_at\_beg();

break;

case 3:

insert\_at\_end();

break;

case 4:

delete\_at\_beg();

break;

case 5:

delete\_at\_last();

break;

case 6:

display();

break;

case 7:

exit(0);

}

getch();

}

}

**Doubly Linked List** Write a C Program to perform following operations on

Doubly Linked List ADT :

i. Create ii. Insert iii. Delete iv. Display

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

struct dlinklist

{

struct dlinklist \*left;

int data;

struct dlinklist \*right;

};

typedef struct dlinklist NODE;

NODE \*start = NULL;

NODE\* getnode()

{

NODE \* newnode;

newnode = (node \*) malloc(sizeof(node));

printf("\n Enter data: ");

scanf("%d", &newnode -> data);

newnode -> left = NULL;

newnode -> right = NULL;

return newnode;

}

int menu()

{

int ch;

clrscr();

printf("\n 1.Create");

printf("\n------------------------------");

printf("\n 2. Insert a node at beginning ");

printf("\n 3. Insert a node at end");

printf("\n------------------------------");

printf("\n 4. Delete a node from beginning");

printf("\n 5. Delete a node from Last");

printf("\n------------------------------");

printf("\n 6. Traverse the list from Left to Right ");

printf("\n 7. Traverse the list from Right to Left ");

printf("\n------------------------------");

printf("\n 8.Exit");

printf("\n\n Enter your choice: ");

scanf("%d", &ch);

return ch;

}

void createlist(int n)

{

int i;

NODE \*newnode, \*temp;

for(i = 0; i < n; i++)

{

newnode = getnode();

if(start == NULL)

start = newnode;

else

{

temp = start;

while(temp -> right)

temp = temp -> right;

temp -> right = newnode;

newnode -> left = temp;

}

}

}

void display\_left\_to\_right()

{

NODE \*temp;

temp = start;

printf("\n The contents of List: ");

if(start == NULL )

printf("\n Empty List");

else

{

while(temp != NULL)

{

printf("\t %d ", temp -> data);

temp = temp -> right;

}

}

}

void display\_right\_to\_left()

{

NODE \*temp;

temp = start;

printf("\n The contents of List: ");

if(start == NULL)

printf("\n Empty List");

else

{

while(temp -> right != NULL)

temp = temp -> right;

}

while(temp != NULL)

{

printf("\t%d", temp -> data);

temp = temp -> left;

}

}

void dll\_insert\_beg()

{

NODE \*newnode;

newnode = getnode();

if(start == NULL)

start = newnode;

else

{

newnode -> right = start;

start -> left = newnode;

start = newnode;

}

}

void dll\_insert\_end()

{

NODE \*newnode, \*temp;

newnode = getnode();

if(start == NULL)

start = newnode;

else

{

temp = start;

while(temp -> right != NULL)

temp = temp -> right;

temp -> right = newnode;

newnode -> left = temp;

}

}

void dll\_delete\_beg()

{

NODE \*temp;

if(start == NULL)

{

printf("\n Empty list");

getch();

return ;

}

else

{

temp = start;

start = start -> right;

start -> left = NULL;

free(temp);

}

}

void dll\_delete\_last()

{

NODE \*temp;

if(start == NULL)

{

printf("\n Empty list");

getch();

return ;

}

else

{

temp = start;

while(temp -> right != NULL)

temp = temp -> right;

temp -> left -> right = NULL;

free(temp);

temp = NULL;

}

}

void main(void)

{

int ch,n;

system("sys");

while(1)

{

ch = menu();

switch( ch)

{

case 1 :

printf("\n Enter Number of nodes to create: ");

scanf("%d", &n);

createlist(n);

printf("\n List created..");

break;

case 2 :

dll\_insert\_beg();

break;

case 3 :

dll\_insert\_end();

break;

case 4 :

dll\_delete\_beg();

break;

case 5 :

dll\_delete\_last();

break;

case 6 :

display\_left\_to\_right();

break;

case 7 :

display\_right\_to\_left();

break;

case 8:

exit(0);

