**BANGALORE INSTITUTE OF TECHNOLOGY**

**K.R. ROAD, V.V PURAM, BENGALURU – 560 004**

**(AFFILIATED TO VTU, BELAGAVI)**

****

**DEPARTMENT OF INFORMATION SCIENCE**

**DATA STRUCTURES LABORATORY**

**CODE: BCSL305**

**III SEMESTER**

**Academic year 2023-2024**

**Prepared By:**

**S. Mercy**

Assistant Professor

Dept of ISE, BIT

**DATA STRUCTURES LABORATORY**

**PREREQUISITES:**

1. Knowledge of C Programming
2. Knowledge of Algorithms
3. Need to know the Fundamentals of Computers

**Course objectives:**

1. To develop skills to apply and analyze linear and nonlinear data structures.
2. To Strengthen the ability to identify and apply the suitable data structure for the given real world problem.
3. To Gain knowledge in practical applications of data structures

**Course Outcomes:**

* Analyze and Compare various linear and non-linear data structures such as stacks, queues, linked lists, trees and graphs using static and dynamic allocation.
* Demonstrate the working nature of different types of data structures and their

applications

* Develop, analyze and evaluate the searching algorithms.
* Apply hash table and hash function to resolve collision using linear probing.
* Choose the appropriate data structure for solving real world problems.

1. Develop a menu driven Program in C for the following  **Array** operations

1. Declare a Calendar as an array of 7 elements (A dynamically Created array) to

represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated string), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).

1. Write functions create(), read(), and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.

2. Develop a Program in C for the following operations on **Strings.**

a. Read a main String (**STR),** a Pattern String (**PAT**) and a Replace String(**REP**)

b. Perform Pattern Matching Operation: Find and Replace all occurrences of **PAT**

in **STR** with **REP** if **PAT** exists in **STR.** Report suitable messages in case **PAT**

does not exist in **STR**

Support the program with functions for each of the above operations. Don't use

Built-in functions.

3.Develop a menu driven Program in C for the following operations on **STACK** of

Integers (Array Implementation of Stack with maximum size **MAX**)

a. ***Push*** an Element on to Stack

b. ***Pop*** an Element from Stack

*c.* Demonstrate how Stack can be used to check ***Palindrome***

d. Demonstrate ***Overflow*** and ***Underflow*** situations on Stack

e. Display the status of Stack

f. Exit

Support the program with appropriate functions for each of the above operations.

4.Develop a Program in C for converting an Infix Expression to Postfix Expression.

Program should support for both parenthesized and free parenthesized Expressions with the operators: **+, -, \*, /, %(Remainder), ^(Power)** and

**alphanumeric** operands**.**

5. Develop a Program in C for the following Stack Applications

a. Evaluation of **Suffix expression** with single digit operands and operators:

**+, -, \*, /, %, ^**

b. Solving **Tower of Hanoi** problem with **n** disks.

6. Develop a menu driven Program in C for the following operations on **Circular**

**QUEUE** of Characters (Array Implementation of Queue with maximum size **MAX**)

a. Insert an Element on to Circular QUEUE

b. Delete an Element from Circular QUEUE

c. Demonstrate ***Overflow*** and ***Underflow*** situations on Circular QUEUE

d. Display the status of Circular QUEUE

e. Exit

Support the program with appropriate functions for each of the above operations

7. Develop a menu driven Program in C for the following operations on **Singly Linked**

**List (SLL)** of Student Data with the fields: ***USN,Name, Branch, Sem, PhNo***

a. Create a **SLL** of **N** Students Data by using ***front insertion***.

b. Display the status of **SLL** and count the number of nodes in it

c. Perform Insertion / Deletion at End of **SLL**

d. Perform Insertion / Deletion at Front of **SLL(Demonstration of stack)**

e. Exit

8. Develop a menu driven Program in C for the following operations on **Doubly Linked**

**List (DLL)** of Employee Data with the fields: ***SSN,Name, Dept, Designation, Sal,***

***PhNo***

a. Create a **DLL** of **N** Employees Data by using ***end insertion***.

b. Display the status of **DLL** and count the number of nodes in it

c. Perform Insertion and Deletion at End of **DLL**

d. Perform Insertion and Deletion at Front of **DLL**

e. Demonstrate how this **DLL** can be used as **Double Ended Queue**

f. Exit

9. Develop a Program in C for the following operations on **Singly Circular Linked List**

**(SCLL)** with header nodes

1. Represent and Evaluate a Polynomial **P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-**

**2xyz3**

1. Find the sum of two polynomials **POLY1(x,y,z)** and **POLY2(x,y,z)** and store

the result in **POLYSUM(x,y,z)**

Support the program with appropriate functions for each of the above operations.

10. Develop a menu driven Program in C for the following operations on **Binary Search**

**Tree (BST)** of Integers

a. Create a BST of **N** Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2

b. Traverse the BST in Inorder, Preorder and Post Order

c. Search the BST for a given element (**KEY**) and report the appropriate message

e. Exit

11. Develop a Program in C for the following operations on **Graph(G)** of Cities

a. Create a Graph of **N** cities using Adjacency Matrix.

b. Print all the nodes **reachable** from a given starting node in a digraph using

DFS/BFS method.

12. Given a File of **N** employee records with a set **K** of Keys(4-digit) which uniquely

determine the records in file **F**. Assume that file **F** is maintained in memory by a

Hash Table(HT) of **m** memory locations with **L** as the set of memory addresses

(2-digit) of locations in HT. Let the keys in **K** and addresses in **L** are Integers.

Design and develop a Program in C that uses Hash function **H: K** **L** as H(**K**)=**K**

mod **m (remainder** method),and implement hashing technique to map a given key **K**

to the address space **L.** Resolve the collision (if any) using **linear probing**.

1. Develop a program in C for the following

1. Declare a Calendar as an array of 7 elements (A dynamically Created array) to

represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated string), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).

1. Write functions create(), read(), and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.

An **array** is a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



**Declaring Arrays**

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows −

type arrayName [ arraySize ];

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement −

double balance[10];

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

**Initializing Arrays**

Initialize an array in C either one by one or using a single statement as follows −

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array −

balance[4] = 50.0;

The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array discussed above −



In this experiment the array can be represented as one / single dimensional elements. Various array operations are implemented with the help of following user defined functions,

a. create()

b. display()

c. insert()

d. del ()

e. exit()

**Application of Arrays:**

* Stores elements of same data type.
* Array used for maintaining multiple variable names using single name.
* Array can be used for Sorting Elements.
* Arrays can perform Matrix Operation.
* Array can be used in CPU Scheduling.

#include<stdio.h>

#include<stdlib.h>

struct DAY

{

char \*dayname;

int date;

char \*activity;

};

void create(struct DAY \*day)

{

day->dayname=(char \*)malloc(sizeof(char)\*20);

day->activity=(char \*)malloc(sizeof(char)\*50);

printf("Enter the name of the day ");

scanf("%s",day->dayname);

printf("Enter the date for the day”);

scanf("%d",&day->date);

printf("Enter the activity for the day”);

scanf("%s",day->activity);

}

void read(struct DAY \*calendar, int size)

{

for(int i=0;i<size;i++)

{

printf(“Enter details for day %d”i+1);

create(&calendar[i]);

}

}

void display(struct DAY \*calendar, int size)

{

printf("Activity Details”);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("Day\t\tName of the day\tDate\tActivity\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

for(int i=0;i<size;i++)

printf("%d\t\t%s\t\t%d\t%s\n",i+1,calendar[i].dayname,calendar[i].date,calendar[i].activity);

}

void freememory(struct DAY \*calendar,int size)

{

for(int i=0;i<size;i++)

{

free(calendar[i].dayname);

free(calendar[i]activity);

}

}

int main()

{

int size;

printf(“Enter the number of days in the week”);

scanf(“%d”,&size);

struct DAY \*calendar=(struct DAY \*)malloc(sizeof(struct DAY)\* size);

if(calendar==NULL)

{

printf(“Memory allocation failed”);

return 1;

}

read(calendar,size);

display(calendar,size);

freememory(calendar,size);

free(calendar);

return 0;

}

2. Develop a Program in C for the following operations on **Strings**

a. Read a main String (**STR),** a Pattern String (**PAT**) and a Replace String(**REP**)

b. Perform Pattern Matching Operation: Find and Replace all occurrences of **PAT**

in **STR** with **REP** if **PAT** exists in **STR.** Report suitable messages in case **PAT**

does not exist in **STR**  Support the program with functions for each of the

above operations. Don't use Built-in functions.

