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

struct NodoLista{
    char dato;
    struct NodoLista *ptrSiguiente;
};

void insertar(struct NodoLista **ptrS); //**ptrS es el apuntador que representa *ptrSiguiente para poder usarse en el programa//
char eliminar (struct NodoLista **ptrS);
void imprimeLista(struct NodoLista *ptrActual);

int main(){
struct NodoLista *ptrInicial;
ptrInicial = NULL;
char letra;
int opcion=1;
while (opcion != 0){
    printf("Ingresa la opcion deseada\n");
    printf("1.- Ingresar\n2.- Sacar\n3.- imprimir\n0.- Salir\n");
    scanf("%d", &opcion);
    fflush(stdin);
    switch (opcion) {
        case 1:
            insertar(&ptrInicial); //ESTOY ASIGNADO A L^ FUNCIÓN UNA DIRECCIÓN DE MEMORIA(&) POR LO QUE LA FUNCION DEBE SER C                    SE USA EL DOBLE APUNTADOR EN EL PROTOTIPO DE LA FUNCIÓN
            break;

        case 2:
            letra = eliminar(&ptrInicial);
            printf("El valor eliminado es %c\n", letra);
            break;

        case 3:
            imprimeLista(ptrInicial);
            break;

        default:
            break;
    }
}
return 0;
}
//=====================================


void insertar(struct NodoLista **ptrS){
    struct NodoLista *ptrActual;
    struct NodoLista *ptrAnterior;
    struct NodoLista *ptrNuevo;
    char letra;
    printf("Ingresa la letra deseada:\n");
    scanf("%c", &letra);
    fflush(stdin);
    ptrNuevo = (struct NodoLista *)malloc(sizeof(struct NodoLista));
    if (ptrNuevo != NULL){
        ptrNuevo->dato = letra;
        printf("\n58ptrNuevo->dato: %c\n",ptrNuevo->dato);
        ptrAnterior = NULL;
        //printf("ptrActual %p\n",ptrActual);
        ptrActual = *ptrS;
        printf("62*ptrS:%p\n", *ptrS);
        printf("63ptrActual: %p *ptrS: %p\n",ptrActual, *ptrS);
```

Considero que [PrintF] es una llamada al sistema porque nos permite la impresión de un dato deseado en el monitor actual. Printf contiene autorización del S.O para tomar el mensaje y poder imprimirlo.

Considero que [Scanf] es una llamada al sistema ya que se carga un determinado tipo de dato en una dirección de memoria que puede recibir ese tipo de dato.

[fflush (stdin)] es una función que realiza la limpieza del buffer de entrada al hacer esto libera memoria es por esto que la considero una llamada al sistema.

[Malloc] le pide memoria al sistema operativo para usarla dentro de mi proceso. Es por eso que considero es una llamada al sistema de tipo control de procesos.

```
64
65          while (ptrActual != NULL && letra > ptrActual->dato){
66              printf("66%p\n",ptrAnterior);
67              ptrAnterior = ptrActual;
68              printf("68 ptrActual: %p  ptrActual->dato: %c\n",ptrActual,ptrActual->dato);
69              ptrActual=ptrActual->ptrSiguiente;
70              printf("70ptrActual: %p\n",ptrActual);
71          }
72          if(ptrAnterior == NULL){
73              ptrNuevo->ptrSiguiente = *ptrS;
74              printf("74*ptrS: %p\n",*ptrS);
75              *ptrS=ptrNuevo;
76              printf("76ptrNuevo: %p\n",ptrNuevo);
77
78          }else{
79              ptrAnterior->ptrSiguiente = ptrNuevo;
80              printf("80ptrNuevo: %p\n",ptrNuevo);
81              ptrNuevo->ptrSiguiente = ptrActual;
82              printf("82ptrActual: %p\n",ptrActual);
83          }
84
85      }
86      else
87      {
88          printf("No hay espacio en memoria\n");
89      }
90  }
91
92
93  char eliminar (struct NodoLista **ptrS){
94      char letra;
95      struct NodoLista *ptrActual;
96      struct NodoLista *ptrAnterior;
97      struct NodoLista *ptrTemp;
98
99      printf("Que letra deseas eliminar?\n");
100     scanf(" %c", &letra);
101     //eliminar el primer dato
102     if (letra == (*ptrS)->dato) {
103         ptrTemp = *ptrS;
104         printf("104*ptrS:%p",*ptrS);
105         *ptrS = (*ptrS)->ptrSiguiente;
106         printf("106(*ptrS)->ptrSiguiente: %p\n",(*ptrS)->ptrSiguiente);
107         free(ptrTemp);
108         return letra;
109     }
110     ptrAnterior = *ptrS;
111     printf("111*ptrS:%p\n",*ptrS);
112     ptrActual = (*ptrS)->ptrSiguiente;
113     printf("113 (*ptrS)->ptrSiguiente: %p\n",(*ptrS)->ptrSiguiente);
114
115     while (ptrActual != NULL && ptrActual -> dato != letra) {
116         ptrAnterior = ptrActual;
117         printf("117ptrActual: %p\n",ptrActual);
118         ptrActual = ptrActual->ptrSiguiente;
119         printf("117ptrActual->ptrSiguiente: %p\n",ptrActual->ptrSiguiente);
120     }
121
122     if (ptrActual != NULL){
123         ptrTemp = ptrActual;
124         printf("ptrActual: %p\n",ptrActual);
125         ptrAnterior->ptrSiguiente = ptrActual->ptrSiguiente;
126         printf("ptrActual->ptrSiguiente: %p",ptrActual->ptrSiguiente);
127         free(ptrTemp);
128         return letra;
129     }
```

free( ) me permite liberar la memoria asignada dinamicamente, es decir en este caso usando Malloc. Al tener esta función sabemos que es una llamada al sistema de control de procesos porque se libera memoria.

```
130     return '\0';
131 }
132
133 void imprimeLista(struct NodoLista *ptrActual){
134     if (ptrActual==NULL){
135         printf("La lista esta vacia\n");
136         return;
137     }
138     printf("La lista es: \n");
139     while (ptrActual != NULL) {
140         printf("%c ===>", ptrActual->dato);
141         ptrActual=ptrActual->ptrSiguiente;
142     }
143     printf("NULL\n");
144 }
145
146 //void eliminar (struct NodoLista **ptrS){
147 //
148 //}
```