}

getch();

}

}

**Circular Linked List** Write a C Program to perform following operations on

Circular Linked List ADT :

i. Create ii. Insert iii. Delete iv. Display

# include <stdio.h>

# include <conio.h>

# include <stdlib.h>

struct cslinklist

{

int data;

struct cslinklist \*next;

};

typedef struct cslinklist NODE;

NODE \*start = NULL;

int nodectr;

NODE\* getnode()

{

node \* newnode;

newnode = (node \*) malloc(sizeof(node));

printf("\n Enter data: ");

scanf("%d", &newnode -> data);

newnode -> next = NULL;

return newnode;

}

int menu()

{

int ch;

system("sys");

printf("\n 1. Create a list ");

printf("\n\n--------------------------");

printf("\n 2. Insert a node at beginning ");

printf("\n 3. Insert a node at end");

printf("\n\n--------------------------");

printf("\n 5. Delete a node from beginning");

printf("\n 6. Delete a node from Last");

printf("\n\n--------------------------");

printf("\n 8. Display the list");

printf("\n 9. Exit");

printf("\n\n--------------------------");

printf("\n Enter your choice: ");

scanf("%d", &ch);

return ch;

}

void createlist(int n)

{

int i;

NODE \*newnode,\*temp;

nodectr = n;

for(i = 0; i < n ; i++)

{

newnode = getnode();

if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL)

temp = temp -> next;

temp -> next = newnode;

}

}

newnode ->next = start; /\* last node is pointing to starting node \*/

}

void display()

{

NODE \*temp;

temp = start;

printf("\n The contents of List (Left to Right): ");

if(start == NULL )

printf("\n Empty List");

else

{

do

{

printf("\t %d ", temp -> data);

temp = temp -> next;

} while(temp != start);

printf(" X ");

}

}

void cll\_insert\_beg()

{

NODE \*newnode, \*last;

newnode = getnode();

if(start == NULL)

{

start = newnode;

newnode -> next = start;

}

else

{

last = start;

while(last -> next != start)

last= last -> next;

newnode -> next = start;

start = newnode;

last -> next = start;

}

printf("\n Node inserted at beginning..");

nodectr++;

}

void cll\_insert\_end()

{

NODE \*newnode, \*temp;

newnode = getnode();

if(start == NULL )

{

start = newnode;

newnode -> next = start;

}

else

{

temp = start;

while(temp -> next != start)

temp = temp -> next;

temp -> next = newnode;

newnode -> next = start;

}

printf("\n Node inserted at end..");

nodectr++;

}

void cll\_delete\_beg()

{

NODE \*temp, \*last;

if(start == NULL)

{

printf("\n No nodes exist..");

getch();

return ;

}

else

{

last = temp = start;

while(last -> next != start)

last= last -> next;

start = start -> next;

last -> next = start;

free(temp);

nodectr--;

printf("\n Node deleted..");

if(nodectr == 0)

start = NULL;

}

}

void cll\_delete\_last()

{

NODE \*temp,\*prev;

if(start == NULL)

{

printf("\n No nodes exist..");

getch();

return ;

}

else

{

temp = start;

prev = start;

while(temp -> next != start)

{

prev = temp;

temp = temp -> next;

}

prev -> next = start;

free(temp);

nodectr--;

if(nodectr == 0)

start = NULL;

printf("\n Node deleted..");

}

}

void main(void)

{

int result;

int ch, n;

system ("sys");

while(1)

{

ch = menu();

switch(ch)

{

case 1 :

if(start == NULL)

{

printf("\n Enter Number of nodes to create: ");

scanf("%d", &n);

createlist(n);

printf("\nList created..");

}

else

printf("\n List is already Exist..");

break;

case 2 :

cll\_insert\_beg();

break;

case 3 :

cll\_insert\_end();

break;

case 4 :

cll\_delete\_beg();

break;

case 5 :

cll\_delete\_last();

break;

case 6 :

display();

break;

case 7 :

exit(0);