A **String** is actually one-dimensional array of characters terminated by a **null** character '\0'. The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

***char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};***

If you follow the rule of array initialization then you can write the above statement as follows:

***char greeting[] = "Hello";***

C language supports a wide range of built-in functions that manipulate null-terminated strings as follows:

strcpy(s1, s2); Copies string s2 into string s1.

strcat(s1, s2); Concatenates string s2 onto the end of string s1.

strlen(s1); Returns the length of string s1.

strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1.

strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.

#include<stdio.h>

char str[100], pat[50], rep[50], ans[100];

int i, j, c, m, k, flag=0;

void stringmatch()

{

i=m=c=j=0;

while(str[c]!='\0')

{

if(str[m]==pat[i])

{

i++;

m++;

if(pat[i]=='\0')

{

flag = 1;

for(k = 0; rep[k] != '\0'; k++, j++)

ans[j] = rep[k];

i = 0;

c = m;

}

}

else

{

ans[j] = str[c];

j++;

c++;

m = c;

i = 0;

}

}

ans[j] = '\0';

}

void main()

{

printf("\nEnter a main string \n");

gets(str);

printf("\nEnter a pattern string \n");

flushall();

gets(pat);

printf("\nEnter a replace string \n");

flushall();

gets(rep);

stringmatch();

if(flag==1)

printf("\nThe resultant string is\n %s" ,ans);

else

printf("\nPattern string NOT found\n");

}

**Output 1**

Enter a main string

This is Data Structure lab

Enter a pattern string

Data Structure

Enter a replace string

Data structure with C

The resultant string is

This is Data structure with C lab

**Output 2**

Enter a main string

This is Data Structure lab

Enter a pattern string

Date

Enter a replace string

DATA

Pattern string NOT found

3. Develop a menu driven Program in C for the following operations on **STACK** of

Integers (Array Implementation of Stack with maximum size **MAX**)

a. ***Push*** an Element on to Stack

b. ***Pop*** an Element from Stack

*c.* Demonstrate how Stack can be used to check ***Palindrome***

d. Demonstrate ***Overflow*** and ***Underflow*** situations on Stack

e. Display the status of Stack

f. Exit

Support the program with appropriate functions for each of the above operations.

**Stack** is an ordered list of similar data type. Stack is a LIFO structure. (Last in First out). push() function is used to insert new elements into the Stack and pop() is used to delete an element from the stack. Both insertion and deletion are allowed at only one end of Stack called Top.



A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, stack is implemented using arrays which make it a fixed size stack implementation.

Basic Operations:

* push() - pushing (storing) an element on the stack.
* pop() - removing (accessing) an element from the stack.
* display() – traversing the content of stack.
* isFull() − check if stack is full.
* isEmpty() − check if stack is empty.

**Application of Stack :**

* Parsing.
* Recursive Function.
* Calling Function.
* Expression Evaluation.
* Expression Conversion.
* Towers of hanoi.

#include<stdio.h>

#define MAX 4

int stack[MAX], item;

int ch, top = -1,status = 0;

/\*PUSH FUNCTION\*/

void push(int stack[], int item)

{

if (top == (MAX-1))

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

else

{

stack[++top] =item;

status++;

}

}

/\*POP FUNCTION\*/

int pop(int stack[])

{

int itemdel;

if(top == -1)

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

else

{

itemdel = stack[top--];

status--;

printf("\n Popped element is %d", itemdel);

}

return itemdel;

}

/\* FUNCTION TO CHECK STACK IS PALINDROME OR NOT \*/

void palindrome(int stack[])

{

int flag=1,i;

printf(" Stack contents are:\n");

for(i=top; i>=0; i--)

printf(“|%d|\n”,s[i]);

printf(“\n Reverse of stack content are:\n”);

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

printf(“|%d|\n”,s[i]);

for(i=0;i<=top/2;i++)

{

if(s[i]!=s[top-i])

{

flag=0;

break;

}

}

if(flag==1)

{

printf(“palindrome”);

}

else

{

printf(“Not a palindrome”);

}

}

void display(int stack[])

{

int i;

if(top == -1)

printf("\nStack is Empty");

else

{

printf("stack contents are\n");

for(i=top; i>=0; i--)

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

printf("\n");

}

}

/\*MAIN PROGRAM\*/

void main()

{

clrscr();

do

{

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

printf("\n1. PUSH (Insert) in the Stack");

printf("\n2. POP (Delete) from the Stack");

printf("\n3. PALINDROME check using Stack");

printf("\n4. DISPLAY the contents of Stack");

printf("\n5. Exit (End the Execution)");

printf("\nEnter Your Choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nEnter a element to be pushed: ");

scanf("%d", &item);

push(stack, item);

display(stack);

break;

case 2: item=pop(stack);

break;

case 3:palindrome(stack);

break;

case 4: display(stack);

break;

case 5:exit(0);

default: printf("\n Invalid choice");

break;

}//end switch

}while (ch != 5);

}

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Display the contents of stack
5. Exit (End the Execution)

|  |  |  |
| --- | --- | --- |
| Enter Your Choice: | 1 |  |
| Enter an element to be pushed: | | 1 |

The stack contents are:

-----

| 1 |

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Display the contents of stack
5. Exit (End the Execution)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Enter Your Choice: | | 1 |  |  |
| Enter an element to be pushed: | | | 2 |  |
| The stack contents are: | |  |  |  |
| ------ | |  |  |  |
| | 2 | | |  |  |  |
| ------ | |  |  |  |
| | 1 | | |  |  |  |
| (--- | AFTER THE 4 TIMES PUSH OPERATION | | | -------) |

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Display the contents of stack
5. Exit (End the Execution)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Enter Your Choice: | | 1 |  | |
| Enter an element to be pushed: | | | 9 | |
| **Stack is Overflow** | |  |  | |
| The stack contents are: | |  |  | |
| ------ | |  |  | |
| | 1 | | |  |  | |
| ------ | |  |  | |
| |2 | | | | |  |
| ------ | | | |  |
| |2 | | | | |  |
| ------ | | | |  |
| |1 | | | | |  |

----MAIN MENU----

1. PUSH (Insert) in the Stack
2. POP (Delete) from the Stack
3. PALINDROME check using Stack
4. Display the contents of stack
5. Exit (End the Execution)

|  |  |  |  |
| --- | --- | --- | --- |
| Enter Your Choice: | | 2 |  |
| Popped element is | | 1 |  |
| The stack contents are: | |  |  |
| ------ | |  |  |
| | 2 | | |  |  |
| ------ | |  |  |
| | 2 | | |  |  |
| ------ | |  |  |
| | 1 | | |  |  |

----MAIN MENU----

1. PUSH (Insert) in the Stack

2. POP (Delete) from the Stack

3. PALINDROME check using Stack

4. Display the contents of stack

5. Exit (End the Execution)

|  |  |  |
| --- | --- | --- |
| Enter Your Choice: | 4 |  |

|  |  |
| --- | --- |
| The stack contents are: | |
| ------ | |
| | 2 | | |
| ------ | |
| | 2 | | |
| ------ | |
| | 1 | | |

----MAIN MENU----

1. PUSH (Insert) in the Stack

2. POP (Delete) from the Stack

3. PALINDROME check using Stack

4. Display the contents of stack

1. Exit (End the Execution)

|  |  |  |
| --- | --- | --- |
| Enter Your Choice: | 3 |  |
| Stack contents are not Palindrome | | | | |  |

1. Develop a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: **+, -, \*, /, %(Remainder), ^(Power)** and **alphanumeric** operands**.**

