RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

25 years OF ACADEMIC EXCELLENCE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Laboratory Manual

REGULATION 2023

CS23231 – DATA STRUCTURES

RAJALAKSHMI ENGINEERING COLLEGE

An Autonomous Institution, Affiliated to Anna University Rajalakshmi Nagar, Thandalam – 602 105

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS23231 – DATA STRUCTURES (*Regulation 2023*)
LAB MANUAL

NAME : PRIYANGA M
REGISTER NO :231901037
YEAR/BRANCH/SECTION:2023/CSE(CS)
SEMESTER :2ND SEMESTER
ACADEMIC YEAR :2023-2024

LESSON PLAN

Course Code	Course Title (Laboratory Integrated Theory Course)	L	T	P	C
CS23231	Data Structures	1	0	6	4

LIST OF EXPERIMENTS	
Sl. No	**Name of the experiment**
Week 1	Implementation of Single Linked List (Insertion, Deletion and Display)
Week 2	Implementation of Doubly Linked List (Insertion, Deletion and Display)
Week 3	Applications of Singly Linked List (Polynomial Manipulation)
Week 4	Implementation of Stack using Array and Linked List implementation
Week 5	Applications of Stack (Infix to Postfix)
Week 6	Applications of Stack (Evaluating Arithmetic Expression)
Week 7	Implementation of Queue using Array and Linked List implementation
Week 8	Implementation of Binary Search Tree
Week 9	Performing Tree Traversal Techniques
Week 10	Implementation of AVL Tree
Week 11	Performing Topological Sorting
Week 12	Implementation of BFS, DFS
Week 13	Implementation of Prim's Algorithm
Week 14	Implementation of Dijkstra's Algorithm
Week 15	Program to perform Sorting
Week 16	Implementation of Open Addressing (Linear Probing and Quadratic Probing)

Week 17	Implementation of Rehashing

INDEX

Note: Students have to write the Algorithms at left side of each problem statements.

Ex. No.: 01	Implementation of Single Linked List	Date: 12.03.2024

Write a C program to implement the following operations on Singly Linked List.

(i) **Insert a node in the beginning of a list.**

(ii) **Insert a node after P**

(iii) **Insert a node at the end of a list**

(iv) **Find an element in a list**

(v) **FindNext**

(vi) **FindPrevious**

(vii) **isLast**

(viii) **isEmpty**

(ix) **Delete a node in the beginning of a list.**

(x) **Delete a node after P**

(xi) **Delete a node at the end of a list**

(xii) **Delete the List**

CODING:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int Element;
struct node *Next;
};
typedef struct node Node;
int IsEmpty(Node *List);
int IsLast(Node *Position);
Node *Find(Node *List, int x);
Node *FindPrevious(Node *List, int x);
Node *FindNext(Node *List, int x);
void InsertBeg(Node *List, int e);
void InsertLast(Node *List, int e);
void InsertMid(Node *List, int p, int e);
void DeleteBeg(Node *List);
void DeleteEnd(Node *List);
void DeleteMid(Node *List, int e);
void Traverse(Node *List);
int main()
{
Node *List = malloc(sizeof(Node));
List->Next = NULL;
Node *Position;
int ch, e, p;
printf("1.Insert Beg \n2.Insert Middle \n3.Insert End");
```

```
printf("\n4.Delete Beg \n5.Delete Middle \n6.Delete End");
printf("\n7.Find \n8.Traverse \n9.Exit\n");
do
{
printf("Enter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
printf("Enter the element : ");
scanf("%d", &e);
InsertBeg(List, e);
break;
case 2:
printf("Enter the position element : ");
scanf("%d", &p);
printf("Enter the element : ");
scanf("%d", &e);
InsertMid(List, p, e);
break;
case 3:
printf("Enter the element : ");
scanf("%d", &e);
InsertLast(List, e);
break;
case 4:
DeleteBeg(List);
break;
case 5:
printf("Enter the element : ");
scanf("%d", &e);
DeleteMid(List, e);
break;
case 6:
DeleteEnd(List);
break;
case 7:
printf("Enter the element : ");
scanf("%d", &e);
Position = Find(List, e);
if(Position != NULL)
printf("Element found...!\n");
else
printf("Element not found...!\n");
break;
case 8:
Traverse(List);
break;
}
} while(ch<= 8);
return 0;
}
int IsEmpty(Node *List)
```

```
{
if(List->Next == NULL)
return 1;
else
return 0;
}
int IsLast(Node *Position)
{
if(Position->Next == NULL)
return 1;
else
return 0;
}
Node *Find(Node *List, int x)
{
Node *Position;
Position = List->Next;
while(Position != NULL && Position->Element != x)
Position = Position->Next;
return Position;
}
Node *FindPrevious(Node *List, int x)
{
Node *Position;
Position = List;
while(Position->Next != NULL && Position->Next->Element != x)
Position = Position->Next;
return Position;
}
Node *FindNext(Node *List, int x)
{
Node *Position;
Position = Find(List, x);
return Position->Next;
}
void InsertBeg(Node *List, int e)
{
Node *NewNode = malloc(sizeof(Node));
NewNode->Element = e;
if(IsEmpty(List))
NewNode->Next = NULL;
else
NewNode->Next = List->Next;
List->Next = NewNode;
}
void InsertLast(Node *List, int e)
{
Node *NewNode = malloc(sizeof(Node));
Node *Position;
NewNode->Element = e;
NewNode->Next = NULL;
if(IsEmpty(List))
List->Next = NewNode;
```

```
else
{
Position = List;
while(Position->Next != NULL)
Position = Position->Next;
Position->Next = NewNode;
}
}
void InsertMid(Node *List, int p, int e)
{
Node *NewNode = malloc(sizeof(Node));
Node *Position;
Position = Find(List, p);
NewNode->Element = e;
NewNode->Next = Position->Next;
Position->Next = NewNode;
}
void DeleteBeg(Node *List)
{
if(!IsEmpty(List))
{
Node *TempNode;
TempNode = List->Next;
List->Next = TempNode->Next;
printf("The deleted item is %d\n", TempNode->Element);
free(TempNode);
}
else
printf("List is empty...!\n");
}
void DeleteEnd(Node *List)
{
if(!IsEmpty(List))
{
Node *Position;
Node *TempNode;
Position = List;
while(Position->Next->Next != NULL)
Position = Position->Next;
TempNode = Position->Next;
Position->Next = NULL;
printf("The deleted item is %d\n", TempNode->Element);
free(TempNode);
}
else
printf("List is empty...!\n");
}
void DeleteMid(Node *List, int e)
{
if(!IsEmpty(List))
{
Node *Position;
Node *TempNode;
```

```
Position = FindPrevious(List, e);
if(!IsLast(Position))
{
TempNode = Position->Next;
Position->Next = TempNode->Next;
printf("The deleted item is %d\n", TempNode->Element);
free(TempNode);
}
}
else
printf("List is empty...!\n");
}
void Traverse(Node *List)
{
if(!IsEmpty(List))
{
Node *Position;
Position = List;
while(Position->Next != NULL)
{
Position = Position->Next;
printf("%d\t", Position->Element);
}
printf("\n");
}
else
printf("List is empty...!\n");
}
```

OUTPUT