}

getch();

}

}

Implement **Stack** using List

# include <stdio.h>

# include <conio.h>

# include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

typedef struct node NODE;

NODE \*start = NULL;

int menu()

{

int ch;

system("cls");

printf("\n 1.Create a list ");

printf("\n--------------------------");

printf("\n 2. PUSH ");

printf("\n 3. POP");

printf("\n 4. Displaying the list");

printf("\n 5. Quit");

printf("\n--------------------------");

printf("\n\n Enter your choice: ");

scanf("%d",&ch);

return ch;

}

NODE\* getnode()

{

NODE \* newnode;

newnode = (NODE \*) malloc(sizeof(NODE));

printf("\n Enter data: ");

scanf("%d", &newnode -> data);

newnode -> next = NULL;

return newnode;

}

void createlist(int n)

{

int i;

NODE \*newnode, \*temp;

for(i = 0; i < n; i++)

{

newnode = getnode();

if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL)

temp = temp -> next;

temp -> next = newnode;

}

}

}

void display()

{

NODE \*temp;

temp = start;

printf("\n The contents of List (Left to Right): \n");

if(start == NULL)

{

printf("\n Empty List");

return;

}

else

{

while(temp != NULL)

{

printf("%d-->", temp -> data);

temp = temp -> next;

}

}

}

void push()

{

NODE \*newnode;

newnode = getnode();

if(start == NULL)

{

start = newnode;

}

else

{

newnode -> next = start;

start = newnode;

}

}

void pop()

{

NODE \*temp;

if(start == NULL)

{

printf("\n No nodes are exist..");

return ;

}

else

{

temp = start;

start = temp -> next;

printf("\n Node deleted %d", temp->data);

free(temp);

}

}

void main(void)

{

int ch, n;

while(1)

{

ch = menu();

switch(ch)

{

case 1:

if(start == NULL)

{

printf("\n Number of nodes you want to create: ");

scanf("%d", &n);

createlist(n);

printf("\n List created..");

}

else

printf("\n List is already created..");

break;

case 2:

push();

break;

case 3:

pop();

break;

case 4:

display();

break;

case 5:

exit(0);

}

getch();

}

}

Implement **Queue** using List

# include <stdio.h>

# include <conio.h>

# include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

typedef struct node NODE;

NODE \*start = NULL;

int menu()

{

int ch;

system("cls");

printf("\n 1.Create a list ");

printf("\n--------------------------");

printf("\n 2.Insert Queue ");

printf("\n 3.Delete Queue");

printf("\n 4.Displaying the list");

printf("\n 5.Exit ");

printf("\n--------------------------");

printf("\n\n Enter your choice: ");

scanf("%d",&ch);

return ch;

}

NODE\* getnode()

{

NODE \* newnode;

newnode = (NODE \*) malloc(sizeof(NODE));

printf("\n Enter data: ");

scanf("%d", &newnode -> data);

newnode -> next = NULL;

return newnode;

}

void createlist(int n)

{

int i;

NODE \*newnode, \*temp;

for(i = 0; i < n; i++)

{

newnode = getnode();

if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL)

temp = temp -> next;

temp -> next = newnode;

}

}

}

void display()

{

NODE \*temp;

temp = start;

printf("\n The contents of List (Left to Right): \n");

if(start == NULL)

{

printf("\n Empty List");

return;

}

else

{

while(temp != NULL)

{

printf("%d-->", temp -> data);

temp = temp -> next;

}

}

}

void insertQ()

{

NODE \*newnode, \*temp;

newnode = getnode();

if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL)

temp = temp -> next;

temp -> next = newnode;

}

}

void deleteQ()

{

NODE \*temp;

if(start == NULL)

{

printf("\n No nodes are exist..");

return ;

}

else

{

temp = start;

start = temp -> next;

printf("\n Node deleted %d", temp->data);

free(temp);

}

}

void main(void)

{

int ch, n;

while(1)

{

ch = menu();

switch(ch)

