Program 7

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

struct node {
   char name[11], usn[11], prog[4], ph[11];     int
sem;     struct node *link;
};

typedef struct node *PTR;

PTR createNode();
PTR Insertfront(PTR first);
PTR Insertrear(PTR first);
PTR Deletefront(PTR first); PTR
Deleterear(PTR first);
void display(PTR first);

int main() {     PTR first =
NULL;
   int choice;

   while (1) {
      printf("\n1.Insertfront \n2.Insertrear \n3.Deletefront \n4.Deleterear \n5.Display \n6.Exit\n");
scanf("%d", &choice);        switch (choice) {  // Fixed: 'switch' was wrong, it should be 'choice'            case
1:               first = Insertfront(first);
            break;           case 2:
first = Insertrear(first);
            break;           case 3:
first = Deletefront(first);
```

```
            break;          case 4:
first = Deleterear(first);
            break;          case
5:              display(first);
break;          case 6:
exit(0);          default:
printf("Invalid choice\n");
      }
   }
   return 0;
}

PTR createNode() {  // Fixed function definition     PTR
temp = (PTR)malloc(sizeof(struct node));     if (!temp)
{       exit(1);
   }

   printf("USN: ");     scanf("%s",
temp->usn);     printf("Name: ");
scanf("%s", temp->name);
printf("Program: ");     scanf("%s",
temp->prog);     printf("Semester:
");
   scanf("%d", &temp->sem);  // Fixed: Added '&' for scanf
   printf("Phone No: ");
   scanf("%s", temp->ph);

   temp->link = NULL;
   return temp;
}

PTR Insertfront(PTR first) {
   PTR temp = createNode();  // Fixed: Function name should match exactly     temp->link =
first;
```

```
   return temp;
}

PTR  Insertrear(PTR  first)  {        PTR
temp = createNode();     if (!first) {
       return temp;
   }

   PTR curr = first;     while
(curr->link) {
       curr = curr->link;  // Fixed: Corrected loop for traversing the list
   }
   curr->link = temp;     return
first;
}

PTR Deletefront(PTR first) {
   if (!first) {
       printf("SLL is Empty\n");        return
NULL;
   }
   PTR temp = first;     first =
first->link;
   printf("Deleted: %s\n", temp->name);
   free(temp);     return
first;
}

PTR Deleterear(PTR first) {
   if (!first) {
       printf("SLL is Empty\n");        return
NULL;
   }
```

```
   if (!first->link) {  // Only one node
      free(first);
      return NULL;
   }

   PTR prev = NULL;     PTR cur
= first;
   while (cur->link) {  // Traverse to the last node        prev =
cur;
      cur = cur->link;
   }

   prev->link = NULL;  // Remove the last node     printf("Deleted:
%s\n", cur->name);
   free(cur);     return
first;
}

void display(PTR first) {
   if (!first) {
      printf("SLL is Empty\n");
      return;
   }

   printf("USN\tName\tProgram\tSem\tPhone\n");
   PTR cur = first;     while
(cur) {
      printf("%s\t%s\t%s\t%d\t%s\n", cur->usn, cur->name, cur->prog, cur->sem, cur->ph);        cur =
cur->link;
   }
}
```

Program 8

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

typedef struct node {
   int usn, sal;
   char name[30], dept[4], desig[30], ph[11];     struct
node *plink, *nlink;
} *NODE;

NODE createNode() {
   NODE temp = (NODE)malloc(sizeof(struct node)); // Fix size allocation
   if (!temp) {
      printf("\nMemory overflow\n");
      exit(0);
   }
   printf("\nEnter SSN, Name, Dept, Designation, Salary, Phone: ");
   scanf("%d %s %s %s %d %s", &temp->usn, temp->name, temp->dept, temp->desig, &temp->sal, temp-
>ph);     temp->plink = temp->nlink =
NULL;     return temp;
}

NODE insertRear(NODE first) {
   NODE temp = createNode();     NODE
cur = first;
   if (!first) {
```

```
        return temp;
    }
    while (cur->nlink) {
cur = cur->nlink;
    }
    cur->nlink = temp;     temp>plink
= cur;
    return first;
}

NODE deleteFront(NODE first) {
    if (!first) {
        printf("\nDLL is empty\n");
return NULL;
    }
    NODE temp = first;     first
= first->nlink;
    if (first) {        first-
>plink = NULL;
    }
    free(temp); // Free the memory of the deleted node
    return first;
}

NODE insertFront(NODE first) {     NODE
temp = createNode();
    temp->nlink = first;
```

```
   if (first) {          first>plink =
temp;    }     return temp;
}

NODE deleteRear(NODE first) {
   if (!first) {
      printf("\nDLL is empty\n");
return NULL;
   }
   NODE cur = first;     while
(cur->nlink) {        cur =
cur->nlink;
   }
   if (cur->plink) {        cur->plink-
>nlink = NULL;
   } else {
      first = NULL; // Handle case when only one node is present
   }
   printf("\nDeleted %s\n", cur->name);     free(cur);
// Free the memory of the deleted node
   return first;
}

void display(NODE first) {
   if (!first) {
      printf("\nDLL is empty\n");
      return;
   }
```

```
   printf("\nSSN\tName\tDept\tDesig\tSalary\tPhone\n");     NODE cur = first;     while (cur) {
printf("%d\t%s\t%s\t%s\t%d\t%s\n", cur->usn, cur->name, cur->dept, cur->desig, cur->sal, cur->ph);
cur = cur->nlink;
   }
}

int main() {
   NODE first = NULL;     int choice, n;
printf("Enter number of employees: ");
scanf("%d", &n);
   while (n--) {        first = insertRear(first); // Insert nodes
into the list
   }

   while (1) {
      printf("\n1.Insert Rear\n2.Insert Front\n3.Delete Front\n4.Delete Rear\n5.Display\n6.Exit\n");
scanf("%d", &choice);

      if (choice == 6) {
break;
      }
      switch (choice) {            case 1:
first = insertRear(first);
// Insert at rear
            break;
case 2:
            first = insertFront(first); // Insert at front
            break;
```

```
case 3:
            first = deleteFront(first); // Delete from front
            break;
case 4:
            first = deleteRear(first); // Delete from rear
            break;          case 5:
display(first); // Display the list              break;
default:
            printf("\nInvalid choice\n");
      }
   }
   return 0;
}
```