**Infix Expression:** Operators are written in-between their operands. Ex: X + Y

**Postfix Expression:** Operators are written after their operands. Ex: XY+

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Expression** | **Current Symbol** | **Stack** | **Output** | **Comment** |
| A/B^C-D | Initial State | NULL | – | Initially Stack is Empty |
| /B^C-D | A | NULL | A | Print Operand |
| B^C-D | / | / | A | Push Operator Onto Stack |
| ^C-D | B | / | AB | Print Operand |
| C-D | ^ | /^ | AB | Push Operator Onto Stack because Priority of ^ is greater than Current Topmost Symbol of Stack i.e ‘/’ |
| -D | C | /^ | ABC | Print Operand |
| D | – | / | ABC^ | **Step 1 :** Now ‘^’ Has Higher Priority than Incoming Operator So We have to Pop Topmost Element . **Step 2 :**Remove Topmost Operator From Stack and Print it |
| D | – | NULL | ABC^/ | **Step 1 :** Now ‘/’ is topmost Element of Stack Has Higher Priority than Incoming Operator So We have to Pop Topmost Element again. **Step 2 :**Remove Topmost Operator From Stack and Print it |
| D | – | – | ABC^/ | **Step 1 :** Now Stack Becomes Empty and We can Push Operand Onto Stack |
| NULL | D | – | ABC^/D | Print Operand |
| NULL | NULL | – | ABC^/D- | Expression Scanning Ends but we have still one more element in stack so pop it and display it |

#include<stdio.h>

#include<string.h>

int stkpre(char symbol)

{

switch(symbol)

{

case '+' :

case '-': return 2;

case '\*':

case '/': return 4;

case '^':

case '$': return 5;

case '(': return 0;

case '#': return -1;

default: return 8;

}

}

int inpre(char symbol)

{

switch(symbol)

{

case '+':

case '-': return 1;

case '\*':

case '/': return 3;

case '^':

case '$': return 6;

case '(': return 9;

case ')': return 0;

default: return 7;

}

}

void infix\_postfix(char infix[], char postfix[])

{

int top, j, i;

char s[30], symbol;

top = -1;

s[++top] = '#';

j = 0;

for(i=0; i < strlen(infix); i++)

{

symbol = infix[i];

while(stkpre(s[top]) > inpre(symbol))

{

postfix[j] = s[top--];

j++;

}

if(stkpre(s[top]) != inpre(symbol))

s[++top] = symbol;

else

top--;

}

while(s[top] != '#')

{

postfix[j++] = s[top--];

}

postfix[j] = '\0';

}

void main()

{

char infix[20], postfix[20];

clrscr();

printf("Enter a valid infix expression\n");

gets(infix);

infix\_postfix(infix,postfix);

printf("The postfix expression is:\n");

printf ("%s",postfix);

}

**Output 1**

Enter a valid infix expression (a+(b-c)\*d)

The postfix expression is: abc -d\*+

**Output 2**

Enter a valid infix expression a+b\*c

The postfix expression is: abc\*+

5. Develop a Program in C for the following Stack Applications

a. Evaluation of **Suffix expression** with single digit operands and operators:

**+, -, \*, /, %, ^**

1. Solving **Tower of Hanoi** problem with **n** disks.

Program 5a

Following are the steps involved for evaluation postfix expressions.  
1) Create a stack to store operands (or values).  
2) Scan the given expression and do following for every scanned element.  
 a) If the element is a number, push it into the stack  
 b) If the element is a operator, pop operands for the operator from stack. Evaluate

the operator and push the result back to the stack  
3) When the expression is ended, the number in the stack is the final answer

**Example:**  
Let the given expression be “2 3 1 \* + 9 -“. We scan all elements one by one.  
1) Scan ‘2’, it’s a number, so push it to stack. Stack contains ‘2’  
2) Scan ‘3’, again a number, push it to stack, stack now contains ‘2 3′ (from bottom to

top)  
3) Scan ‘1’, again a number, push it to stack, stack now contains ‘2 3 1′  
4) Scan ‘\*’, it’s an operator, pop two operands from stack, apply the \* operator on

operands, we get 3\*1 which results in 3. We push the result ‘3’ to stack. Stack now

becomes ‘2 3′.  
5) Scan ‘+’, it’s an operator, pop two operands from stack, apply the + operator on

operands, we get 3 + 2 which results in 5. We push the result ‘5’ to stack. Stack

now becomes ‘5’.  
6) Scan ‘9’, it’s a number, we push it to the stack. Stack now becomes ‘5 9′.  
7) Scan ‘-‘, it’s an operator, pop two operands from stack, apply the – operator on

operands, we get 5 – 9 which results in -4. We push the result ‘-4′ to stack. Stack

now becomes ‘-4′.  
8) There are no more elements to scan, we return the top element from stack (which is

the only element left in stack).

Program 5b

The **Tower of Hanoi** is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

* Only one disk can be moved at a time.
* Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
* No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is 2*n* - 1, where *n* is the number of disks.

Program 5A

#include<stdio.h>

#include<math.h>

#include<string.h>

double compute(char symbol, double op1, double op2)

{

switch(symbol)

{

case '+': return op1 + op2;

case' -': return op1 - op2;

case '\*': return op1 \* op2;

case '/': return op1 / op2;

case '$':

case '^': return pow(op1,op2);

default: return 0;

}

}

void main()

{

double s[20], res, op1, op2;

int top, i;

char postfix[20], symbol;

printf("\nEnter the postfix expression:\n");

gets(postfix);

top=-1;

for(i=0; i<strlen(postfix); i++)

{

symbol = postfix[i];

if(isdigit(symbol))

s[++top] = symbol - '0';

else

{

op2 = s[top--];

op1 = s[top--];

res = compute(symbol, op1, op2);

s[++top] = res;

}

}

res = s[top--];

printf("\nThe result is : %f\n", res);

}

Program 5B : Solving **Tower of Hanoi** problem with **n** disks.

#include<stdio.h>

#include<math.h>

int count=0;

void tower(int n, int src, int temp,int dest)

{

if(n == 0)

return;

tower(n-1, src, dest, temp);

printf("\n Move disc %d from %c to %c", n, src, dest);

count++;

tower(n-1, temp, src, dest);

}

void main()

{

int n;

printf("\n Enter the number of discs: \n");

scanf("%d", &n);

tower(n, 'A', 'B', 'C');

printf("\n total number of moves = %d",count);

}

**5 a) program output**

**Output 1**

Enter the postfix expression: 23+

The result is: 5.000000

**Output 2**

Enter the postfix expression: 23+7\*

The result is: 35.000000

**5 b) program output**

Enter the number of discs: 3

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

Move disc 3 from A to C

Move disc 1 from B to A

Move disc 2 from B to C

Move disc 1 from A to C

Total Number of moves are : 7

6.Develop a menu driven Program in C for the following operations on **Circular**

**QUEUE** of Characters (Array Implementation of Queue with maximum size **MAX**)

a. Insert an Element on to Circular QUEUE

b. Delete an Element from Circular QUEUE

c. Demonstrate ***Overflow*** and ***Underflow*** situations on Circular QUEUE

d. Display the status of Circular QUEUE

e. Exit

Support the program with appropriate functions for each of the above operations.

**Circular queue** is a linear data structure. It follows FIFO principle. In **circular queue** the last node is connected back to the first node to make a **circle**. **Circular** linked list fallow the First In First Out principle. Elements are added at the rear end and the elements are deleted at front end of the **queue**. The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.

A circular queue looks like



**Consider the example with Circular Queue implementation:**

**Application of Queues:**

* Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.
* Computer systems must often provide a “holding area” for messages between two processes, two programs, or even two systems. This holding area is usually called a “buffer” and is often implemented as a queue.