```
1.Insert Beg
2.Insert Middle
3.Insert End
4.Delete Beg
5.Delete Middle
6.Delete End
7.Find
8.Traverse
9.Exit
Enter your choice : 1
Enter the element : 40
Enter your choice : 1
Enter the element : 30
Enter your choice : 1
Enter the element : 20
Enter your choice : 1
Enter the element : 10
Enter your choice : 8
10 20 30 40
Enter your choice : 7
Enter the element : 30
Element found...!
Enter your choice : 1
Enter the element : 5
```

Enter your choice : 8
5 10 20 30 40
Enter your choice : 3
Enter the element : 45
Enter your choice : 8
5 10 20 30 40 45
Enter your choice : 2
Enter the position element : 20
Enter the element : 25
Enter your choice : 8
5 10 20 25 30 40 45
Enter your choice : 4
The deleted item is 5
Enter your choice : 8
10 20 25 30 40 45
Enter your choice : 6
The deleted item is 45
Enter your choice : 8
10 20 25 30 40
Enter your choice : 5
Enter the element : 30
The deleted item is 30
Enter your choice : 8
10 20 25 40
Enter your choice : 9

Algorithm:

1. **Start**
2. **Create a structure and functions for each operations**
3. **Display the main menu**
4. **Read user choice**
5. **Execute choice operation**
6. **Display operation completion**
7. **Back to main menu**
8. **Check for exit, if no Execute the operation for the given choice**
9. **Otherwise end Program:**

Ex. No.: 02	Implementation of Doubly Linked List	Date: 19.03.2024

Write a C program to implement the following operations on Doubly Linked List.

(i) Insertion

(ii) Deletion

(iii) Search

(iv) Display

CODING:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
struct node *Prev;
int Element;
struct node *Next;
};
typedef struct node Node;
int IsEmpty(Node *List);
int IsLast(Node *Position);
Node *Find(Node *List, int x);
void InsertBeg(Node *List, int e);
void InsertLast(Node *List, int e);
void InsertMid(Node *List, int p, int e);
void DeleteBeg(Node *List);
void DeleteEnd(Node *List);
void DeleteMid(Node *List, int e);
void Traverse(Node *List);
int main()
{
Node *List = malloc(sizeof(Node));
List->Prev = NULL;
List->Next = NULL;
Node *Position;
int ch, e, p;
printf("1.Insert Beg \n2.Insert Middle \n3.Insert End");
printf("\n4.Delete Beg \n5.Delete Middle \n6.Delete End");
printf("\n7.Find \n8.Traverse \n9.Exit\n");
do
{
printf("Enter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
printf("Enter the element : ");
scanf("%d", &e);
InsertBeg(List, e);
```

```
break;
case 2:
printf("Enter the position element : ");
scanf("%d", &p);
printf("Enter the element : ");
scanf("%d", &e);
InsertMid(List, p, e);
break;
case 3:
printf("Enter the element : ");
scanf("%d", &e);
InsertLast(List, e);
break;
case 4:
DeleteBeg(List);
break;
case 5:
printf("Enter the element : ");
scanf("%d", &e);
DeleteMid(List, e);
break;
case 6:
DeleteEnd(List);
break;
case 7:
printf("Enter the element : ");
scanf("%d", &e);
Position = Find(List, e);
if(Position != NULL)
printf("Element found...!\n");
else
printf("Element not found...!\n");
break;
case 8:
Traverse(List);
break;
}
} while(ch<= 8);
return 0;
}
int IsEmpty(Node *List)
{
if(List->Next == NULL)
return 1;
else
return 0;
}
int IsLast(Node *Position)
{
if(Position->Next == NULL)
return 1;
```

```
else
return 0;
}
Node *Find(Node *List, int x)
{
Node *Position;
Position = List->Next;
while(Position != NULL && Position->Element != x)
Position = Position->Next;
return Position;
}
void InsertBeg(Node *List, int e)
{
Node *NewNode = malloc(sizeof(Node));
NewNode->Element = e;
if(IsEmpty(List))
NewNode->Next = NULL;
else
{
NewNode->Next = List->Next;
NewNode->Next->Prev = NewNode;
}
NewNode->Prev = List;
List->Next = NewNode;
}
void InsertLast(Node *List, int e)
{
Node *NewNode = malloc(sizeof(Node));
Node *Position;
NewNode->Element = e;
NewNode->Next = NULL;
if(IsEmpty(List))
{
NewNode->Prev = List;
List->Next = NewNode;
}
else
{
Position = List;
while(Position->Next != NULL)
Position = Position->Next;
Position->Next = NewNode;
NewNode->Prev = Position;
}
}
void InsertMid(Node *List, int p, int e)
{
Node *NewNode = malloc(sizeof(Node));
Node *Position;
Position = Find(List, p);
NewNode->Element = e;
```

```
NewNode->Next = Position->Next;
Position->Next->Prev = NewNode;
Position->Next = NewNode;
NewNode->Prev = Position;
}
void DeleteBeg(Node *List)
{
if(!IsEmpty(List))
{
Node *TempNode;
TempNode = List->Next;
List->Next = TempNode->Next;
if(List->Next != NULL)
TempNode->Next->Prev = List;
printf("The deleted item is %d\n", TempNode->Element);
free(TempNode);
}
else
printf("List is empty...!\n");
}
void DeleteEnd(Node *List)
{
if(!IsEmpty(List))
{
Node *Position;
Node *TempNode;
Position = List;
while(Position->Next != NULL)
Position = Position->Next;
TempNode = Position;
Position->Prev->Next = NULL;
printf("The deleted item is %d\n", TempNode->Element);
free(TempNode);
}
else
printf("List is empty...!\n");
}
void DeleteMid(Node *List, int e)
{
if(!IsEmpty(List))
{
Node *Position;
Node *TempNode;
Position = Find(List, e);
if(!IsLast(Position))
{
TempNode = Position;
Position->Prev->Next = Position->Next;
Position->Next->Prev = Position->Prev;
printf("The deleted item is %d\n", TempNode->Element);
free(TempNode);
```

```
}
}
else
printf("List is empty...!\n");
}
void Traverse(Node *List)
{
if(!IsEmpty(List))
{
Node *Position;
Position = List;
while(Position->Next != NULL)
{
Position = Position->Next;
printf("%d\t", Position->Element);
}
printf("\n");
}
else
printf("List is empty...!\n");
}
```

OUTPUT

```
1.Insert Beg
2.Insert Middle
3.Insert End
4.Delete Beg
5.Delete Middle
6.Delete End
7.Find
8.Traverse
9.Exit
Enter your choice : 1
Enter the element : 40
Enter your choice : 1
Enter the element : 30
Enter your choice : 1
Enter the element : 20
Enter your choice : 1
Enter the element : 10
Enter your choice : 8
10 20 30 40
Enter your choice : 7
Enter the element : 30
Element found...!
Enter your choice : 1
Enter the element : 5
Enter your choice : 8
5 10 20 30 40
Enter your choice : 3
Enter the element : 45
Enter your choice : 8
```

5 10 20 30 40 45
Enter your choice : 2
Enter the position element : 20
Enter the element : 25
Enter your choice : 8
5 10 20 25 30 40 45
Enter your choice : 4
The deleted item is 5
Enter your choice : 810 20 25 30 40 45
Enter your choice : 6
The deleted item is 45
Enter your choice : 8
10 20 25 30 40
Enter your choice : 5
Enter the element : 30
The deleted item is 30
Enter your choice : 8
10 20 25 40
Enter your choice : 9

Algorithm:

1. **Start**
2. **Create a structure and functions for each operations**
3. **Declare the variables**
4. **Create a do-while loop to display the menu and execute operations based on your input until the user chooses to exit**
5. **Inside the loop display the menu options**
6. **Prompt the user to enter their choice**
7. **Use switch statement to perform different operations based on the user's choice and display it.**
8. **Repeat the loop until the user chooses to exit**
9. **Exit**

Write a C program to implement the following operations on Singly Linked List.

(i) Polynomial Addition

(ii) Polynomial Subtraction

(iii) Polynomial Multiplication

Algorithm:

1. **Start**
2. **Define structure**
3. **Create term functions**
4. **Insert term into the polynomial and add, subtract and multiplication these polynomial and display it**
5. **End Program**

CODING:

(i) Polynomial Addition

```
#include <stdio.h>
#include <stdlib.h>
struct poly
{
int coeff;
int pow;
struct poly *Next;
};
typedef struct poly Poly;
void Create(Poly *List);
void Display(Poly *List);
void Addition(Poly *Poly1, Poly *Poly2, Poly *Result);
int main()
{
Poly *Poly1 = malloc(sizeof(Poly));
Poly *Poly2 = malloc(sizeof(Poly));
Poly *Result = malloc(sizeof(Poly));
Poly1->Next = NULL;
Poly2->Next = NULL;
printf("Enter the values for first polynomial :\n");
Create(Poly1);

printf("The polynomial equation is : ");
Display(Poly1);
printf("\nEnter the values for second polynomial :\n");
Create(Poly2);
printf("The polynomial equation is : ");
Display(Poly2);
Addition(Poly1, Poly2, Result);
```

```
printf("\nThe polynomial equation addition result is : ");
Display(Result);
return 0;
}
void Create(Poly *List)
{
int choice;
Poly *Position, *NewNode;
Position = List;
do
{
NewNode = malloc(sizeof(Poly));
printf("Enter the coefficient : ");
scanf("%d", &NewNode->coeff);
printf("Enter the power : ");
scanf("%d", &NewNode->pow);
NewNode->Next = NULL;
Position->Next = NewNode;
Position = NewNode;
printf("Enter 1 to continue : ");
scanf("%d", &choice);
} while(choice == 1);
}
void Display(Poly *List)
{
Poly *Position;
Position = List->Next;
while(Position != NULL)
{
printf("%dx^%d", Position->coeff, Position->pow);
Position = Position->Next;
if(Position != NULL && Position->coeff> 0)
{
printf("+");
}
}
}
void Addition(Poly *Poly1, Poly *Poly2, Poly *Result)
{
Poly *Position;
Poly *NewNode;
Poly1 = Poly1->Next;
Poly2 = Poly2->Next;
Result->Next = NULL;
Position = Result;
while(Poly1 != NULL && Poly2 != NULL)
{
NewNode = malloc(sizeof(Poly));
if(Poly1->pow == Poly2->pow)
{
NewNode->coeff = Poly1->coeff + Poly2->coeff;
NewNode->pow = Poly1->pow;
Poly1 = Poly1->Next;
```

```
Poly2 = Poly2->Next;
}
else if(Poly1->pow > Poly2->pow)
{
NewNode->coeff = Poly1->coeff;
NewNode->pow = Poly1->pow;
Poly1 = Poly1->Next;
}
else if(Poly1->pow < Poly2->pow)
{
NewNode->coeff = Poly2->coeff;
NewNode->pow = Poly2->pow;
Poly2 = Poly2->Next;
}
NewNode->Next = NULL;
Position->Next = NewNode;
Position = NewNode;
}
while(Poly1 != NULL || Poly2 != NULL)
{
NewNode = malloc(sizeof(Poly));
if(Poly1 != NULL)
{
NewNode->coeff = Poly1->coeff;
NewNode->pow = Poly1->pow;
Poly1 = Poly1->Next;
}
if(Poly2 != NULL)
{
NewNode->coeff = Poly2->coeff;
NewNode->pow = Poly2->pow;
Poly2 = Poly2->Next;
}
NewNode->Next = NULL;
Position->Next = NewNode;
Position = NewNode;
}
}
```

OUTPUT

```
Enter the values for first polynomial :
Enter the coefficient : 2
Enter the power : 2
Enter 1 to continue : 1
Enter the coefficient : 6
Enter the power : 1
Enter 1 to continue : 1
Enter the coefficient : 5
Enter the power : 0
Enter 1 to continue : 0
The polynomial equation is : 2x^2+6x^1+5x^0
Enter the values for second polynomial :
Enter the coefficient : 3
Enter the power : 2
```

Enter 1 to continue : 1
Enter the coefficient : -2
Enter the power : 1
Enter 1 to continue : 1
Enter the coefficient : -1
Enter the power : 0
Enter 1 to continue : 0
The polynomial equation is : 3x^2-2x^1-1x^0
The polynomial equation addition result is : 5x^2+4x^1+4x^0

(ii) Polynomial Subtraction

CODING:

```
#include <stdio.h>
#include <stdlib.h>
struct poly
{
int coeff;
int pow;
struct poly *Next;
};
typedef struct poly Poly;
void Create(Poly *List);
void Display(Poly *List);
void Subtraction(Poly *Poly1, Poly *Poly2, Poly *Result);
int main()
{
Poly *Poly1 = malloc(sizeof(Poly));
Poly *Poly2 = malloc(sizeof(Poly));
Poly *Result = malloc(sizeof(Poly));
Poly1->Next = NULL;
Poly2->Next = NULL;
printf("Enter the values for first polynomial :\n");
Create(Poly1);
printf("The polynomial equation is : ");
Display(Poly1);
printf("\nEnter the values for second polynomial :\n");
Create(Poly2);
printf("The polynomial equation is : ");
Display(Poly2);
Subtraction(Poly1, Poly2, Result);
printf("\nThe polynomial equation subtraction result is : ");
Display(Result);
return 0;
}
void Create(Poly *List)
{
int choice;
Poly *Position, *NewNode;
Position = List;
do
{
NewNode = malloc(sizeof(Poly));
```

```
printf("Enter the coefficient : ");
scanf("%d", &NewNode->coeff);
printf("Enter the power : ");
scanf("%d", &NewNode->pow);
NewNode->Next = NULL;
Position->Next = NewNode;
Position = NewNode;
printf("Enter 1 to continue : ");
scanf("%d", &choice);
} while(choice == 1);
}
void Display(Poly *List)
{
Poly *Position;
Position = List->Next;
while(Position != NULL)
{
printf("%dx^%d", Position->coeff, Position->pow);
Position = Position->Next;
if(Position != NULL && Position->coeff> 0)
{
printf("+");
}
}
}
void Subtraction(Poly *Poly1, Poly *Poly2, Poly *Result)
{
Poly *Position;
Poly *NewNode;
Poly1 = Poly1->Next;
Poly2 = Poly2->Next;
Result->Next = NULL;
Position = Result;
while(Poly1 != NULL && Poly2 != NULL)
{
NewNode = malloc(sizeof(Poly));
if(Poly1->pow == Poly2->pow)
{
NewNode->coeff = Poly1->coeff - Poly2->coeff;
NewNode->pow = Poly1->pow;
Poly1 = Poly1->Next;
Poly2 = Poly2->Next;
}
else if(Poly1->pow > Poly2->pow)
{
NewNode->coeff = Poly1->coeff;
NewNode->pow = Poly1->pow;
Poly1 = Poly1->Next;
}
else if(Poly1->pow < Poly2->pow)
{
NewNode->coeff = -(Poly2->coeff);
NewNode->pow = Poly2->pow;
```

```
Poly2 = Poly2->Next;
}
NewNode->Next = NULL;
Position->Next = NewNode;
Position = NewNode;
}
while(Poly1 != NULL || Poly2 != NULL)
{
NewNode = malloc(sizeof(Poly));
if(Poly1 != NULL)
{
NewNode->coeff = Poly1->coeff;
NewNode->pow = Poly1->pow;
Poly1 = Poly1->Next;
}
if(Poly2 != NULL)
{
NewNode->coeff = -(Poly2->coeff);
NewNode->pow = Poly2->pow;
Poly2 = Poly2->Next;
}
NewNode->Next = NULL;
Position->Next = NewNode;
Position = NewNode;
}
}
```

OUTPUT

```
Enter the values for first polynomial :
Enter the coefficient : 3
Enter the power : 2
Enter 1 to continue : 1
Enter the coefficient : 4
Enter the power : 1
Enter 1 to continue : 1
Enter the coefficient : -2
Enter the power : 0
Enter 1 to continue : 0
The polynomial equation is : 3x^2+4x^1-2x^0
Enter the values for second polynomial :
Enter the coefficient : -7
Enter the power : 2
Enter 1 to continue : 1
Enter the coefficient : -10
Enter the power : 1
Enter 1 to continue : 1
Enter the coefficient : 17
Enter the power : 0
Enter 1 to continue : 0
The polynomial equation is : -7x^2-10x^1+17x^0
The polynomial equation subtraction result is : 10x^2+14x^1-19x^0
```

(iii) Polynomial differentiation:

```
#include <stdio.h>
#include <stdlib.h>
struct poly
{
int coeff;
int pow;
struct poly *Next;
};
typedef struct poly Poly;
void Create(Poly *List);
void Display(Poly *List);
void Differentiation(Poly *Poly1, Poly *Result);
int main()
{
Poly *Poly1 = malloc(sizeof(Poly));
Poly *Result = malloc(sizeof(Poly));
Poly1->Next = NULL;
printf("Enter the values for polynomial :\n");
Create(Poly1);
printf("The polynomial equation is : ");
Display(Poly1);
Differentiation(Poly1, Result);
printf("\nThe polynomial differentiation equation is : ");
Display(Result);
return 0;
}
void Create(Poly *List)
{
int choice;
Poly *Position, *NewNode;
Position = List;
do
{
NewNode = malloc(sizeof(Poly));
printf("Enter the coefficient : ");
scanf("%d", &NewNode->coeff);
```

```
printf("Enter the power : ");
scanf("%d", &NewNode->pow);
NewNode->Next = NULL;
Position->Next = NewNode;
Position = NewNode;
printf("Enter 1 to continue : ");
scanf("%d", &choice);
} while(choice == 1);
}
void Display(Poly *List)
{
Poly *Position;
Position = List->Next;
while(Position != NULL && Position->pow >= 0)
{
printf("%dx^%d", Position->coeff, Position->pow);
Position = Position->Next;
if(Position != NULL && Position->coeff> 0)
{
printf("+");
}
}
}
void Differentiation(Poly *Poly1, Poly *Result)
{
Poly *Position;
Poly *NewNode;
Poly1 = Poly1->Next;
Result->Next = NULL;
Position = Result;
while(Poly1 != NULL)
{
NewNode = malloc(sizeof(Poly));
NewNode->coeff = Poly1->coeff * Poly1->pow;
NewNode->pow = Poly1->pow - 1;
Poly1 = Poly1->Next;
NewNode->Next = NULL;
Position->Next = NewNode;
```

```
Position = NewNode;
}
}
```

Output

```
Enter the values for polynomial :
Enter the coefficient : 3
Enter the power : 5
Enter 1 to continue : 1
Enter the coefficient : -2
Enter the power : 3
Enter 1 to continue : 1
Enter the coefficient : 1
Enter the power : 1
Enter 1 to continue : 1
Enter the coefficient : 5
Enter the power : 0
Enter 1 to continue : 0
The polynomial equation is : 3x^5-2x^3+1x^1+5x^0
The polynomial differentiation equation is : 15x^4-6x^2+1x^0
```

Ex. No.: 04	Implementation of Stack using Array and Linked List Implementation	Date: 02.04.2024

Write a C program to implement a stack using Array and linked List implementation and execute the following operation on stack.