{

case 1:

if(start == NULL)

{

printf("\n Number of nodes you want to create: ");

scanf("%d", &n);

createlist(n);

printf("\n List created..");

}

else

printf("\n List is already created..");

break;

case 2:

insertQ();

break;

case 3:

deleteQ();

break;

case 4:

display();

break;

case 5:

exit(0);

break;

}

getch();

}

}

Implement **Binary Search Tree** using List

A binary search tree (BST), also known as an ordered binary tree, is a node-based data structure

in which each node has no more than two child nodes. Each child must either be a leaf node or

the root of another binary search tree. The left sub-tree contains only nodes with value less

than the parent node; the right sub-tree contains only nodes with value greater than or equal

to the parent node.

The BST data structure is the basis for a number of highly efficient sorting and searching

algorithms, and it can be used to construct more abstract data structures including sets,

multisets, and associative arrays.

**Tree Traversal:**

Displaying (or) visiting order of nodes in a tree is called as Tree Traversal.

There are three types tree traversals:

1. In - Order Traversal
2. Pre - Order Traversal
3. Post - Order Traversal

**1) In-Order Traversal(Left, Root, Right)**

In In-Order traversal, the left child node is visited first, then the root node is visited and later we

go for visiting right child node. This in-order traversal is applicable for every root node of all

sub-trees in the tree.

i.e

a) Traverse the left subtree in inorder

b) Process the root Node

c) Traverse the right subtree in inorder

**2) Pre-Order Traversal( Root,Left, Right)**

In Pre-Order traversal, the root node is visited first, then its left child and later its right child.

This pre-order traversal is applicable for every root node of all sub-trees in the tree.

 i. e.

a) Process the root Node

b) Traverse the left subtree in preorder

c) Traverse the right subtree in preorder

**3) Post-Order Traversal( Left, Right, Root)**

In Post-Order traversal, left child node is visited first, then its right child and then its root node.

This is recursively performed until the right most node is visited.

i.e

a) Traverse the left subtree in postorder

b) Traverse the right subtree in Postorder

c) Process the root Node

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct BST

{

int info;

struct BST \*llink,\*rlink;

};

typedef struct BST node;

node \*create(node \*root,int ele)

{

node \*temp,\*prev,\*cur;

temp=(node \*)malloc(sizeof(node));

temp->info=ele;

temp->llink=NULL;

temp->rlink=NULL;

if(root==NULL)

return temp;

prev=NULL;

cur=root;

while(cur!=NULL)

{

prev=cur;

if(ele<cur->info)

cur=cur->llink;

else

cur=cur->rlink;

}

if(ele<prev->info)

prev->llink=temp;

else

prev->rlink=temp;

return root;

}

void preorder(node \*root)

{

if(root==NULL)

return;

printf("%d ",root->info);

preorder(root->llink);

preorder(root->rlink);

}

void inorder(node \*root)

{

if(root==NULL)

return;

inorder(root->llink);

printf("%d ",root->info);

inorder(root->rlink);

}

void postorder(node \*root)

{

if(root==NULL)

return;

postorder(root->llink);

postorder(root->rlink);

printf("%d ",root->info);

}

int main()

{

node \*root=NULL;

int ch,ele;

while(1)

{

printf("\n 1: Create 2: Preorder 3:Inorder 4: Postorder 5:exit:");

printf("\n Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\n Enter the element to be inserted:");

scanf("%d",&ele);

root=create(root,ele);

break;

case 2:

preorder(root);

break;

case 3:

inorder(root);

break;

case 4:

postorder(root);

break;

case 5:

exit(0);

}

}

getch();

return 0;

}

## Lab exercises12:

* 1. Implement a simple heap ii) Implement Priority Queue using heap

## Implement a simple Heap

#include <stdio.h> int array[100], n; void main()

{

int choice, num;

n = 0;/\*Represents number of nodes in the heap\*/ while(1)