#include<stdio.h>

#define MAX 4

int ch, front = 0, rear = -1, count=0, q[MAX], item;

void insert(int item, int \*rear, int \*q, int \*count)

{

if(\*count == MAX)

printf("Circular Queue is Full\n");

else

{

\*rear = (\*rear + 1) % MAX;

q[\*rear]=item;

(\*count)++;

}

}

void del(int \*front, int \*q, int \*count)

{

if(\*count == 0)

printf("Circular Queue is underflow\n");

else

{

item=q[\*front];

printf("\nDeleted item is: %d",item);

\*front = (\*front + 1) % MAX;

(\*count)--;

}

}

void display(int front, int q[], int count)

{

int i;

if(count == 0)

printf("\nCircular Queue is Empty");

else

{

printf("\nContents of Circualr Queue is:\n");

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

{

printf("%d\t", q[front]);

front = (front + 1) % MAX;

}

}

}

void main()

{

do

{

printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit");

printf("\nEnter the choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nEnter the item to be inserted: ");

scanf("%d", &item);

insert(item, &rear, q, &count);

break;

case 2: del(&front, q, &count);

break;

case 3: display(front, q, count);

break;

case 4: exit(0);

break;

}

}while(ch!=4);

}

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SAMPLE POUTPUT:** | | | | | | | |  | |  | | |
| 1. Insert | | 2. | | | Delete | | | 3. Display | | 4. Exit | | |
| Enter the choice: | | | | | 1 | | |  | |  | | |
| Enter the character / item to be inserted: | | | | | | | | | | **1** | | |
| 1. Insert | | 2. | | | Delete | | | 3. Display | | 4. Exit | | |
| Enter the choice: | | | | | 1 | | |  | |  | | |
| Enter the character / item to be inserted: | | | | | | | | | | **2** | |
| 1. Insert | | | 2. | | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 1 | |  | |  | |
| Enter the character / item to be inserted: | | | | | | | | | | **3** | |
| 1. Insert | | | 2. | | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 1 | |  | |  | |
| Enter the character / item to be inserted: | | | | | | | | | | **4** | |
| 1. Insert | | | 2. | | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 3 | |  | |  | |
| **Contents of Queue are:** | | | | | | | |  | |  | |
| **1** | **2** | | **3** | |  | **4** | |  | |  | |
| 1. Insert | | | 2. | | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 1 | |  | |  | |
| Enter the character / item to be inserted: | | | | | | | | | | **5** | |
| **Circular Queue is Full** | | | |  |  |  | |  | |  | |
| 1. Insert | | | | 2. | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 2 | |  | |  | |
| Deleted item is: | | | |  | **1** | |  | |  | |
| 1. Insert | | | 2. | | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 2 | |  | |  | |
| Deleted item is: | | | |  | **2** | |  | |  | |
| 1. Insert | | | | 2. | **Delete** | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | **3** | |  | |  | |
| **Contents of Queue is:** | | | | | | | |  | |  | |
| **3** | **4** | |  | |  |  | |  | |  | |
| 1. Insert | | | 2. | | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 1 | |  | |  | |
| Enter the character / item to be inserted: | | | | | | | | | | **5** | |
| 1. Insert | | | 2. | | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 3 | |  | |  | |
| **Contents of Queue is:** | | | | | | | |  | |  | |
| **3** | **4** | | **5** | |  | |  | |  | |
| 1. Insert | | | 2. | | Delete | | | 3. Display | | 4. Exit | |
| Enter the choice: | | | |  | 4 | |  | |  | |

7.Develop a menu driven Program in C for the following operations on  **Singly Linked List (SLL)** of Student Data with the fields: ***USN,Name, Branch, Sem, PhNo***

a. Create a **SLL** of **N** Students Data by using ***front insertion***.

b. Display the status of **SLL** and count the number of nodes in it

c. Perform Insertion / Deletion at End of **SLL**

d. Perform Insertion / Deletion at Front of **SLL(Demonstration of stack)**

e. Exit

#include<stdio.h>

int MAX=4,count;

struct student

{

char usn[10];

char name[30];

char branch[5];

int sem;

int phno[10];

struct student \*next; //Self referential pointer.

};

typedef struct student NODE;

int countnodes(NODE \*head)

{

NODE \*p;

count=0;

p=head;

while(p!=NULL)

{

p=p->next;

count++;

}

return count;

}

NODE\* getnode(NODE \*head)

{

NODE \*newnode;

newnode=(NODE\*)malloc(sizeof(NODE)); //Create first NODE

printf("\nEnter USN, Name, Branch, Sem, Ph.No\n");

gets(newnode->usn);

flushall();

gets(newnode->name);

flushall();

gets(newnode->branch);

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

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

newnode->next=NULL; //Set next to NULL...

head=newnode;

return head;

}

NODE\* display(NODE \*head)

{

NODE \*p;

if(head == NULL)

printf("\n No student data\n");

else

{

p=head;

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

printf("\nUSN\tNAME\t\tBRANCH\tSEM\tPh.NO.");

while(p!=NULL)

{

printf("\n%s\t%s\t\t%s\t%d\t%s", p->usn, p->name, p->branch, p->sem, p->phno);

p=p->next; //Go to next node...

}

printf("\n The no. of nodes in list is: %d",countnodes(head));

}

return head;

}

NODE\* create(NODE \*head)

{

NODE \*newnode;

if(head==NULL)

{

newnode=getnode(head);

head=newnode;

}

else

{

newnode=getnode(head);

newnode->next=head;

head=newnode;

}

return head;

}

NODE\* insert\_front(NODE \*head)

{

if(countnodes(head)==MAX)

printf("\nList is Full / Overflow!!");

else

head=create(head); //create()insert nodes at front

return head;

}

NODE\* insert\_rear(NODE \*head)

{

NODE \*p, \*newnode; p=head;

if(countnodes(head) == MAX)

printf("\nList is Full(QUEUE)!!");

else

{

if(head == NULL)

{

newnode=getnode(head);

head=newnode; //set first node to be head

}

else

{

newnode=getnode(head);

while(p->next!=NULL)

{

p=p->next;

}

p->next=newnode;

}

}

return head;

}

NODE\* insert(NODE \*head)

{

int ch;

do

{

printf("\n1.Insert at Front(First)\t2.Insert at End(Rear/Last)\t3.Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1: head=insert\_front(head);

break;

case 2: head=insert\_rear(head);

break;

case 3: break;

}

head=display(head);

}while(ch!=3);

return head;

}

NODE\* delete\_front(NODE \*head)

{

NODE \*p;

if(head==NULL)

printf("\nList is Empty/Underflow(STACK)");

else

{

p=head;

p=p->next; //Set 2nd NODE as head free(p);

printf("\n Front(first)node is deleted");

}

return head;

}

NODE\* delete\_rear(NODE \*head)

{

NODE \*p, \*q; p=head; q=NULL;

while(p->next!=NULL)

{

q=p;

p=p->next; //Go upto -1 NODE which you want to delete

}

q->next=NULL;

free(p); //Delete last NODE

p->next=NULL;

printf("\n Last(end) entry is deleted");

return head;

}

NODE\* del(NODE \*head)

{

int ch;

do

{

printf("\n1.Delete from Front(First)\t2. Delete from

End(Rear/Last))\t3.Exit");

printf("\nEnter your choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: head=delete\_front(head); break;

case 2: head=delete\_rear(head); break;

case 3: break;

}

head=display(head);

}

while(ch!=3);

return head;

}

NODE\* stack(NODE \*head)

{

int ch;

do

{

printf("\nSSL used as Stack...");

printf("\n 1.Insert at Front(PUSH) \t 2.Delete from Front(POP))\t3.Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1: head=insert\_front(head); break;

case 2: head=delete\_front(head); break;

case 3: break;

}

head=display(head);

}while(ch!=3);

return head;

}

void main()

{

int ch, i, n;

NODE \*head;

head=NULL;

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

do

{

printf("\n 1.Create\t 2.Display\t 3.Insert\t 4.Delete\t 5.Stack\t 6.Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1: printf("\nHow many student data you want to create: ");

scanf("%d", &n);

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

head=create(head); //Call to Create NODE...

break;

case 2: head=display(head); //Call to Display...

break;

case 3: head=insert(head); //Call to Insert...

break;

case 4: head=del(head); //Call to delete

break;

case 5: head=stack(head);

break;

case 6: exit(); //Exit...

