Códigos Para Examen 2 Parcial Programación.

Listas Dinámicas.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct Def\_Cola\_Impresion

{

char NombreDoc[200],Estado[200];

int ID,NumHojas,HojasImpresas;

struct Def\_Cola\_Impresion \*sig;

}TipoDocumento;

void IntroducirDoc(TipoDocumento \*\*ColaImpresion, TipoDocumento NuevoDocumento, int Numeracion);

void ListarTrabajos(TipoDocumento \*ColaImpresion);

void EliminarTrabajos(TipoDocumento \*\*ColaImpresion,int ID);

void ImprimirDoc(TipoDocumento \*\*ColaImpresion);

void GuardarColaArch(TipoDocumento \*ColaImpresion);

void LeerColaArch(TipoDocumento \*\*ColaImpresion,int \*Numeracion);

void Borrar\_Lista(TipoDocumento \*ColaImpresion);

int main (void)

{

TipoDocumento \*ColaImpresion=NULL, NuevoDocumento;

int Opcion,Salir=0,Numeracion=1,ErrorLectura=0,ID;

do

{

system("clear");

printf("Menú Cola Impresión:\n");

printf("1.- Introducir documento a la cola de impresión.\n");

printf("2.- Listar trabajos en la cola de impresión.\n");

printf("3.- Eliminar trabajo de la cola de impresión.\n");

printf("4.- Imprimir documento.\n");

printf("5.- Guardar cola en archivo.\n");

printf("6.- Leer cola en archivo.\n");

printf("7.- Salir.\n\n");

printf("Escoge una opción: \n");

scanf(" %d",&Opcion);

switch(Opcion)

  {

  case 1:

   system("clear");

   printf("Ingresar a la cola documento:\n");

   printf("Introduzca el nombre del documento\n");

   \_\_fpurge(stdin);

   gets(NuevoDocumento.NombreDoc);

   printf("Introduzca el número de hojas que contiene el documento\n");

   \_\_fpurge(stdin);

   scanf("%d",&NuevoDocumento.NumHojas);

   if(NuevoDocumento.NumHojas<=0)

   ErrorLectura=1;

   if(ErrorLectura==0)

   {

   IntroducirDoc(&ColaImpresion,NuevoDocumento,Numeracion);

   Numeracion++;

   }

   else

   printf("El numero hojas debe ser de al menos 1 hoja, intente de nuevo\n");

   printf("Presione Enter para continuar...\n");

   \_\_fpurge(stdin);

   getchar();

   break;

  case 2:

   system("clear");

   ListarTrabajos(ColaImpresion);

   break;

  case 3:

   system("clear");

   printf("Introduzca el ID del documento a eliminar de la cola\n");

   scanf(" %d",&ID);

   EliminarTrabajos(&ColaImpresion,ID);

   printf("Presione Enter para continuar...\n");

   \_\_fpurge(stdin);

   getchar();

   break;

  case 4:

   system("clear");

   ImprimirDoc(&ColaImpresion);

   break;

  case 5:

   system("clear");

   GuardarColaArch(ColaImpresion);

   break;

  case 6:

   system("clear");

   LeerColaArch(&ColaImpresion,&Numeracion);

   break;

  case 7:

   Salir=1;

   break;

  default:

   system("clear");

   printf("La opción a seleccionar ha sido incorrecta, intentelo de nuevo :(\n");

   break;

  }

} while(Salir==0);

Borrar\_Lista(ColaImpresion);

printf("Gracias por utilizar el programa, hasta la próxima\n");

}

void ListarTrabajos(TipoDocumento \*ColaImpresion)

{

system("clear");

TipoDocumento \*temp,\*temp2;

temp=ColaImpresion;

temp2=ColaImpresion;

printf("Cola de Impresion:\n");

while(temp!=NULL)

{

if(strcmp("Espera",temp->Estado)==0)

  printf("%d: %s, %d hojas, %s\n",temp->ID, temp->NombreDoc, temp->NumHojas, temp->Estado);

temp=temp->sig;

}

while(temp2!=NULL)

{

if(strcmp("Imprimiendo",temp2->Estado)==0)

  printf("%d: %s, %d hojas, %s, %d hoja(s) impresa(s)\n",temp2->ID, temp2->NombreDoc, temp2->NumHojas, temp2->Estado, temp2->HojasImpresas);

temp2=temp2->sig;

