SINGLY LINKED LIST:

```
#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
```

DOUBLY LINKED LIST:

```
#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 : 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
```

POLYNOMIAL:

(MANIPULATION)

```
#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
```

```
#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
```