}

}while(ch!=6);

}

**OUTPUT:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. Create | 2. Display | | 3. Insert | | | | 4. Delete | | 5. Stack | 6. Exit | | Exit |
| Enter your choice: 1 | | |  | |  | |  |  |  |  | |  |
| How many student data you want to create: | | | | | | | 2 |  |  |  | |  |
| Enter USN, Name, Branch, Sem, Ph.No | | | | | | |  |  |  |  | |  |
| 1bi12is001 | kumar | | is | | 3 | | 9900099000 | |  |  | |  |
| Enter USN, Name, Branch, Sem, Ph.No | | | | | | |  |  |  |  | |  |
| 1bi12is002 | ravi | | is | | 3 | | 9900099111 | |  |  | |  |
| 1. Create | 2. Display | | 3. Insert | | | | 4. Delete | | 5. Stack |  | |  |
| Enter your choice: 2 | | |  | |  | |  |  |  |  | |  |
| ----STUDENT DATA---- | | |  | |  | |  |  |  |  | |  |
| USN | NAME | | BRANCH | | | | SEM | Ph.NO. |  |  | |  |
| 1bi12is002 | ravi | | is | |  | | 3 | 9900099111 | |  | |  |
| 1bi12is001 | kumar | | is | |  | | 3 | 9900099000 | |  | |  |
| The no. of nodes in list is: | | | 2 | |  | |  |  |  |  | |  |
| 1. Create | 2. Display | | 3. Insert | | | | 4. Delete | | 5. Stack | 6. Exit | |  |
| Enter your choice: 3 | | |  | |  | |  |  |  |  | |  |
| 1.Insert at Front(First) | | | 2.Insert at End(Rear/Last) | | | | | | 3.Exit |  | |  |
| Enter your choice: 1 | | |  | |  | |  |  |  |  | |  |
| Enter USN, Name, Branch, Sem, Ph.No | | | | | | |  |  |  |  | |  |
| 1bi12is003 | suresh | | is | | 3 | | 9900099222 | |  |  | |  |
| ----STUDENT DATA---- | | |  | |  | |  |  |  |  | |  |
| USN | NAME | | BRANCH | | | | SEM | Ph.NO. |  |  | |  |
| 1bi12is003 | suresh | | is | |  | | 3 | 9900099222 | |  | |  |
| 1bi12is002 | ravi | | is | |  | | 3 | 9900099111 | |  | |  |
| 1bi12is001 | kumar | | is | |  | | 3 | 9900099000 | |  | |  |
| The no. of nodes in list is: | | | 3 | |  | |  |  |  |  | |  |
| 1.Insert at Front(First) | | | 2.Insert at End(Rear/Last) | | | | | | 3.Exit |  | |  |
| Enter your choice: 2 | | |  | |  | |  |  |  |  | |  |
| Enter USN, Name, Branch, Sem, Ph.No | | | | | | |  |  |  |  | |  |
| 1bi12is004 | naresh | | is | |  | | 3 | 9900099333 | |  | |  |
| ----STUDENT DATA---- | | |  | |  | |  |  |  |  | |  |
| USN | NAME | | BRANCH | | | | SEM | Ph.NO. |  |  | |  |
| 1bi12is003 | suresh | | is | |  | | 3 | 9900099222 | |  | |  |
| 1bi12is002 | ravi | | is | |  | | 3 | 9900099111 | |  | |  |
| 1bi12is001 | kumar | | is | |  | | 3 | 9900099000 | |  | |  |
| 1bi12is004 | naresh | | is | |  | | 3 | 9900099333 | |  | |  |
| The no. of nodes in list is: | | | 4 | |  | |  |  |  |  | |  |
| 1.Insert at Front(First) | | | | 2.Insert at End(Rear/Last) | | | | | 3.Exit | |  |  |
| Enter your choice: | | 3 | |  | |  | |  |  | |  |  |
| 1. Create | 2. Display | | | 3. Insert | | 4. Delete | | | 5. Stack | | 6.Exit |  |
| Enter your choice: | | 4 | |  | |  | |  |  | |  |  |
| 1. Delete from Front (First) | | | | 2. Delete from End (Rear/Last) | | | | | | | 3.Exit |  |
| Enter your choice: | | 1 | |  | |  | |  |  | |  |  |
| **Front (first) node is deleted** | | | |  | |  | |  |  | |  |  |
| ----STUDENT DATA---- | | | |  | |  | |  |  | |  |  |
| USN | NAME | | | BRANCH | | SEM | | Ph.NO. |  | |  |  |
| 1bi12is002 | ravi |  | | is | | 3 | | 9900099111 | | |  |  |
| 1bi12is001 | kumar | | | is | | 3 | | 9900099000 | | |  |  |
| 1bi12is004 | naresh | | | is | | 3 | | 9900099333 | | |  |  |
| The no. of nodes in list is: | | | | 3 | |  | |  |  | |  |  |
| 1. Delete from Front (First) | | | | 2. Delete from End (Rear/Last) | | | | | | | 3.Exit |  |
| Enter your choice: | | 2 | |  | |  | |  |  | |  |  |
| **Last (end) node is deleted** | | | |  | |  | |  |  | |  |  |

8.Develop a menu driven Program in C for the following operations on **DoublyLinked**

**List (DLL)** of Employee Data with the fields: ***SSN, Name, Dept, Designation, Sal,***

***PhNo***

a. Create a **DLL** of **N** Employees Data by using ***end insertion***.

b. Display the status of **DLL** and count the number of nodes in it

c. Perform Insertion and Deletion at End of **DLL**

d. Perform Insertion and Deletion at Front of **DLL**

e. Demonstrate how this **DLL** can be used as **Double Ended Queue**

f. Exit

#include<stdio.h>

int MAX=4, count;

struct emp

{

int ssn;

char name[20];

char dept[10];

char desig[15];

int sal;

char phno[10];

struct emp \*left;

struct emp \*right;

};

typedef struct emp NODE;

int countnodes(NODE \*head)

{

NODE \*p;

count=0;

p=head;

while(p!=NULL)

{

p=p->right;

count++;

}

return count;

}

NODE\* getnode(NODE \*head)

{

NODE \*newnode;

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

newnode->right=newnode->left=NULL;

printf("\nEnter SSN, Name, Dept, Designation, Sal,Ph.No\n");

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

flushall();

gets(newnode->name);

flushall();

gets(newnode->dept);

flushall();

gets(newnode->desig);

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

flushall();

gets(newnode->phno);

head=newnode;

return head;

}

NODE\* display(NODE \*head)

{

NODE \*p;

if(head==NULL)

printf("\nNo Employee data\n");

else

{

p=head;

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

printf("\nSSN\tNAME\tDEPT\tDESINGATION\tSAL\t\tPh.NO.");

while(p!=NULL)

{

printf("\n%d\t%s\t%s\t%s\t\t%d\t\t%s", p->ssn, p->name, p->dept, p->desig,

p- >sal, p->phno);

p = p->right; //Go to next node...

}

printf("\nThe no. of nodes in list is: %d", countnodes(head));

}

return head;

}

NODE\* create(NODE \*head) // creating & inserting at end.

{

NODE \*p, \*newnode;

p=head;

if(head==NULL)

{

newnode=getnode(head);

head=newnode;

}

else

{

newnode=getnode(head);

while(p->right!=NULL)

{

p=p->right;

}

p->right=newnode;

newnode->left=p;

}

return head;

}

NODE\* insert\_end(NODE \*head)

{

if(countnodes(head)==MAX)

printf("\nList is Full!!");

else

head=create(head);

return head;

}

NODE\* insert\_front(NODE \*head)

{

NODE \*p, \*newnode;

if(countnodes(head)==MAX)

printf("\nList is Full!!");

else

{

if(head==NULL)

{

newnode=getnode(head);

head=newnode; //set first node to be head

}

else

{

newnode=getnode(head);

newnode->right=head;

head->left=newnode;

head=newnode;

}

}

return head;

}

NODE\* insert(NODE \*head)

{

int ch;

do

{

printf("\n 1.Insert at Front(First) \t 2.Insert at End(Rear/Last)\t3.Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1: head=insert\_front(head); break;

case 2: head=insert\_end(head); break;

case 3: break;

}

head=display(head);

} while(ch!=3);

return head;

}

NODE\* delete\_front(NODE \*head)

{

NODE \*p;

if(head==NULL)

printf("\nList is Empty (QUEUE)");

else

{

p=head;

head=head->right;

head->right->left=NULL;

free(p);

printf("\nFront(first)node is deleted");