}

printf("Presione Enter para continuar...\n");

\_\_fpurge(stdin);

getchar();

}

void EliminarTrabajos(TipoDocumento \*\*ColaImpresion,int ID)

{

TipoDocumento \*temp,\*temp2;

temp=\*ColaImpresion;

temp2=\*ColaImpresion;

if(\*ColaImpresion==NULL) // Si la lista está vacía

{

printf("La lista está vacía, no puede eliminar ningún documento\n");

}

else

{

while(temp!=NULL)

  {

   if(temp->ID == ID)

   {

   if(temp == \*ColaImpresion)

    {

     \*ColaImpresion = temp->sig;

     free(temp);

    }

   else

    {

     while(temp2->sig!=temp)

     temp2=temp2->sig;

     temp2->sig=temp->sig;

     free(temp);

    }

   }

   temp=temp->sig;

  }

}

}

void IntroducirDoc(TipoDocumento \*\*ColaImpresion, TipoDocumento NuevoDocumento, int Numeracion)

{

TipoDocumento \*nuevo;

nuevo=(TipoDocumento \*)malloc(sizeof(TipoDocumento));

strcpy(nuevo->Estado,"Espera");

strcpy(nuevo->NombreDoc,NuevoDocumento.NombreDoc);

nuevo->NumHojas=NuevoDocumento.NumHojas;

nuevo->HojasImpresas=0;

nuevo->ID=Numeracion;

nuevo->sig=\*ColaImpresion;

\*ColaImpresion=nuevo;

}

void ImprimirDoc(TipoDocumento \*\*ColaImpresion)

{

TipoDocumento \*temp,\*temp2;

int i=1,Verificado=0;

temp=\*ColaImpresion;

temp2=\*ColaImpresion;

while(temp!=NULL)

{

if(strcmp(temp->Estado,"Imprimiendo")==0)

  {

if(temp->HojasImpresas==temp->NumHojas)

   EliminarTrabajos(ColaImpresion,temp->ID);

   else

   {

temp->HojasImpresas+=1;

   printf("Se imprimió con éxito\n");

   Verificado=1;

   }

  }

temp=temp->sig;

}

if(Verificado==0)

{

while(temp2!=NULL && Verificado==0)

  {

   if(temp2->sig==NULL && strcmp(temp2->Estado,"Espera")==0)

   {

   strcpy(temp2->Estado,"Imprimiendo");

   temp2->HojasImpresas=1;

   Verificado=1;

   printf("Se imprimió con éxito\n");

   if(temp2->HojasImpresas==temp2->NumHojas)

    EliminarTrabajos(ColaImpresion,temp2->ID);

   }

   temp2=temp2->sig;

  }

}

printf("Presione Enter para continuar...\n");

\_\_fpurge(stdin);

getchar();

}

void GuardarColaArch(TipoDocumento \*ColaImpresion)

{

TipoDocumento \*temp;

char NombreArchivo[200];

FILE \*Archivo;

temp=ColaImpresion; printf("Ingresa el nombre del archivo a guardar\n");

\_\_fpurge(stdin);

gets(NombreArchivo);

Archivo = fopen(NombreArchivo,"wt");

while(temp!=NULL)

{

fprintf(Archivo,"%d\n%s\n%d\n%d\n%s\n",temp->ID, temp->NombreDoc, temp->NumHojas,temp->HojasImpresas,temp->Estado);

temp=temp->sig;

}

fclose(Archivo);

}

void LeerColaArch(TipoDocumento \*\*ColaImpresion,int \*Numeracion)

{

TipoDocumento \*temp, \*nuevo;

int NuevoID\_Documento;

char NombreArchivo[200];

FILE \*Archivo;

printf("Ingresa el nombre del archivo a leer\n");

\_\_fpurge(stdin);

gets(NombreArchivo);

Archivo = fopen(NombreArchivo,"rt");

if(Archivo==NULL)

printf("No existe el archivo,intentelo de nuevo\n");

else

{

while ((fscanf (Archivo, " %d", &NuevoID\_Documento)) == 1)

  {

   nuevo = (TipoDocumento \*)malloc(sizeof(TipoDocumento));

   nuevo -> ID = NuevoID\_Documento;