(i) Push an element into a stack

(ii) Pop an element from a stack

(iii) Return the Top most element from a stack

(iv) Display the elements in a stack

Algorithm:

1.Start

2. Create a structure and functions for the given operations

3.Initialize stack array with capacity and top=-1

4.To push an element into a stack read the data to be pushed. If the top is equal to capacity-1 display stack overflow. Otherwise increment the top and push the data onto stack at index top

5.To pop an element from a stack if the top is equal to -1 display as stack underflow. Otherwise pop data from stack at index top the decrement the top and display the popped data

6. To return the top most element from a stack if the top is equal to -1 display stack is empty. Otherwise display data at index top

7.After these operations display all elements in stack from top to 0

8.End

CODING:

ARRAY IMPLEMENTATION OF STACK

```
#include <stdio.h>

#define MAX 5

int Stack[MAX], top = -1;

int IsFull();

int IsEmpty();

void Push(int ele);

void Pop();

void Top();

void Display();
```

```
int main()
{
int ch, e;
do
{
printf("1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT");
printf("\nEnter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
printf("Enter the element : ");
scanf("%d", &e);
Push(e);
break;
case 2:
Pop();
break;
case 3:
Top();
break;
case 4:
Display();
break;
}
} while(ch<= 4);
return 0;
}
int IsFull()
{
if(top == MAX - 1)
return 1;
```

```
else
return 0;
}
int IsEmpty()
{
if(top == -1)
return 1;
else
return 0;
}
void Push(int ele)
{
if(IsFull())
printf("Stack Overflow...!\n");
else
{
top = top + 1;
Stack[top] = ele;
}
}
void Pop()
{
if(IsEmpty())
printf("Stack Underflow...!\n");
else
{
printf("%d\n", Stack[top]);
top = top - 1;
}
}
void Top()
{
```

```
if(IsEmpty())
printf("Stack Underflow...!\n");
else
printf("%d\n", Stack[top]);
}
void Display()
{
int i;
if(IsEmpty())
printf("Stack Underflow...!\n");
else
{
for(i = top; i>= 0; i--)
printf("%d\t", Stack[i]);
printf("\n");
}
}
```

Output

```
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 10
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 20
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 30
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 40
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
```

Enter the element : 50

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 1

Enter the element : 60

Stack Overflow...!

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 4

50 40 30 20 10

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 3

50

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

50

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

40

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

30

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

20

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

10

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

Stack Underflow...!

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 5

LINKED LIST IMPLEMENTATION OF STACK

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int Element;
struct node *Next;
}*List = NULL;
typedef struct node Stack;
int IsEmpty();
void Push(int e);
void Pop();
void Top();
void Display();
int main()
{
int ch, e;
do
{
printf("1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT");
printf("\nEnter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
printf("Enter the element : ");
scanf("%d", &e);
Push(e);
break;
case 2:
Pop();
break;
```

```
case 3:
Top();
break;
case 4:
Display();
break;
}
} while(ch<= 4);
return 0;
}
int IsEmpty()
{
if(List == NULL)
return 1;
else
return 0;
}
void Push(int e)
{
Stack *NewNode = malloc(sizeof(Stack));
NewNode->Element = e;
if(IsEmpty())
NewNode->Next = NULL;
else
NewNode->Next = List;
List = NewNode;
}
void Pop()
{
if(IsEmpty())
printf("Stack is Underflow...!\n");
else
{
```

```
Stack *TempNode;
TempNode = List;
List = List->Next;
printf("%d\n", TempNode->Element);
free(TempNode);
}
}
void Top()
{
if(IsEmpty())
printf("Stack is Underflow...!\n");
else
printf("%d\n", List->Element);
}
void Display()
{
if(IsEmpty())
printf("Stack is Underflow...!\n");
else
{
Stack *Position;
Position = List;
while(Position != NULL)
{
printf("%d\t", Position->Element);
Position = Position->Next;
}
printf("\n");
}
}
```

Output

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 1

Enter the element : 10

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 1

Enter the element : 20

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 1

Enter the element : 30

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 1

Enter the element : 40

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 1

Enter the element : 50

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 4

50 40 30 20 10

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 3

50

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

50

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

40

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

30

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

20

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

10

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 2

Stack is Underflow...!

1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT

Enter your choice : 5

Ex. No.: 05	Infix to Postfix Conversion	Date: 16.04.2024

Write a C program to perform infix to postfix conversion using stack.

Algorithm:

1. **Start**
2. **Initialize variables and stack**
3. **Read infix expression from the user**
4. **Scan infix expression by character**
5. **If the character is an operand append it to the postfix expression**
6. **If the character is an open parenthesis push it onto a stack**
7. **If the character is a closed parenthesis pop and append operators from the stack until an open parenthesis is encountered.**
8. **If the character is an operator while the stack is not empty and precedence of the top operator is greater or equal to precedence of the scanned operator, pop and append top operator to postfix expression 9. Push scanned operator onto the stack**
10. **After scanning the entire infix expression pop and append all the operators from the stack**
11. **Output the postfix expression**
12. **End**

CODING:

PROGRAM (USING ARRAY)

```
#include <stdio.h>
#include <string.h>
#define MAX 20
int Stack[MAX], top = -1;
char expr[MAX], post[MAX];
void Push(char sym);
char Pop();
char Top();
int Priority(char sym);
int main()
{
int i;
printf("Enter the infix expression : ");
gets(expr);
for(i = 0; i<strlen(expr); i++)
```

```
{
if(expr[i] >= 'a' && expr[i] <= 'z')
printf("%c", expr[i]);
else if(expr[i] == '(')
Push(expr[i]);
else if(expr[i] == ')')
{
while(Top() != '(')
printf("%c", Pop());
Pop();
}
else
{
while(Priority(expr[i])<=Priority(Top()) && top!=-1)
printf("%c", Pop());
Push(expr[i]);
}
}
for(i = top; i>= 0; i--)
printf("%c", Pop());
return 0;
}
void Push(char sym)
{
top = top + 1;
Stack[top] = sym;
}
char Pop()
{
char e;
e = Stack[top];
top = top - 1;
return e;
}
char Top()
{
```

```
return Stack[top];;
}
int Priority(char sym)
{
int p = 0;
switch(sym)
{
case '(':
p = 0;
break;
case '+':
case '-':
p = 1;
break;
case '*':
case '/':
case '%':
p = 2;
break;
case '^':
p = 3;
break;
}
return p;
}
```

Output

```
Enter the infix expression : a/b^c+d*e-f*g
abc^/de*+fg*-
```

Ex. No.: 06	Evaluating Arithmetic Expression	Date: 16.04.2024

Write a C program to evaluate Arithmetic expression using stack.

Algorithm:

1. **Start**
2. **Create an empty stack to hold operands.**
3. **Initialize a variable top to -1 which represents the top of the stack**
4. **Read the input from user**
5. **Iterate through each character in the expression**
6. **If the character is a digit, convert the character to its integer value and push the integer into stack.**
7. **if the character is an operator, pop the top two operands from the stack and perform the corresponding operation.**
8. **Get the result and display it**
9. **End**

PROGRAM (USING ARRAY)