}

return head;

}

NODE\* delete\_end(NODE \*head)

{

NODE \*p, \*q;

q=NULL;

p=head;

while(p->right!=NULL)

{

q=p;

p=p->right; //Go upto -1 node which you want to delete

}

q->right=NULL;

free(p); //Delete last node...

printf("\nLast(end) entry is deleted");

return head;

}

NODE \*del(NODE \*head)

{

int ch;

do

{

printf("\n1.Delete from Front(First)\t2. Delete from End(Rear/Last))\t3.Exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1: head=delete\_front(head); break;

case 2: head=delete\_end(head); break;

case 3: break;

}

head=display(head);

}while(ch!=3);

return head;

}

NODE\* dqueue(NODE \*head)

{

int ch;

do

{

printf("\n DLL used as Double Ended Queue");

printf("\n 1. Insert at Rear \n

2. Delete from Front\n

3.Insert at Front \n

4.Delete from Rear\n

5.display\n

6. exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1: head=insert\_end(head); break;

case 2: head=delete\_front(head); break;

case 3: head=insert\_front(head); break;

case 4: head=delete\_end(head); break;

case 5: head=display(head);break;

case 6: break;

}

}while(ch!=6);

return head;

}

void main()

{

int ch, i, n;

NODE \*head;

head=NULL;

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

do

{

printf("\n1.Create\t2.Display\t3.Insert\t4.Delete\t5.Queue\t6.Exit");

printf("\nEnter your choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nHow many employees data you want to create: ");

scanf("%d", &n);

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

head=create(head); //Call to Create node...

break;

case 2: head=display(head); //Call to Display...

break;

case 3: head=insert(head); //Call to Insert...

break;

case 4: head=del(head); //Call to delete

break;

case 5: head=dqueue(head);

break;

case 6: exit(0); //Exit...

break;

}

}while(ch!=6);

}

**OUTPUT**

----------Employee Database-----------

1. Create 2. Display 3. Insert 4. Delete 5. Queue 7. Exit

Enter your choice: 1

How many employees data you want to create: 2

Enter SSN, Name, Dept, Designation, Sal, Ph.No

1 KUMAR ISE INSTRUCTOR 8000 900099000

Enter SSN, Name, Dept, Designation, Sal, Ph.No

2 RAVI ISE INSTRUCTOR 9000 900099111

1. Create 2. Display 3. Insert 4. Delete 5. Queue 7. Exit

Enter your choice: 2

----EMPLOYEE DATA----

SSN NAME DEPT DESINGATION SAL Ph.NO.

1 KUMAR ISE INSTRUCTOR 8000 900099000

2 RAVI ISE INSTRUCTOR 9000 900099111

The no. of nodes in list is: 2

1. Create 2. Display 3. Insert 4. Delete 5. Queue 7. Exit

Enter your choice: 3

1. Insert at Front (First) 2.Insert at End (Rear/Last) 3.Exit

Enter your choice: 1

Enter SSN, Name, Dept, Designation, Sal, Ph.No

3 SUNIL ISE ATTENDER 6000 900099333

----EMPLOYEE DATA----

SSN NAME DEPT DESINGATION SAL Ph.NO.

3 JIM CSE ATTENDER 6000 900099333

1 KUMAR CSE INSTRUCTOR 8000 900099000

2 RAVI ISE INSTRUCTOR 9000 900099111

The no. of nodes in list is: 3

1. Insert at Front (First) 2.Insert at End (Rear/Last) 3.Exit

Enter your choice: 3

1. Create 2. Display 3. Insert 4. Delete 5. Queue 7. Exit

Enter your choice: 7

9.Develop a Program in C for the following operations on **Singly Circular Linked**

**List (SCLL)** with header nodes

a. Represent and Evaluate a Polynomial **P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3**

b. Find the sum of two polynomials **POLY1(x,y,z)** and **POLY2(x,y,z)** and store

the result in **POLYSUM(x,y,z).**

Support the program with appropriate functions for each of the above operations.

#include<stdio.h>

#include<alloc.h>

#include<math.h>

struct node

{

int cf, px, py, pz;

int flag;

struct node \*link;

};

typedef struct node NODE;

NODE\* getnode()

{

NODE \*x;

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

if(x==NULL)

{

printf("Insufficient memory\n");

exit(0);

}

return x;

}

void display(NODE \*head)

{

NODE \*temp;

if(head->link==head)

{

printf("Polynomial does not exist\n");

return;

}

temp=head->link;

printf("\n");

while(temp!=head)

{

printf("%d x^%d y^%d z^%d",temp->cf,temp->px,temp->py,temp->pz);

if(temp->link != head)

printf(" + ");

temp=temp->link;

}

printf("\n");

}

NODE\* insert\_rear(int cf,int x,int y,int z,NODE \*head)

{

NODE \*temp,\*cur;

temp=getnode();

temp->cf=cf;

temp->px=x;

temp->py=y;

temp->pz=z;

cur=head->link;

while(cur->link!=head)

{

cur=cur->link;

}

cur->link=temp;

temp->link=head;

return head;

}

NODE\* read\_poly(NODE \*head)

{

int px, py, pz, cf, ch;

printf("\nEnter coeff: ");

scanf("%d",&cf);

printf("\nEnter x, y, z powers(0-indiacate NO term): ");

scanf("%d%d%d", &px, &py, &pz);

head=insert\_rear(cf,px,py,pz,head);

printf("\nIf you wish to continue press 1 otherwise 0: ");

scanf("%d", &ch);

while(ch != 0)

{

printf("\nEnter coeff: ");

scanf("%d",&cf);

printf("\nEnter x, y, z powers(0-indiacate NO term): ");

scanf("%d%d%d", &px, &py, &pz);

head=insert\_rear(cf,px,py,pz,head);

printf("\nIf you wish to continue press 1 otherwise 0: ");

scanf("%d", &ch);

}

return head;

}

NODE\* add\_poly(NODE \*h1,NODE \*h2,NODE \*h3)

{

NODE \*p1,\*p2;

int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;

p1=h1->link;

while(p1!=h1)

{

x1=p1->px;

y1=p1->py;

z1=p1->pz;

cf1=p1->cf;

p2=h2->link;

while(p2!=h2)

{

x2=p2->px;

y2=p2->py;

z2=p2->pz;

cf2=p2->cf;

if(x1==x2 && y1==y2 && z1==z2)break;

p2=p2->link;

}

if(p2!=h2)

{

cf=cf1+cf2;

p2->flag=1;

if(cf!=0)

h3=insert\_rear(cf,x1,y1,z1,h3);

}

else

h3=insert\_rear(cf1,x1,y1,z1,h3);

p1=p1->link;

}

p2=h2->link;

while(p2!=h2)

{

if(p2->flag==0)

h3=insert\_rear(p2->cf,p2->px,p2->py,p2->pz,h3);

p2=p2->link;

}

return h3;

}

void evaluate(NODE \*h)

{

NODE \*head;

int x, y, z;

float result=0.0;

head=h;

printf("\nEnter x, y, z, terms to evaluate:\n");

scanf("%d%d%d", &x, &y, &z);

while(h->link != head)

{

result = result + (h->cf \* pow(x,h->px) \* pow(y,h->py) \* pow(z,h->pz));

h=h->link;

}

result = result + (h->cf \* pow(x,h->px) \* pow(y,h->py) \* pow(z,h->pz));

printf("\nPolynomial result is: %f", result);

}

void main()

{

NODE \*h,\*h1,\*h2,\*h3;

int ch;

while(1)

{

printf("\n\n1.Evaluate polynomial\n2.Add two polynomials\n3.Exit\n");

printf("Enter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:printf("\nEnter polynomial to evaluate:\n");

h=getnode();

h->link=h;

h=read\_poly(h);

display(h);

evaluate(h);

break;

case 2:printf("\nEnter the first polynomial:");

h1=getnode();

h1->link=h1;

h1=read\_poly(h1);

printf("\nEnter the second polynomial:");

h2=getnode();

h2->link=h2;

h2=read\_poly(h2);

h3=getnode();

h3->link=h3;

h3=add\_poly(h1,h2,h3);

printf("\nFirst polynomial is: ");

display(h1);

printf("\nSecond polynomial is: ");

display(h2);

printf("\nThe sum of 2 polynomials is: ");

display(h3);

break;

case 3:exit(0);

break;

default:printf("\nInvalid entry");

break;