   \*Numeracion = NuevoID\_Documento;

   fscanf (Archivo, " %[^\n]", nuevo -> NombreDoc);

   fscanf (Archivo, " %d", &nuevo -> NumHojas);

   fscanf (Archivo, " %d", &nuevo -> HojasImpresas);

   fscanf (Archivo, " %[^\n]", nuevo -> Estado);

   nuevo -> sig = NULL;

   if (\*ColaImpresion != NULL)

   {

   temp = \*ColaImpresion;

   while (temp -> sig != NULL)

    temp = temp -> sig;

   temp -> sig = nuevo;

   }

   else

   \*ColaImpresion = nuevo;

  }

if(\*ColaImpresion==NULL)

\*Numeracion=1;

else

\*Numeracion+=1;

fclose(Archivo);

}

printf("Presione Enter para continuar...\n");

\_\_fpurge(stdin);

getchar();

}

void Borrar\_Lista(TipoDocumento \*ColaImpresion)

{

TipoDocumento \*temp;

temp=ColaImpresion;

while(temp!=NULL)

{

ColaImpresion=ColaImpresion->sig;

free(temp);

temp=ColaImpresion;

}

}

Listas Dinámicas.

#include <stdio.h>

#include <stdlib.h>

typedef struct def\_elemento

{

int dato;

struct def\_elemento \*ant;

struct def\_elemento \*sig;

} TipoElemento;

void Inserta\_Fin(TipoElemento \*\*Inicio, TipoElemento \*nuevo);

void Inserta\_Inicio(TipoElemento \*\*Inicio, TipoElemento \*nuevo);

void Ordenar(TipoElemento \*\*Inicio, int numero);

void ImprimirDerecha(TipoElemento \*Inicio);

void ImprimirIzquierda(TipoElemento \*Inicio);

void Imprimir(TipoElemento \*Inicio);

void Borrar\_Lista(TipoElemento \*Inicio);

int main (void)

{

TipoElemento \*Primero=NULL;

int valor,Opcion,Verificado=0;

do {

system("clear");

printf("Menú\n\n");

printf("1.- Insertar Número.\n");

printf("2.- Imprimir Derecha.\n");

printf("3.- Imprimir Izquierda.\n");

printf("4.- Salir.\n\n");

printf("Ingresa la opción a usar\n");

scanf(" %d", &Opcion);

switch(Opcion)

{

case 1:

system("clear");

printf("¿Qué número desea insertar?\n");

if(scanf(" %d", &valor)==1);

Ordenar(&Primero,valor);

printf("\nPresione enter para continuar\n");

\_\_fpurge(stdin);

getchar();

break;

case 2:

system("clear");

printf("Número en lista: \n");

ImprimirDerecha(Primero);

printf("Presione enter para continuar\n");

\_\_fpurge(stdin);

getchar();

break;

case 3:

system("clear");

printf("Número en lista: \n");

ImprimirIzquierda(Primero);

printf("Presione enter para continuar\n");

\_\_fpurge(stdin);

getchar();

break;

case 4:

Borrar\_Lista(Primero);

printf("Saliendo del programa...\n");

Verificado=1;

break;

default:

printf("Opción errónea, inténtelo de nuevo\n");

printf("Presione enter para continuar\n");

\_\_fpurge(stdin);

getchar();

break;

}

} while(Verificado==0);

}

void Ordenar(TipoElemento \*\*Inicio, int numero)

{

TipoElemento \*pos, \*temp2, \*nuevo;

nuevo=(TipoElemento \*)malloc(sizeof(TipoElemento));

nuevo->dato=numero;

pos=\*Inicio;

while((pos!=NULL)&&(numero>pos->dato))

pos=pos->sig;

if(pos==NULL)

Inserta\_Fin(Inicio,nuevo);

else

{

if(pos==\*Inicio)

Inserta\_Inicio(Inicio,nuevo);

else

{

temp2=\*Inicio;

   while(temp2->sig!=pos)

   temp2=temp2->sig;

   nuevo->sig=pos;

nuevo->ant=temp2;

pos->ant=nuevo;

   temp2->sig=nuevo;

}

}

}

void Inserta\_Inicio(TipoElemento \*\*Inicio, TipoElemento \*nuevo)

{

nuevo->ant=NULL;

nuevo->sig=\*Inicio;