{

printf("1.Insert the element \n");

printf("2.Delete the element \n"); printf("3.Display all elements \n");

printf("4.Quit \n"); printf("Enter your choice : "); scanf("%d", &choice); switch(choice)

{

case 1:

printf("Enter the element to be inserted to the list : "); scanf("%d", &num);

insert(num, n); n = n + 1; break;

case 2:

printf("Enter the elements to be deleted from the list: "); scanf("%d", &num);

delete(num); break;

case 3:

display(); break;

case 4:

exit(0); default:

printf("Invalid choice \n");

}/\*End of switch \*/

}/\*End of while \*/

}/\*End of main()\*/

display()

{

int i;

if (n == 0)

{

printf("Heap is empty \n"); return;

}

for (i = 0; i < n; i++) printf("%d ", array[i]);

printf("\n");

}/\*End of display()\*/

insert(int num, int location)

{

int parentnode; while (location > 0)

{

parentnode =(location - 1)/2; if (num <= array[parentnode])

{

array[location] = num; return;

}

array[location] = array[parentnode]; location = parentnode;

}/\*End of while\*/

array[0] = num; /\*assign number to the root node \*/

}/\*End of insert()\*/

delete(int num)

{

int left, right, i, temp, parentnode;

for (i = 0; i < num; i++) { if (num == array[i])

break;

}

if (num != array[i])

{

printf("%d not found in heap list\n", num); return;

}

array[i] = array[n - 1]; n = n - 1;

parentnode =(i - 1) / 2; /\*find parentnode of node i \*/ if (array[i] > array[parentnode])

{

insert(array[i], i); return;

}

left = 2 \* i + 1; /\*left child of i\*/ right = 2 \* i + 2; /\* right child of i\*/ while (right < n)

{

if (array[i] >= array[left] && array[i] >= array[right]) return;

if (array[right] <= array[left])

{

temp = array[i]; array[i] = array[left]; array[left] = temp;

i = left;

}

else

{

temp = array[i]; array[i] = array[right]; array[right] = temp;

i = right;

}

left = 2 \* i + 1; right = 2 \* i + 2;

}/\*End of while\*/

if (left == n - 1 && array[i]) { temp = array[i];

array[i] = array[left]; array[left] = temp;

}

}

## Implement Priority Queue using Heap

#include<stdio.h> #include<malloc.h> void insert();

void del(); void display();

struct node

{

int priority; int info;

struct node \*next;

}\*start=NULL,\*q,\*temp,\*new;

typedef struct node N; int main()

{

int ch;

clrscr(); do

{

printf("\n[1] INSERTION\t[2] DELETION\t[3] DISPLAY [4] EXIT\t:");

scanf("%d",&ch); switch(ch)

{

case 1:insert(); break;

case 2:del(); break;

case 3:display(); break;

case 4: break;

}

}

while(ch<4);

}

void insert()

{

int item,itprio; new=(N\*)malloc(sizeof(N));

printf("ENTER THE ELT.TO BE INSERTED :\t");

scanf("%d",&item);

printf("ENTER ITS PRIORITY :\t");

scanf("%d",&itprio); new->info=item; new->priority=itprio; new->next=NULL; if(start==NULL )

{

//new->next=start; start=new;

}

else if(start!=NULL&&itprio<=start->priority)

{ new->next=start; start=new;

}

else

{

q=start;

while(q->next != NULL && q->next->priority<=itprio)

{q=q->next;}

new->next=q->next; q->next=new;

}

}

void del()

{

if(start==NULL)

{

printf("\nQUEUE UNDERFLOW\n");

}

else

{

new=start;

printf("\nDELETED ITEM IS %d\n",new->info); start=start->next;

//free(start);

}

}

void display()

{

temp=start; if(start==NULL)

printf("QUEUE IS EMPTY\n"); else

{

printf("QUEUE IS:\n"); if(temp!=NULL)

for(temp=start;temp!=NULL;temp=temp->next)

{

printf("\n%d priority =%d\n",temp->info,temp->priority);

//temp=temp->next;

}

}

}