}

}

}

**OUTPUT:**

1. Evaluate polynomial

2. Add two polynomials

3. Exit

Enter your choice: 1

Enter polynomial to evaluate:

Enter coeff: 6

Enter x, y, z powers: 2 2 1

If you wish to continue press 1 otherwise 0: 1

Enter coeff: -4

Enter x, y, z powers: 0 1 5

If you wish to continue press 1 otherwise 0: 1

Enter coeff: 3

Enter x, y, z powers: 3 1 1

If you wish to continue press 1 otherwise 0: 1

Enter coeff: 2

Enter x, y, z powers: 1 5 1

If you wish to continue press 1 otherwise 0: 1

Enter coeff: -2

Enter x, y, z powers: 1 1 3

If you wish to continue press 1 otherwise 0: 0

Polynomial is:

6 x^2 y^2 z^1 + -4 x^0 y^1 z^5 + 3 x^3 y^1 z^1 + 2 x^1 y^5 z^1 + -2 x^1 y^1 z^3

Enter x, y, z, terms to evaluate: 1 1 1

Polynomial result is: 5.000000

1. Evaluate polynomial

2. Add two polynomials

3. Exit

Enter your choice: 2

Enter 1st polynomial:

Enter coeff: 4

Enter x, y, z powers: 2 2 2

If you wish to continue press 1 otherwise 0: 1

Enter coeff: 3

Enter x, y, z powers: 1 1 2

If you wish to continue press 1 otherwise 0: 0

Enter 2nd polynomial:

Enter coeff: 6

Enter x, y, z powers: 2 2 2

If you wish to continue press 1 otherwise 0: 0

1st Polynomial is:

4 x^2 y^2 z^2 + 3 x^1 y^1 z^2

2nd Polynomial is: 6 x^2 y^2 z^2

Result:10 x^2 y^2 z^2 + 3 x^1 y^1 z^2

1. Evaluate polynomial

2. Add two polynomials

3. Exit

Enter your choice: 3

10. Develop a menu driven Program in C for the following operations on **Binary Search**

**Tree (BST)** of Integers

a. Create a BST of **N** Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2

b. Traverse the BST in Inorder, Preorder and Post Order

c. Search the BST for a given element (**KEY**) and report the appropriate message

e. Exit

A **binary search tree** (BST) is a tree in which all nodes follows the below

mentioned properties

* The left sub-tree of a node has key less than or equal to its parent node's key.
* The right sub-tree of a node has key greater than or equal to its parent node's key.

Thus, a binary search tree (BST) divides all its sub-trees into two segments; left sub-tree and right sub-tree and can be defined as



Following are basic primary operations of a tree which are following.

* Search − search an element in a tree.
* Insert − insert an element in a tree.
* Preorder Traversal − traverse a tree in a preorder manner.
* Inorder Traversal − traverse a tree in an inorder manner.
* Postorder Traversal − traverse a tree in a postorder manner.

**Application of Trees:**

* Manipulate hierarchical data.
* Make information easy to search (see tree traversal).
* Manipulate sorted lists of data.
* As a workflow for compositing digital images for visual effects.
* Router algorithms

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node \*llink;

struct node \*rlink;

};

typedef struct node NODE;

NODE \*insert(int item,NODE \*root)

{

NODE \*temp,\*cur,\*prev;

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

temp->info=item;

temp->llink=NULL;

temp->rlink=NULL;

if(root==NULL)

return temp;

prev=NULL;

cur=root;

while(cur!=NULL)

{

prev=cur;

cur=(item<=cur->info)?cur->llink:cur->rlink;

}

if(item<prev->info)

prev->llink=temp;

else

prev->rlink=temp;

return root;

}

NODE \*construct\_BST(NODE \*root)

{

int a,n,i;

printf("Enter the number of elements\n");

scanf("%d",&n);

printf("Enter the elements to be inserted in the tree\n");

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

{

scanf("%d",&a);

root=insert(a,root);

}

printf("Tree Constructed Successfully!!!!!!\n");

return root;

}

void preorder(NODE \*root)

{

if(root!=NULL)

{

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

preorder(root->llink);

preorder(root->rlink);

**}**

}

void inorder(NODE \*root)

{

if(root!=NULL)

{

inorder(root->llink);

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

inorder(root->rlink);

}

}

void postorder(NODE \*root)

{

if(root!=NULL)

{

postorder(root->llink);

postorder(root->rlink);

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

}

}

int search\_element(NODE \*root,int key)

{

NODE \*cur;

int n=0;

cur=root;

if (cur!=NULL)

{

if (key==cur->info)

n=1;

else if (key<cur->info)

return search\_element(cur->llink,key);

else

return search\_element(cur->rlink,key);

}

else

return n;

}

void main()

{

int item,ch,key,n;

NODE \*root;

root=NULL;

while (1)

{

printf(" 1.Construct BST\n

2.Preorder\n

3.Inorder\n

4.Postorder\n

5.Search an Element\n

6:Exit\n");

printf("\nEnter the choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1: root=construct\_BST(root);

break;

case 2: preorder(root);

break;

case 3: inorder(root);

break;

case 4: postorder(root);

break;

case 5: if (root==NULL)

printf("List Empty\n");

else

{

printf("Enter the element\n");

scanf("%d",&key);

n=search\_element(root,key);

if(n)

printf("Key found\n");

else

printf("Not found\n");

}

break;

case 6: exit(0);

default: printf("Wrong Choice\n");

}

}

}

**OUTPUT:**

1. Construct BST

2. Preorder

3. Inorder

4. Postorder

5.Search an element

6. Exit

Enter your choice

1

Enter the number of Elements

4

Enter the Elements to be inserted in the tree

1 2 3 4

Tree Constructed Successfully!!!!!!

1. Construct BST

2. Preorder

3. Inorder

4. Postorder

5. Search an element

6. Exit

Enter your choice

2

1 2 3 4

1. Construct BST

2. Preorder

3. Inorder

4. Postorder

5. Search an element

6. Exit

Enter your choice

3

1 2 3 4

1. Construct BST

2. Preorder

3. Inorder

4. Postorder

5. Search an element

6. Exit

Enter your choice 4

4 3 2 1

1. Construct BST

2. Preorder

3. Inorder

4. Postorder

5.Search an element

6. Exit

Enter your choice

5

Enter the element

3

Key found

1. Construct BST

2. Preorder

3. Inorder

4. Postorder

5. Search an element

6. Exit

Enter your choice : 6

11. Design, Develop and Implement a Program in C for the following operations on

**Graph(G)** of Cities

a. Create a Graph of **N** cities using Adjacency Matrix.

b. Print all the nodes **reachable** from a given starting node in a digraph using

DFS/BFS method.

A **graph** G = (V, E) where v= {0, 1, 2, . . .n-1} can be represented using two dimensional integer array of size n x n. a[20][20] can be used to store a graph with 20 vertices. a[i][j] = 1, indicates presence of edge between two vertices i and j. a[i][j] = 0, indicates absence of edge between two vertices i and j. A graph is represented using square matrix.

* Adjacency matrix of an undirected graph is always a symmetric matrix, i.e. an edge (i, j) implies the edge (j, i).
* Adjacency matrix of a directed graph is never symmetric, adj[i][j] = 1 indicates a directed edge from vertex i to vertex j.

**Application of Graphs:**

* Social network graphs: to tweet or not to tweet.
* Transportation networks.
* Utility graphs and Document link graphs.
* Robot planning and neural networks.