\*Inicio=nuevo;

}

void Inserta\_Fin(TipoElemento \*\*Inicio, TipoElemento \*nuevo)

{

TipoElemento \*temp2;

nuevo->sig=NULL;

if(\*Inicio!=NULL)

{

temp2=\*Inicio;

while(temp2->sig!=NULL)

temp2=temp2->sig;

nuevo->ant=temp2;

temp2->sig=nuevo;

}

else

{

nuevo->ant=NULL;

\*Inicio=nuevo;

}

}

void ImprimirDerecha(TipoElemento \*Inicio)

{

TipoElemento \*temp;

temp=Inicio;

while(temp!=NULL)

{

printf("%d\n",temp->dato);

temp=temp->sig;

}

}

void ImprimirIzquierda(TipoElemento \*Inicio)

{

TipoElemento \*temp;

temp=Inicio;

if(Inicio==NULL)

{

printf("No hay elementos en la lista\n");

}

else

{

while(temp->sig!=NULL)

{

temp=temp->sig;

}

while(temp!=NULL)

{

printf("%d\n",temp->dato);

temp=temp->ant;

}

}

}

void Borrar\_Lista(TipoElemento \*Inicio)

{

TipoElemento \*temp;

temp=Inicio;

while(temp!=NULL)

{

Inicio=Inicio->sig;

free(temp);

temp=Inicio;

}

}

Árboles Binarios.

#include <stdio.h>

#include <stdlib.h>

typedef struct nodo

{

int num;

struct nodo \*izq, \*der;

} TipoNodo;

void Inserta(TipoNodo \*\*Raiz, int numero);

void Imprime(TipoNodo \*Raiz);

void Cambiar(TipoNodo \*\*Raiz, TipoNodo \*\*temp);

void Eliminar(TipoNodo \*\*Raiz, int numero);

void Borrar(TipoNodo \*Raiz);

int main (void)

{

int numero,Opcion,Salir=0;

TipoNodo \*Raiz=NULL;

do

{

  system("clear");

printf("Menú Árbol Binario:\n");

printf("1.- Insertar número\n");

printf("2.- Borrar número\n");

printf("3.- Imprimir Árbol\n");

printf("4.- Salir\n");

printf("Escoga una opción\n");

scanf(" %d", &Opcion);

switch(Opcion)

{

case 1:

printf("Teclea enteros, seguido por un caracter no numérico\n");

scanf(" %d",&numero);

Inserta(&Raiz,numero);

break;

case 2:

   printf("Ingrese el número a eliminar\n");

   scanf(" %d", &numero);

Eliminar(&Raiz,numero);

break;

case 3:

printf("Números, impresion en orden:\n");

Imprime(Raiz);

getchar();

  getchar();

break;

case 4:

Salir=1;

break;

}

} while(Salir==0);

Borrar(Raiz);

printf("Gracias por utilizar el programa\n");

}

void Inserta(TipoNodo \*\*Raiz, int numero)

{

TipoNodo \*Avanza, \*Nuevo;

Avanza=\*Raiz;

if((Nuevo=(TipoNodo \*) malloc(sizeof(TipoNodo)))==NULL)

{

printf("No hay memoria\n");

exit(1);

}

Nuevo->num=numero;

Nuevo->izq=NULL;

Nuevo->der=NULL;

while (Avanza!=NULL)

{

if(numero > Avanza->num)

{

if(Avanza->der !=NULL)

Avanza= Avanza->der;

else

{

Avanza->der= Nuevo;

return;

}

}

if (numero <= Avanza->num)

{

if(Avanza->izq !=NULL)

Avanza= Avanza->izq;

else

{

Avanza->izq= Nuevo;

return;

}

}

}

Avanza=Nuevo;

\*Raiz=Avanza;

}

void Eliminar(TipoNodo \*\*Raiz, int numero)

{

TipoNodo \*temp;

if (\*Raiz == NULL)

return;

if ((\*Raiz)->num < numero)

Eliminar(&(\*Raiz)->der, numero);

else if ((\*Raiz)->num > numero)

Eliminar(&(\*Raiz)->izq, numero);

else if ((\*Raiz)->num == numero)

{

temp = \*Raiz;

if ((\*Raiz)->izq == NULL)