Program 9

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

struct polyt  {
int cf,px,
py,pz;
struct polyt*
next;
};

typedef struct polyt* PTR;

PTR insert(PTR poly, int cf, int px, int py, int pz) {
             PTR cur;
   PTR nn = (PTR)malloc(sizeof(struct polyt));
```

```
   nn->cf = cf;     nn->px
= px;     nn->py = py;
nn->pz = pz;      nn-
>next = NULL;

   cur = poly;

   while(cur->next != poly)
   {
            cur = cur->next;
   }

          cur->next = nn;
          nn->next = poly;
          return poly;
}
void disp(PTR poly)
{
   if (poly->next == poly)
   {
      printf("Polynomial is empty.\n");
      return;
   }
   PTR cur = poly->next;
   do
   {
      printf("%dx^%dy^%dz^%d ", cur->cf, cur->px, cur->py, cur->pz);        cur = cur->next;
      if (cur != poly)
      {
         printf("+ ");
      }
   } while (cur != poly);     printf("\n");
}
```

```
int evaluate(PTR poly, int x, int y, int z)
{
   int result = 0;
   if (poly->next == poly)
   {
      return result;
   }
   PTR cur = poly->next;     do
   {
      int termValue = cur->cf;        termValue *=
pow(x, cur->px);        termValue *= pow(y, cur->py);
      termValue *= pow(z, cur->pz);

      result += termValue;

      cur = cur->next;     } while
(cur != poly);

   return result;
}

bool fmatch(PTR p1, PTR p2)
{
 bool  match  =  true;   if(p1->px  !=  p2-
>px)   match = false;
           if(p1->py != p2->py)
match = false;
              if(p1->pz != p2->pz)
                     match = false;
             return match;
}
```

```
PTR add(PTR poly1, PTR poly2, PTR polySum)
{
   PTR cur1 = poly1->next;
   PTR cur2 = poly2->next;
   do
   {
    polySum = insert(polySum, cur1->cf, cur1->px, cur1->py, cur1->pz);      cur1 = cur1->next;
   } while(cur1 != poly1);
   do
   {
      cur1 = polySum->next;
      bool matchfound = false;
      do
      {
         if(fmatch(cur1, cur2))
         {
            cur1->cf += cur2->cf;               matchfound =
true;
            break;
         }
         cur1 = cur1->next;        }
while(cur1 != polySum);
if(!matchfound)
      {
         polySum = insert(polySum, cur2->cf, cur2->px, cur2->py, cur2->pz);
      }
      cur2 = cur2->next;     }
while(cur2 != poly2);
   return polySum;
```

```
}

int main()
{
   PTR poly1 = (PTR)malloc(sizeof(struct polyt));     poly1->next =
poly1;
   PTR poly2 = (PTR)malloc(sizeof(struct polyt));     poly2->next =
poly2;
   PTR polySum = (PTR)malloc(sizeof(struct polyt));     polySum->next
= polySum;

   poly1 = insert(poly1, 6, 2, 2, 1);     poly1 =
insert(poly1, 4, 0, 1, 5);     poly1 = insert(poly1, 3, 3,
1, 1);     poly1 = insert(poly1, 2, 1, 5, 1);     poly1 =
insert(poly1, 2, 1, 1, 3);

   // Display the polynomial P(x, y, z)     printf("POLY1(x, y, z) = ");
   disp(poly1);

   // Read and evaluate the second polynomial POLY2(x, y, z)     // Represent
the polynomial P(x, y, z) = xyz + 4x^3yz     poly2 = insert(poly2, 1, 1, 1, 1);  //
Example term     poly2 = insert(poly2, 4, 3, 1, 1);

   //    Display    the    second    polynomial    POLY2(x,    y,    z)
printf("POLY2(x, y, z) = ");    disp(poly2);

   // Add POLY1(x, y, z) and POLY2(x, y, z) and store the result in POLYSUM(x, y, z)     polySum =
add(poly1, poly2, polySum);

   //    Display    the    sum    POLYSUM(x,    y,    z)
printf("\nPOLYSUM(x, y, z) = ");     disp(polySum);

   // Evaluate POLYSUM(x, y, z) for specific values     int x = 1, y = 2, z
= 3;
   int res = evaluate(polySum, x, y, z);
   printf("\nResult of POLYSUM(%d, %d, %d): %d\n", x, y, z, res);
```

```
    return 0;
}
```