```
#include <stdio.h>
#include <string.h>
#define MAX 20
int Stack[MAX], top = -1;
char expr[MAX];
void Push(int ele);
int Pop();
int main()
{
int i, a, b, c, e;
printf("Enter the postfix expression : ");
gets(expr);
for(i = 0; i<strlen(expr); i++)
{
if(expr[i]=='+'||expr[i]=='-'||expr[i]=='*'||expr[i]=='/')
{
b = Pop();
a = Pop();
```

```
switch(expr[i])
{
case '+':
c = a + b;
Push(c);
break;
case '-':
c = a - b;
Push(c);
break;
case '*':
c = a * b;
Push(c);
break;
case '/':
c = a / b;
Push(c);
break;
}
}
else
{
printf("Enter the value of %c : ", expr[i]);
scanf("%d", &e);
Push(e);
}
}
printf("The result is %d", Pop());
return 0;
}
void Push(int ele)
{
top = top + 1;
```

```
Stack[top] = ele;
}
int Pop()
{
int e;
e = Stack[top];
top = top - 1;
return e;
}
```

Output

```
Enter the postfix expression :abc+*d*
Enter the value of a : 2
Enter the value of b : 3
Enter the value of c : 4
Enter the value of d : 5
The result is 70
```

Ex. No.: 07	Implementation of Queue using Array and Linked List Implementation	Date: 22.04.2024

Write a C program to implement a Queue using Array and linked List implementation and execute the following operation on stack.

(i) Enqueue

(ii) Dequeue

(iii) Display the elements in a Queue

Algorithm:

1. **Start**
2. **To enqueuer an element read the value**
3. **If rear is equal to MAX_SIZE-1, print Queue is full**
4. **Otherwise if front is -1, set front to 0 , increment rear by 1 and assign the value to queue[rear].**
5. **To dequeuer an element if front is -1 print Queue is emprty and return 1**
6. **Otherwise assign element as queue[front], increment front by 1**
7. **End**

PROGRAM

```
#include <stdio.h>

#define MAX 5

int Queue[MAX], front = -1, rear = -1;

int IsFull();

int IsEmpty();

void Enqueue(int ele);

void Dequeue();

void Display();

int main()

{

int ch, e;

do

{

printf("1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT");

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch(ch)
```

```
{
case 1:
printf("Enter the element : ");
scanf("%d", &e);
Enqueue(e);
break;
case 2:
Dequeue();
break;
case 3:
Display();
break;
}
} while(ch<= 3);
return 0;
}
int IsFull()
{
if(rear == MAX - 1)
return 1;
else
return 0;
}
int IsEmpty()
{
if(front == -1)
return 1;
else
return 0;
}
void Enqueue(int ele)
{
```

```
if(IsFull())
printf("Queue is Overflow...!\n");
else
{
rear = rear + 1;
Queue[rear] = ele;
if(front == -1)
front = 0;
}
}
void Dequeue()
{
if(IsEmpty())
printf("Queue is Underflow...!\n");
else
{
printf("%d\n", Queue[front]);
if(front == rear)
front = rear = -1;
else
front = front + 1;
}
}
void Display()
{
int i;
if(IsEmpty())
printf("Queue is Underflow...!\n");
else
{
for(i = front; i<= rear; i++)
printf("%d\t", Queue[i]);
printf("\n");
```

```
}
}
```

Output

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 1

Enter the element : 10

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 1

Enter the element : 20

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 1

Enter the element : 30

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 1

Enter the element : 40

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 1

Enter the element : 50

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 1

Enter the element : 60

Queue is Overflow...!

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 3

10 20 30 40 50

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

10

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

20

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

30

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

40

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

50

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

Queue is Underflow...!

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 3

Queue Underflow...!

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 4

LINKED LIST IMPLEMENTATION OF QUEUE

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int Element;
struct node *Next;
}*Front = NULL, *Rear = NULL;
typedef struct node Queue;
int IsEmpty(Queue *List);
void Enqueue(int e);
void Dequeue();
void Display();
int main()
{
int ch, e;
```

```
do
{
printf("1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT");
printf("\nEnter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
printf("Enter the element : ");
scanf("%d", &e);
Enqueue(e);
break;
case 2:
Dequeue();
break;
case 3:
Display();
break;
}
} while(ch<= 3);
return 0;
}
int IsEmpty(Queue *List)
{
if(List == NULL)
return 1;
else
return 0;
}
void Enqueue(int e)
{
Queue *NewNode = malloc(sizeof(Queue));
NewNode->Element = e;
```

```
NewNode->Next = NULL;
if(Rear == NULL)
Front = Rear = NewNode;
else
{
Rear->Next = NewNode;
Rear = NewNode;
}
}
void Dequeue()
{
if(IsEmpty(Front))
printf("Queue is Underflow...!\n");
else
{
Queue *TempNode;
TempNode = Front;
if(Front == Rear)
Front = Rear = NULL;
else
Front = Front->Next;
printf("%d\n", TempNode->Element);
free(TempNode);
}
}
void Display()
{
if(IsEmpty(Front))
printf("Queue is Underflow...!\n");
else
{
Queue *Position;
Position = Front;
```

```
while(Position != NULL)
{
printf("%d\t", Position->Element);
Position = Position->Next;
}
printf("\n");
}
}
```

Output

```
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 10
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 20
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 30
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 40
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 3
10 20 30 40 50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
10
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
20
```

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

30

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

40

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

50

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 2

Queue is Underflow...!

1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT

Enter your choice : 4

Ex. No.: 08	Tree Traversal	Date: 30.04.2024

Write a C program to implement a Binary tree and perform the following tree traversal operation.

(i) Inorder Traversal

(ii) Preorder Traversal

(iii) Postorder Traversal

Algorithm:

1. **Start**
2. **Create a node which contains data, left, right their member**
3. **Create 3 different types of functions to traversal in 3 different ways: inorder, preorder, postorder.**
4. **Call each functions and display the output**
5. **End**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
struct node *left;
int element;
struct node *right;
};
typedef struct node Node;
Node *Insert(Node *Tree, int e);
void Inorder(Node *Tree);
void Preorder(Node *Tree);
void Postorder(Node *Tree);
int main()
{
Node *Tree = NULL;
int n, i, e, ch;
printf("Enter number of nodes in the tree : ");
scanf("%d", &n);
```

```
printf("Enter the elements :\n");
for (i = 1; i<= n; i++)
{
scanf("%d", &e);
Tree = Insert(Tree, e);
}
do
{
printf("1. Inorder \n2. Preorder \n3. Postorder \n4. Exit\n");
printf("Enter your choice : ");
scanf("%d", &ch);
switch (ch)

{
case 1:
Inorder(Tree);
printf("\n");
break;
case 2:
Preorder(Tree);
printf("\n");
break;
case 3:
Postorder(Tree);
printf("\n");
break;
}
} while (ch<= 3);
return 0;
}
Node *Insert(Node *Tree, int e)
{
Node *NewNode = malloc(sizeof(Node));
```

```
if (Tree == NULL)
{
NewNode->element = e;
NewNode->left = NULL;
NewNode->right = NULL;
Tree = NewNode;
}
else if (e < Tree->element)
{
Tree->left = Insert(Tree->left, e);
}
else if (e > Tree->element)
{
Tree->right = Insert(Tree->right, e);
}
return Tree;
}
void Inorder(Node *Tree)
{
if (Tree != NULL)
{
Inorder(Tree->left);
printf("%d\t", Tree->element);
Inorder(Tree->right);
}
}
void Preorder(Node *Tree)
{
if (Tree != NULL)
{
printf("%d\t", Tree->element);
Preorder(Tree->left);
Preorder(Tree->right);
```

```
}
}
void Postorder(Node *Tree)
{
if (Tree != NULL)
{
Postorder(Tree->left);
Postorder(Tree->right);
printf("%d\t", Tree->element);
}
}
```

Ex. No.: 09	Implementation of Binary Search tree	Date: 30.04.2024

Write a C program to implement a Binary Search Tree and perform the following operations.

(i) Insert

(ii) Delete

(iii) Search

(iv) Display

Algorithm:

1. **Start**
2. **Defines a structure Node representing a node in the binary search tree. Each node contains data, left child pointer, and right child pointer.**
3. **Provides a function to create a new node with the given value and initialize its pointers.**
4. **To insert a new node into the binary search tree while maintaining the BST property. Create a function to check if the value is less than the current node's data, it traverses to the left subtree; otherwise, it traverses to the right subtree.**
5. **To delete a node from the binary search tree while preserving the BST property. Create a function to handle cases where the node has zero, one, or two children by finding the successor node and replacing the node to be deleted with it.**
6. **Create a function to find the node with the minimum value in a subtree, which is used in deletion operation.**
7. **To search for a value in the binary search tree, Create a recursive function to traverses the tree, comparing the value with each node's data until the value is found or the tree is exhausted.**
8. **Provide a function to perform an inorder traversal of the binary search tree, printing the nodes in sorted order.**
9. **End Program:**

