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

#define tamLista 5
#define false 0
#define true 1
//#include <stdbool.h>

typedef int TipoChave;

typedef struct{ TipoChave chave;
} Registro;

typedef struct { Registro A[tamLista];
                 int nroElemento;
} Lista;

void menu();
void inicializaLista(Lista* l);
void tamanhoLista(Lista* l);
void exibeLista(Lista* l);
bool inserirLista(Lista* l, Registro reg, int pos);
bool inserirListaCompleta(Lista* l);
bool alterarLista(Lista* l, TipoChave chave, int pos);
bool excluirLista(Lista* l, TipoChave chave);
int buscarLista(Lista* l, TipoChave chave);

int main(){
    Registro reg;
    Lista l;
    inicializaLista(&l);
    int opc;
    int pos;
    int chave;

    do {
        menu();
        printf("Selecione Opcao? "); scanf("%d", &opc);
        switch (opc) {
            case 0: printf("[Lista Reinicializada]\n"); inicializaLista(&l); break;
            case 1: printf("[Tamanho da Lista]\n"); tamanhoLista(&l); break;
            case 2: printf("[Exibindo Lista] \n"); exibeLista(&l); break;

            case 3: printf("[Modulo Inclusao] Digite Elemento?");scanf("%d", &reg);
                    printf("Digite Posicao?");scanf("%d", &pos);
                    if ( inserirLista(&l, reg, pos) == true ) printf("Elemento INSERIDO...

            case 4: inserirListaCompleta(&l); break;

            case 5: printf("[Modulo Alteracao] Digite Elemento para alteracao?");scanf("%d
                    printf("Digite Posicao?");scanf("%d", &pos);
                    if ( alterarLista(&l, chave, pos) == true ) printf("Elemento ALTERADO.

            case 6: printf("[Modulo Exclusao] Digite Elemento a ser Excluido?");scanf("%d"
                    if ( excluirLista(&l, chave) == true ) printf("Elemento EXCLUIDO...");

            case 7: printf("[Modulo Consulta] Digite Elemento a ser Consultado?");scanf("%
                    if ( buscarLista(&l, chave) >= 0 ) printf("\n Registro Encontrado"); e

            case 8: exit(1); break;
```

```
61:             default:
62:             printf("Opcao Invalida\n");
63:             }
64:             printf("\n\n");
65:             system("PAUSE");
66:        } while ( opc != 8 );
67:     return 0;
68: }
69:
70: void menu() {
71:         system("CLS");
72:         printf("0 - Inicializar Lista \n");
73:         printf("1 - Exibir Tamanho da Lista \n");
74:         printf("2 - Exibir Lista \n");
75:         printf("3 - Inserir na Lista \n");
76:         printf("4 - Inserir na Lista Completa \n");
77:         printf("5 - Alterar na Lista \n");
78:         printf("6 - Excluir na Lista \n");
79:         printf("7 - Buscar na Lista \n");
80:         printf("8 - Sair da Rotina \n");
81: };
82:
83: void inicializaLista(Lista* l) {
84:     l->nroElemento = 0;
85: }
86:
87: void tamanhoLista(Lista* l){
88:     printf("\nQuantidade de Elementos na Lista = %d \n", l->nroElemento);
89:     printf("\nTotal em Bytes %d \n", sizeof(l));
90: }
91:
92: void exibeLista(Lista* l){
93:     printf("\n\n");
94:     if ( l->nroElemento == 0 ) printf("Lista Vazia\n");
95:
96:     for (int i = 0; i<l->nroElemento; i++)
97:       printf("Elemento [%d]=%d\n", i, l->A[i].chave);
98:
99:     if ( l->nroElemento < tamLista)
100:      printf("\n Proxima Posicao LIVRE da Lista = %d", l->nroElemento);
101:    else
102:      printf("\n Lista CHEIA");
103: };
104:
105: bool inserirLista(Lista* l, Registro reg, int pos){
106:     if ( l->nroElemento == tamLista || pos > l->nroElemento || pos < 0 || pos > tamLista )
107:
108:     for (int i = l->nroElemento; i > pos; i--)
109:       l->A[i] = l->A[i-1];
110:
111:     l->A[pos] = reg;
112:     l->nroElemento++;
113:     return true;
114:  }
115:
116: bool inserirListaCompleta(Lista* l) {
117:     Registro reg;
118:
119:     if ( l->nroElemento > 0 ) { printf("\nLista ja INSERIDA"); return false; }
120:
```

```
121:     for (int i = 0; i < tamLista; i++){
122:         printf("\nDigite Elemento [%d]=? ", i); scanf("%d", &reg);
123:         l->A[i] = reg;
124:         l->nroElemento++;
125:      }
126:
127:      printf("\nLista INSERIDA");
128:      return true;
129: }
130:
131: bool alterarLista(Lista* l, TipoChave chave, int pos){
132:     if ( pos < 0 || pos > tamLista || pos > l->nroElemento-1 || l->nroElemento == 0 ) retu
133:
134:     l->A[pos].chave = chave;
135:     return true;
136:  }
137:
138: bool excluirLista(Lista* l, TipoChave chave){
139:     int pos = buscarLista(l, chave);
140:     if ( pos == -1 ) return false;
141:
142:     for ( int i = pos; i < l->nroElemento-1; i++)
143:       l->A[i] = l->A[i+1];
144:
145:     l->nroElemento--;
146:     return true;
147:  }
148:
149:  int buscarLista(Lista* l, TipoChave chave){
150:     for (int pos=0; pos < l->nroElemento; pos++)
151:       if ( chave == l->A[pos].chave )
152:         return pos;
153:
154:     // Se não encontrou
155:     return -1;
156:  }
157:
158:  // Outras opções de funcionalidades que podem ser implementadas
159:
160:  // Primeiro ELemento da Lista
161:  TipoChave primeiroElementoLista(Lista* l){
162:     if ( l->nroElemento > 0 ) return l->A[0].chave;
163:     else return -1;
164:  }
165:
166: // Último ELemento da Lista
167:  TipoChave ultimoElementoLista(Lista* l){
168:     if ( l->nroElemento > 0 ) return l->A[l->nroElemento-1].chave;
169:     else return -1;
170:  }
171:
172: // Retorna a chave da posição da LISTA
173: TipoChave chaveElementoLista(Lista* l, int pos){
174:     if ( ( l->nroElemento > 0 ) && ( pos >= 0 && pos < l->nroElemento ) ) return l->A[pos]
175:     else return -1;
176:  }
177:
```