**Breadth First Search (BFS)** algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs following rules.

* Rule 1 − Visit adjacent unvisited vertex. Mark it visited. Display it. Insert it in a queue.
* Rule 2 − If no adjacent vertex found, remove the first vertex from queue.
* Rule 3 − Repeat Rule 1 and Rule 2 until queue is empty.

|  |  |  |
| --- | --- | --- |
| **Step** | **Traversal** | **Description** |
| 1. | Breadth First Search Step One | Initialize the queue. |
| 2. | Breadth First Search Step Two | We start from visiting **S** (starting node), and mark it as visited. |
| 3. | Breadth First Search Step Three | We then see an unvisited adjacent node from **S**. In this example, we have three nodes but alphabetically we choose **A**, mark it as visited and enqueue it. |
| 4. | Breadth First Search Step Four | Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it. |
| 5. | Breadth First Search Step Five | Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it. |
| 6. | Breadth First Search Step Six | Now, **S** is left with no unvisited adjacent nodes. So, we dequeue and find **A**. |
| 7. | Breadth First Search Step Seven | From **A** we have **D** as unvisited adjacent node. We mark it as visited and enqueue it. |

**Depth First Search (DFS)** algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, DFS algorithm traverses from A to B to C to D first then to E, then to F and lastly to G. It employs the following rules.

* **Rule 1** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
* **Rule 2** − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
* **Rule 3** − Repeat Rule 1 and Rule 2 until the stack is empty.

|  |  |  |
| --- | --- | --- |
| **Step** | **Traversal** | **Description** |
| 1. | Depth First Search Step One | Initialize the stack. |
| 2. | Depth First Search Step Two | Mark **S** as visited and put it onto the stack. Explore any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order. |
| 3. | Depth First Search Step Three | Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only. |
| 4. | Depth First Search Step Four | Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order. |
| 5. | Depth First Search Step Five | We choose **B**, mark it as visited and put onto the stack. Here **B** does not have any unvisited adjacent node. So, we pop **B** from the stack. |
| 6. | Depth First Search Step Six | We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find **D** to be on the top of the stack. |
| 7. | Depth First Search Step Seven | Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack |

#include<stdio.h>

#include<stdlib.h>

void bfs(int a[10][10], int n, int u)

{

Int f,r,q[10],v;

int s[10]={0};//initialize all elem in s to 0, no node visited

printf("The nodes visited from %d:",u);

f=0, r=-1;// Q empty

q[++r]=u; // Insert u into Q

s[u]=1; // Insert u to s

printf("%d",u); //pritn node visited

while(f<=r)

{

u=q[f++]; //del an elem from Q

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

{

if(a[u][v]==1) //If v is adjacent to u

{

if(s[v]==0) // If v is not in S i.e, V has not been visited

{

printf("%d",v); // print the node visited

s[v]=1; //add v to s,mark as visited

q[++r]=v; //insert v into Q

}

}

}

}

printf("\n");

}

void main()

{

int n,a[10][10], source, i,j;

printf("Enter the number of nodes:");

scanf("%d",&n);

printf("Enter the adjacency matrix:\n");

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

{

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

{

scanf("%d", &a[i][j]);

}

}

printf(" enter the source vertex\n");

scanf("%d", &source);

bfs(a,n,source);

}

b. Print all the nodes **reachable** from a given starting node in a digraph using DFS/BFS

method.

#include<stdio.h>

#include<stdlib.h>

int visited[10];

int a[10][10];

int n;

void readadjmatrix();

void dfs(int);

void main()

{

int start;clrscr(); readadjmatrix();

printf("Enter the starting vertex:\n");

scanf("%d",&start);

dfs(start);

}

void readadjmatrix()

{

int i,j;

printf("Enter the number of vertices:\n");

scanf("%d",&n);

printf("Enter adjacency matrix\n"); for(i=1;i<=n;i++)

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

scanf("%d",&a[i][j]);

}

void dfs(int v)

{

int w;

visited[v]=1;

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

{

if(visited[w]==0&&a[v][w]==1)

{

printf("%d",w);

dfs(w);

}

}

}

**BFS output:**

Enter the number of nodes: 4

Enter the adjacency matrix:

0 1 1 0

0 0 1 1

0 0 0 1

0 0 0 0

The nodes visited from 0: 0 1 2 3

The nodes visited from 1: 1 2 3

The nodes visited from 2: 2 3

The nodes visited from 3: 3

**DFS output:**

Enter the number of nodes: 4

Enter the adjacency matrix:

0 1 1 0

0 0 1 1

0 0 0 1

0 0 0 0

Enter the starting vertex: 1

1->2->3->4

12. Given a File of **N** employee records with a set **K** of Keys(4-digit) which uniquely

determine the records in file **F**. Assume that file **F** is maintained in memory by a

Hash Table(HT) of **m** memory locations with **L** as the set of memory addresses

(2-digit) of locations in HT. Let the keys in **K** and addresses in **L** are Integers.

Design and develop a Program in C that uses Hash function **H: K** **L** as H(**K**)=**K**

mod **m (remainder** method), and implement hashing technique to map a given key

**K** to the address space **L.** Resolve the collision (if any) using **linear probing**.

**Hashing:**

Hashing is a technique to convert a range of key values into a range of indexes of an array. **Hash Table** is a data structure which store data in associative manner. In hash table, data is stored in array format where each data values have its own unique index value. Access of data becomes very fast if we know the index of desired data. Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of size of data. Hash Table uses array as a storage medium and uses hash technique to generate index where an element is to be inserted or to be located from. We're going to use modulo operator to get a range of key values. Item are in (key, value) format.



**Hash Function:** Define a hashing method to compute the hash code of the key of the data item.

int hashCode(int key)

{

return key % SIZE;

}

**Applications of Hashing:**

* Hashing is used in them to index and retrieve items.
* Hashing as a method is used in comparing strings by generating rolling hash as part of Rabin–Karp algorithm.
* Hashing algorithms (Hash functions) are widely used in cryptography.
* Hashing is used to map values to cache machines.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define HASH\_SIZE 100

typedef struct employee

{

int id;

char name[20];

}EMPLOYEE;

/\*Create initial hash table\*/

void initialize\_hash\_table(EMPLOYEE a[])

{

int i;

for(i=0; i<HASH\_SIZE; i++)

{

a[i].id=0;

}

}

/\*Compute hash value using the function: H(K)=k%m\*/

int H(int k)

{

return k % HASH\_SIZE;

}

/\*Insert an item into the empty slot using linear probing\*/

void insert\_hash\_table(int id, char name[], EMPLOYEE a[])

{

int i,index,h\_value;

h\_value= H(id);

for(i=0; i<HASH\_SIZE; i++)

{

index=(h\_value+i)% HASH\_SIZE;

if(a[index].id==0) //empty slot found

{

a[index].id=id; //insert employee id

strcpy(a[index].name,name); //insert employee name

break;

}

}

if(i==HASH\_SIZE) printf("Hash table full\n"); // NO empty slot

}

/\*Display the hash table\*/

void display\_hash\_table(EMPLOYEE a[], int n)

{

int k,i;

printf("H\_Value\t Emp\_id\t Emp\_name\n");

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

{

if(a[i].id!=0)

printf("%d\t %d\t %s\n",i,a[i].id,a[i].name);

}

}

void main()

{

EMPLOYEE a[HASH\_SIZE];

char name[20];

int key,id,choice,flag;

initialize\_hash\_table(a); //Create intial hash table

while(1)

{

printf("1:Insert 2:Display 3:Exit\n");

printf("Enter the choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("Enter emp id:");

scanf("%d",&id);

printf("Enter the name:");

scanf("%s",name);

insert\_hash\_table(id,name,a);

break;

case 2: printf("Contents of hash table\n");

display\_hash\_table(a,HASH\_SIZE);

printf("\n");

break;

case 3: exit(0);

default: printf("Invalid choice\n");

}

}

}

**OUTPUT:**

1: Insert 2: Display 3: Exit

Enter the choice: 1

Enter the empid: 1234

Enter the name: Anu

1: Insert 2: Display 3: Exit

Enter the choice: 1

Enter the empid: 1236

Enter the name: Anil

1: Insert 2: Display 3: Exit

Enter the choice: 1

Enter the empid: 1334

Enter the name: Charan

1: Insert 2: Display 3: Exit

Enter the choice: 2

Contents of hash table

H\_value Emp\_id Emp\_name

34 1234 Anu

35 1334 Charan

36 1236 Anil

1: Insert 2: Display 3: Exit

Enter the choice: 3