```
#include <stdio.h>
#include <stdlib.h>

struct Node {       int
data;       struct Node*
left;
    struct Node* right;
};

struct Node* createNode(int
value) {          struct Node*
newNode
= (struct
Node*)malloc(sizeof(struct
Node)); newNode->data =
value;
newNode->left    =    NULL;
newNode->right = NULL;
    return newNode;
```

```
}

struct Node* insert(struct
Node* root, int value) {      if
(root == NULL) {         return
   createNode(value);
      }

      if (value < root->data) {
         root->left =
   insert(root->left,     value);
} else if (value > root>data) {
root->right    =    insert(root-
>right, value);
      }

      return root;
   }

   struct                  Node*
minValueNode(struct Node*
node)  {          struct  Node*
current =
   node;

      while (current && current-
>left != NULL) {         current =
current>left;
      }

      return current;
   }

   struct                  Node*
deleteNode(struct       Node*
root, int value) {     if (root ==
NULL) {
         return root;
      }

      if (value < root->data) {
root->left = deleteNode(root-
>left, value);    } else if (value
> root>data) {        root->right
=       deleteNode(root->right,
value);     } else {        if (root-
>left ==
   NULL) {
            struct Node* temp =
root->right;           free(root);
return temp;
```

```
        } else if (root->right
    == NULL) {
            struct Node* temp =
  root->left;         free(root);
  return temp;
        }

        struct Node* temp = minValueNode(root-
    >right);      root->data =
  temp>data;
        root->right  =  deleteNode(root-
  >right, temp->data);
      }

      return root;
    }

    struct Node* search(struct
  Node* root, int value) {     if
  (root == NULL || root>data ==
  value) {      return root;
      }

      if  (root->data  <  value)  {
  return search(root-
    >right, value);
      }

      return search(root->left,
    value);
    }

    void display(struct Node*
  root) {     if (root != NULL) {
  display(root->left);
  printf("%d   ",   root>data);
  display(root->right);
      }
    }

    int main() {      struct
  Node* root =
    NULL;
       root = insert(root, 50);
insert(root, 30);   insert(root, 20);
insert(root, 40);   insert(root, 70);
insert(root, 60);   insert(root, 80);

    printf("Binary  Search  Tree
  Inorder Traversal: ");
```

```
        display(root);
        printf("\n");

        root = deleteNode(root,
   20);           printf("Binary
Search      Tree      Inorder
Traversal after deleting 20:
");     display(root);
        printf("\n");

        struct Node* searchResult
= search(root, 30);
        if (searchResult != NULL) {
   printf("Element  30  found  in
the Binary
   Search Tree.\n");
        }           else            {
printf("Element 30 not found
in the Binary
   Search Tree.\n");
        }

        return 0;
   }
```

Ex. No.: 10	Implementation of AVL Tree	Date: 07.05.2024

Write a function in C program to insert a new node with a given value into an AVL tree. Ensure that the tree remains balanced after insertion by performing rotations if necessary. Repeat the above operation to delete a node from AVL tree.

Algorithm:

1. **Start**
2. **Defines a structure Node representing a node in the AVL tree, containing data, left and right child pointers, and height.**
3. **Create a function to calculate the height of a node and its balance factor.**
4. **Implement a function to create a new node with the given data and initial height.**
5. **Create a function to performs a right rotation to balance the tree.**
6. **Create a function to performs left rotation to balance the tree.**
7. **Implement a function to insert a node into the AVL tree while maintaining AVL property and performing rotations as needed.**
8. **For deletion, implements a function to find the node with the minimum value in a subtree and implements a function to delete a node from the AVL tree while maintaining AVL property and performing rotations as needed.**
9. **Provides a function to perform inorder traversal of the AVL tree, printing the nodes in sorted order**
10. **End**

Program:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
int data;      struct Node
*left;         struct Node
*right;
    int height;
} Node;

// Function to get the height of a node int
height(Node *node) {
    if (node == NULL)
return 0;      return node-
>height;
}

// Function to get the balance factor of a node
int balance_factor(Node *node) {     if (node ==
NULL)      return 0;
    return height(node->left) - height(node->right);
}
```

```
// Function to create a new node
Node* newNode(int data) {
    Node* node = (Node*)malloc(sizeof(Node));
node->data = data;    node->left = NULL;    node-
>right = NULL;    node->height = 1;
    return node;
}

// Function to perform a right rotation
Node* rotate_right(Node *y) {
   Node *x = y->left;
   Node *T2 = x->right;

   // Perform rotation   x->right
= y; y->left = T2;

   // Update heights     y->height = 1 + (height(y->left) > height(y->right) ?
height(y->left) : height(y-
>right));
    x->height = 1 + (height(x->left) > height(x->right) ? height(x->left) :
height(x>right));

    return x;
}

// Function to perform a left rotation
Node* rotate_left(Node *x) {
    Node *y = x->right;
    Node *T2 = y->left;

    // Perform rotation       y-
>left = x;
    x->right = T2;

    // Update heights
    x->height = 1 + (height(x->left) > height(x->right) ? height(x->left) : height(x-
>right));
    y->height = 1 + (height(y->left) > height(y->right) ? height(y->left) :
height(y>right));

    return y;
}

// Function to insert a node into AVL tree
Node* insert(Node *node, int data) {     if
(node == NULL)
        return newNode(data);

    if (data < node->data)
```

```
          node->left    =    insert(node->left,    data);
else if (data > node->data)
          node->right    =    insert(node->right,    data);
else // Duplicate keys not allowed
          return node;

      // Update height of current node       node->height = 1 + (height(node->left) >
height(node->right) ? height(node->left) : height(node->right));

      // Get the balance factor    int balance
= balance_factor(node);

      // Perform rotations if needed
      if (balance > 1 && data < node->left->data)
           return rotate_right(node);
      if (balance < -1 && data > node->right->data)
return rotate_left(node);    if (balance > 1 && data
>   node->left->data)   {                    node->left   =
rotate_left(node->left);
          return rotate_right(node);
      }
      if (balance < -1 && data < node->right->data) {
node->right = rotate_right(node->right);          return
rotate_left(node);
      }

      return node;
   }

   // Function to find the node with minimum value
   Node*  minValueNode(Node  *node)  {
Node* current = node;     while (current-
>left != NULL)           current = current-
>left;     return current;
   }

   // Function to delete a node from AVL tree Node*
deleteNode(Node *root, int data) {
      if (root == NULL)
          return root;

      if (data < root->data)
          root->left    =    deleteNode(root->left,    data);
else if (data > root->data)
          root->right    =    deleteNode(root->right,    data);
else {
          if  (root->left  ==  NULL  ||  root->right  ==  NULL)  {
Node *temp = root->left ? root->left : root->right;
```

```
            if (temp == NULL) {
temp = root;                root =
NULL;
            } else
               *root = *temp; // Copy the contents of the non-empty child

            free(temp);
         } else {
         Node *temp = minValueNode(root->right);            root-
 >data = temp->data;
           root->right = deleteNode(root->right, temp->data);
        }
     }
      if (root == NULL)
        return root;

   // Update height of current node     root->height = 1 + (height(root->left) >
height(root->right) ? height(root->left) : height(root->right));

      // Get the balance factor
      int balance = balance_factor(root);

      // Perform rotations if needed
      if (balance > 1 &&balance_factor(root->left) >= 0)
return rotate_right(root);
      if (balance > 1 &&balance_factor(root->left) < 0) {
         root->left = rotate_left(root->left);
return rotate_right(root);
      }
      if (balance < -1 &&balance_factor(root->right) <= 0)
return rotate_left(root);
      if (balance < -1 &&balance_factor(root->right) > 0) {
         root->right = rotate_right(root->right);
return rotate_left(root);
      }

      return root;
   }

   // Function to print AVL tree inorder
void inorder(Node *root) {     if (root !=
NULL) {               inorder(root->left);
printf("%d ", root->data);
   inorder(root->right);
      }
   }

   int main() {
      Node *root = NULL;
```

```
      //     Inserting     nodes
root    =    insert(root,    10);
root    =    insert(root,    20);
root    =    insert(root,    30);
root    =    insert(root,    40);
root = insert(root, 50);    root
= insert(root, 25);

   printf("Inorder traversal of the constructed AVL tree: ");
inorder(root);    printf("\n");
     //  Deleting  node       printf("Delete  node
30\n");          root   =   deleteNode(root,   30);
printf("Inorder traversal after deletion: ");
   inorder(root);
      printf("\n");

      return 0;
   }
```

:

Ex. No.: 11	Graph Traversal	Date: 14.05.2024

Write a C program to create a graph and perform a Breadth First Search and Depth First Search.

Algorithm:

1. **Start**
2. **Create a node which contains vertex and next as their members.**
3. **Allocates memory dynamically for nodes and the graph structure using malloc().**
4. **Create a graph with a specified number of vertices and initialize adjacency lists.**
5. **Create a function to add the edges between the vertices**
6. **Performs BFS traversal starting from a given vertex using a queue data structure to maintain order.**
7. **Conducts DFS traversal starting from a specified vertex, employing recursion to explore graph branches.**

8.Displays the adjacency list representation of the graph and the traversal sequences for both BFS and DFS.

9. End

Program:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

struct Node {       int
vertex;     struct Node*
next;
};

struct Node* createNode(int v);

struct Graph {          int
numVertices;           struct
Node** adjLists;
    int* visited;
};

struct Graph* createGraph(int vertices);

void addEdge(struct Graph* graph, int src, int dest);
void printGraph(struct Graph* graph);

void BFS(struct Graph* graph, int startVertex);
```

```
void DFS(struct Graph* graph, int startVertex);

int main() {
    struct Graph* graph = createGraph(4);
addEdge(graph, 0, 1);     addEdge(graph, 0,
2);                    addEdge(graph,    1,    2);
addEdge(graph, 2, 0);     addEdge(graph, 2,
3);    addEdge(graph, 3, 3);

    printf("Graph:\n");
  printGraph(graph);

  printf("\nBFS Traversal:\n");
  BFS(graph, 2);

  printf("\nDFS Traversal:\n");
  DFS(graph, 2);

    return 0;
}

struct Node* createNode(int v) {
    struct   Node*   newNode   =   (struct   Node*)malloc(sizeof(struct
Node));     newNode->vertex = v;     newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));       graph-
>numVertices = vertices;

    graph->adjLists = (struct Node**)malloc(vertices * sizeof(struct Node*));    graph-
>visited = (int*)malloc(vertices * sizeof(int));

    for (int i = 0; i< vertices; i++) {            graph-
>adjLists[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
struct Node* newNode = createNode(dest);       newNode-
>next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

  newNode = createNode(src);   newNode->next =
graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}
```

```
   void printGraph(struct Graph* graph) {    for (int
v = 0; v < graph->numVertices; v++) {         struct
Node* temp = graph->adjLists[v];
   printf("Vertex %d: ", v);             while
(temp) {                printf("%d -> ", temp-
>vertex);
            temp = temp->next;
         }
         printf("NULL\n");
      }
   }

   void BFS(struct Graph* graph, int startVertex) {
struct Node* queue[MAX];
      int front = 0, rear = 0;      queue[rear] =
createNode(startVertex);
      graph->visited[startVertex] = 1;

   printf("Visited %d\n", startVertex);

      while (front <= rear) {
         struct Node* currentNode = queue[front];
         front++;        while (currentNode) {           int
adjVertex = currentNode->vertex;            if (!graph-
>visited[adjVertex]) {              printf("Visited %d\n",
adjVertex);                              queue[++rear] =
createNode(adjVertex);
               graph->visited[adjVertex] = 1;
            }
   currentNode = currentNode->next;
         }
      }
   }

   void DFSUtil(struct Graph* graph, int vertex) {
struct Node* temp = graph->adjLists[vertex];
graph->visited[vertex] = 1;          printf("Visited
%d\n", vertex);
      while (temp) {        int adjVertex =
temp->vertex;                     if (!graph-
>visited[adjVertex]) {
   DFSUtil(graph, adjVertex);
         }
         temp = temp->next;
      }
   }

   void DFS(struct Graph* graph, int startVertex) {             graph-
>visited[startVertex] = 1;
   printf("Visited %d\n", startVertex);
```

```
    struct Node* temp = graph->adjLists[startVertex];

    while (temp) {        int adjVertex =
temp->vertex;                    if  (!graph-
>visited[adjVertex]) {
  DFSUtil(graph, adjVertex);
        }
        temp = temp->next;
    }
  }
```

Ex. No.: 12	Topological Sorting	Date: 21.05.2024

Write a C program to create a graph and display the ordering of vertices.

Algorithm:

1. **Start**
2. **Read the input from the user to create an adjacency matrix representing the graph.**
3. **Implements the DFS algorithm to traverse the graph recursively.**
4. **Create a recursive function to explores the graph starting from a given vertex and stops when all adjacent vertices have been visited.**
5. **After traversal, the program prints the ordering of vertices.**
6. **End**

Program:

```
#include <stdio.h>

#define MAX_VERTICES 10

int graph[MAX_VERTICES][MAX_VERTICES] = {0};
int visited[MAX_VERTICES] = {0}; int
vertices;

void createGraph() {
    int i, j;
printf("Enter the number of vertices: ");    scanf("%d",
 &vertices);
printf("Enter the adjacency matrix:\n");
    for (i = 0; i< vertices; i++) {    for
 (j   =   0;   j   <   vertices;   j++)   {
 scanf("%d", &graph[i][j]);
        }
    }
}

void dfs(int vertex) {
    int i;
printf("%d ", vertex);     visited[vertex]
= 1;    for (i = 0; i< vertices; i++) {        if
(graph[vertex][i]     &&     !visited[i])     {
dfs(i);
        }
    }
}

int main() {
    int i;
createGraph();
printf("Ordering of vertices after DFS traversal:\n");
```

```
        for (i = 0; i< vertices; i++) {
            if      (!visited[i])      {
    dfs(i);
            }
        }
        return 0;
    }
```

Ex. No.:	Graph Traversal	Date:

Writea C program to create a graph and find a minimum spanning tree using prims algorithm.

Algorithm:

1. **Start**
2. **Input the number of vertices and the adjacency matrix representing the graph.**

3.Find the vertex with the minimum key value among the vertices not yet included in the MST.

4. Initializes key values and mstset for all vertices , then iteratively select the vertex with the minimum key value and updates the key values of its adjacent vertices of a shorter edge is found.

5.Print the edges of the MST along with their weights.

6. End

Program:

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_VERTICES 10
#define INF 999999

int graph[MAX_VERTICES][MAX_VERTICES]; int
vertices;

void createGraph() {
    int i, j;
printf("Enter the number of vertices: ");
scanf("%d", &vertices);
printf("Enter the adjacency matrix:\n");
    for (i = 0; i< vertices; i++) {
for (j = 0; j < vertices; j++) {
scanf("%d", &graph[i][j]);
        }
    }
}
```

```
   int  findMinKey(int  key[],  bool  mstSet[])  {
int min = INF, min_index;      for (int v = 0; v <
vertices; v++) {          if (mstSet[v] == false &&
key[v] < min) {
              min = key[v];
   min_index = v;
         }
      }
      return min_index;
   }

   void    printMST(int    parent[])    {
   printf("Edge  \tWeight\n");       for
   (int i = 1; i< vertices; i++) {
   printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
      }
   }

   void primMST() {      int
parent[vertices];         int
key[vertices];
      bool mstSet[vertices];

      for  (int  i  =  0;  i<  vertices;  i++)  {
key[i] = INF;
   mstSet[i] = false;
      }

   key[0] = 0; // Make key 0 so that this vertex is picked as the first vertex    parent[0]
= -1; // First node is always root of MST

     for (int count = 0; count < vertices - 1; count++) {
int u = findMinKey(key, mstSet);
      mstSet[u] = true;
         for (int v = 0; v < vertices; v++) {
            if  (graph[u][v]  &&mstSet[v]  ==  false  &&  graph[u][v]  <  key[v])  {
    parent[v] = u;
               key[v] = graph[u][v];
            }
         }
      }
   printMST(parent);
   }

   int             main()             {
createGraph();     primMST();
      return 0;
   }
```

EX NO.:14 **GRAPH TRAVERSAL.** DATE: 21.05.2024

Write a C program to create a graph and find the shortest path using Dijikstra's Algorithm.

Algorithm:

1. **Start**
2. **The main function calls create graph funcion.**
3. **Input the number of vertices and the adjacency matrix representing the graph.**
4. **find the vertex ward the minimum distance from the source vertex among the vertices.**
5. **Then at implements Dijkstra's algorithm, it initializes distance values and sptset for all vertices, then iteratively updates the distance values unit all vertices are included in the shortest part pree.**
6. **Display the Output**
7. **End**

Program:

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_VERTICES 10
#define INF 999999

int graph[MAX_VERTICES][MAX_VERTICES]; int
vertices;

void createGraph() {
    int i, j;
printf("Enter the number of vertices: ");
scanf("%d", &vertices);
printf("Enter the adjacency matrix:\n");
    for (i = 0; i< vertices; i++) {
for (j = 0; j < vertices; j++) {
scanf("%d", &graph[i][j]);
        }
    }
}

int minDistance(int dist[], bool sptSet[]) {
int min = INF, min_index;    for (int v = 0; v <
vertices; v++) {          if (sptSet[v] == false
&&dist[v] <= min) {         min = dist[v];
min_index = v;
        }
    }
    return min_index;
}
```

```
   void printSolution(int dist[]) {
   printf("Vertex \t Distance from Source\n");
      for (int i = 0; i< vertices; i++) {
   printf("%d \t %d\n", i, dist[i]);
      }
   }

   void   dijkstra(int   src)   {
int dist[vertices];
      bool sptSet[vertices];

      for (int i = 0; i< vertices; i++) {  dist[i]
= INF;
   sptSet[i] = false;
      }

   dist[src] = 0;

      for (int count = 0; count < vertices - 1; count++) {
int u = minDistance(dist, sptSet);
   sptSet[u] = true;

         for (int v = 0; v < vertices; v++) {
            if  (!sptSet[v]  &&  graph[u][v]  &&dist[u]  !=  INF  &&dist[u]  +  graph[u][v]
<dist[v])
   {
   dist[v] = dist[u] + graph[u][v];
            }
         }
      }

   printSolution(dist);
   }

   int main() {
      createGraph();    int
    source;
      printf("Enter   the   source   vertex:   ");
    scanf("%d", &source); dijkstra(source);
      return 0;
   }
```

Ex. No.: 15	Sorting	Date: 04.05.2024

Write a C program to take n numbers and sort the numbers in ascending order. Try to implement the same using following sorting techniques.

1. **Quick Sort**
2. **Merge Sort**

Algorithm:

Algorithm:

1. **Start**
2. **Read the number of elements n from the user**
3. **Read n elements into array**
4. **Sort the array using Quick sort function and print the sorted array**
5. **sort the array using merge sort function and print the sorted array**
6. **End**

Program:

```
#include <stdio.h>
#include <stdlib.h>

void swap(int *a, int *b) {
int temp = *a;     *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high) {
int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {  i++;
            swap(&arr[i], &arr[j]);
        }
    }
swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high) {
if (low < high) {
        int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);
    quickSort(arr, pi + 1, high);
    }
```

```
   }

   void merge(int arr[], int l, int m, int r) {
int i, j, k;     int n1 = m - l + 1;
       int n2 = r - m;

       int L[n1], R[n2];

       for  (i  =  0;  i<  n1;  i++)
L[i] = arr[l + i];      for (j = 0; j
< n2; j++)         R[j] = arr[m +
1 + j];
   i = 0;     j =
0;
       k = l;     while (i< n1 && j <
n2) {              if (L[i] <= R[j]) {
arr[k] = L[i];             i++;        }
else {                   arr[k] = R[j];
j++;        }        k++;
       }

       while   (i<   n1)   {
arr[k] = L[i];        i++;
k++;
       }

       while  (j  <  n2)  {
arr[k] = R[j];        j++;
k++;
       }
   }

   void mergeSort(int arr[], int l, int r) {
if (l < r) {        int m = l + (r - l) / 2;

   mergeSort(arr, l, m);
   mergeSort(arr, m + 1, r);     merge(arr, l, m, r);
       }
   }

   int main() {
       int n;
   printf("Enter     the     number     of     elements:     ");
scanf("%d", &n);

       int arr[n];
   printf("Enter %d elements:\n", n);
       for (int i = 0; i< n; i++) {
   scanf("%d", &arr[i]);
       }
```

```
   printf("\nSorting using Quick Sort:\n");
quickSort(arr, 0, n - 1);     for (int i = 0; i< n;
i++) {        printf("%d ", arr[i]);
      }

   printf("\n\nSorting using Merge Sort:\n");
mergeSort(arr, 0, n - 1);     for (int i = 0; i< n;
i++) {        printf("%d ", arr[i]);
      }

      return 0;
   }
```

Ex. No.: 16	Hashing	Date: 04.05.2024

Write a C program to create a hash table and perform collision resolution using the following techniques.

(i) Open addressing

(ii) Closed Addressing

(iii) Rehashing

Algorithm:

1. **Start**
2. **Read data from the user**
3. **Allocate memory fir a new node**
4. **Set the data field of the new node to the input data**
5. **Set the next pointer of the new node to the NULL and return the pointer to the new node**
6. **Read key from the user**
7. **Calculate the index by taking the modulo of the key with table size and return the calculated index**
8. **Input the table and calculate the index using the hash function**
9. **Iterate until an empty slot is found in the table**
10. **Create a news node to the table at the calculated index and return the pointer to the inserted node**
11. **Assign the new miss to the table at the calculated index and return the pointer to the inserted node.**
12. **Repeat this to insert elements using closed addressing and rehashing**
13. **Then display the hash table using display function**
14. **End**

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define TABLE_SIZE 10

typedef struct Node {
int data;      struct Node*
next;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
if (newNode == NULL) {       printf("Memory allocation
failed!\n");
exit(1);
    }
```

```
   newNode->data = data;   newNode->next =
NULL;
       return newNode;
   }

   int hashFunction(int key) {
       return key % TABLE_SIZE;
   }

   Node* insertOpenAddressing(Node* table[], int key)
{    int index = hashFunction(key);    while (table[index]
!= NULL) {
           index = (index + 1) % TABLE_SIZE;
       }
       table[index]      =      createNode(key);
return table[index];
   }

   void displayHashTable(Node* table[]) {
printf("Hash Table:\n");     for (int i = 0; i<
TABLE_SIZE; i++) {
   printf("%d: ", i);        Node* current
= table[i];        while (current != NULL)
{            printf("%d ", current->data);
              current = current->next;
   }        printf("\n");
       }
   }

   Node* insertClosedAddressing(Node* table[], int key)
{     int index = hashFunction(key);     if (table[index] ==
NULL) {
           table[index] = createNode(key);
       } else {
           Node* newNode = createNode(key);   newNode->next
= table[index];
           table[index] = newNode;
       }
       return table[index];
   }

   int rehashFunction(int key, int attempt) {
// Double Hashing Technique
       return (hashFunction(key) + attempt * (7 - (key % 7))) % TABLE_SIZE;
   }

   Node* insertRehashing(Node* table[], int key) {
int index = hashFunction(key);     int attempt = 0;
while (table[index] != NULL) {
           attempt++;
```

```
        index = rehashFunction(key, attempt);
    }
    table[index] = createNode(key);
    return table[index];
}

int main() {
    Node* openAddressingTable[TABLE_SIZE] = {NULL};
    Node* closedAddressingTable[TABLE_SIZE] = {NULL};
    Node* rehashingTable[TABLE_SIZE] = {NULL};

    // Insert elements into hash tables
insertOpenAddressing(openAddressingTable, 10);
insertOpenAddressing(openAddressingTable, 20);
  insertOpenAddressing(openAddressingTable, 5);

  insertClosedAddressing(closedAddressingTable, 10);
insertClosedAddressing(closedAddressingTable, 20);
insertClosedAddressing(closedAddressingTable, 5);

  insertRehashing(rehashingTable, 10);
insertRehashing(rehashingTable, 20);
  insertRehashing(rehashingTable, 5);

    // Display hash tables
  displayHashTable(openAddressingTable);
displayHashTable(closedAddressingTable);
  displayHashTable(rehashingTable);

    return 0;
}
```

Rajalakshmi Institutions

Rajalakshmi Engineering College
Rajalakshmi Nagar Thandalam, Chennai - 602 105.
Phone : +91-44-67181111 , 67181112
Website : www.rajalakshmi.or g