\*Raiz = (\*Raiz)->der;

else if ((\*Raiz)->der == NULL)

\*Raiz = (\*Raiz)->izq;

else

Cambiar(&(\*Raiz)->izq, &temp);

free(temp);

}

}

void Cambiar(TipoNodo \*\*Raiz, TipoNodo \*\*temp)

{

if ((\*Raiz)->der == NULL)

{

(\*temp)->num = (\*Raiz)->num;

\*temp = \*Raiz;

\*Raiz = (\*Raiz)->izq;

}

else

Cambiar(&(\*Raiz)->der, temp);

}

void Imprime(TipoNodo \*Raiz)

{

if(Raiz!=NULL)

{

Imprime(Raiz->izq);

printf("%d\n",Raiz->num);

Imprime(Raiz->der);

}

}

void Borrar(TipoNodo \*Raiz)

{

if(Raiz!=NULL)

{

Borrar(Raiz->izq);

Borrar(Raiz->der);

free(Raiz);

}

}

Métodos de Ordenamiento

Burbuja

#include <stdio.h>

void Burbuja(int m[], int elementos);

int main(void)

{

int m[]={25,2,7,3,67,43,1,15,9,32};

int i;

printf("Arreglo original\n");

for(i=0; i<10; i++)

printf("%d ", m[i]);

printf("\n");

printf("Arreglo ordenado\n");

Burbuja(m,10);

for(i=0; i<10; i++)

printf("%d ", m[i]);

printf("\n");

}

void Burbuja(int m[], int elementos)

{

int aux, i, s=1;

while(s && (--elementos>0))

{

s=0;

for(i=1; i<=elementos; i++)

if(m[i-1]>m[i])

{

aux=m[i-1];

m[i-1]=m[i];

m[i]=aux;

s=1;

}

}

}

Inserción

#include <stdio.h>

void Insercion(int m[], int elementos);

int main(void)

{

int m[]={25,2,7,3,67,43,1,15,9,32};

int i;

printf("Arreglo original\n");

for(i=0; i<10; i++)

printf("%d ", m[i]);

printf("\n");

printf("Arreglo ordenado\n");

Insercion(m,10);

for(i=0; i<10; i++)

printf("%d ", m[i]);

printf("\n");

}

void Insercion(int m[], int elementos)

{

int i, k, x;

//desde el segundo elemento.

for(i=1; i<elementos; i++)

{

x=m[i];

k=i-1;

while (k>=0 && x<m[k])

{

m[k+1]=m[k]; //hacer espacio para insertar

k--;

}

m[k+1]=x;

}

}

Selección

#include <stdio.h>

#define SIZE 10

void Seleccion(int Arreglo[]);

int main(void)

{

int Arreglo[SIZE], i;

for(i=0; i<SIZE;i++)

{

printf("Ingrese el elemento %d:\n",i+1);

scanf(" %d", &Arreglo[i]);

}

Seleccion(Arreglo);

printf("Despues de la seleccion de elementos\n");

for(i=0; i<SIZE;i++)

printf("%d\t",Arreglo[i]);

printf("\n");

getchar();

}

void Seleccion(int Arreglo[])

{

int i,j,min,temp;

for(i=0; i<SIZE; i++)

{

min=i;

for(j=i+1; j<SIZE; j++)

if(Arreglo[j]<Arreglo[min])

min=j;

temp= Arreglo[i];

Arreglo[i]=Arreglo[min];

Arreglo[min]=temp;

}

}

Heapsort

#include <stdio.h>

#define SIZE 10

void makeheap(int x [], int n);

void heapsort (int x[], int n);

int main(void)

{

int Arreglo[SIZE], i;

for(i=0; i<SIZE;i++)

{

printf("Ingrese el elemento %d:\n",i+1);

scanf(" %d", &Arreglo[i]);

}

makeheap(Arreglo,10);

heapsort(Arreglo,10);

printf("Despues del Heapsort de elementos\n");

for(i=0; i<SIZE;i++)

printf("%d\t",Arreglo[i]);

printf("\n");

getchar();

}

void makeheap(int x [], int n)

{

int i, val, s, f;

for(i=1; i<n; i++)

{

val = x[i];

s=i;

f = (s-1)/2;

while (s>0 && x[f]<val)

{

x[s] = x[f];

s= f;

f= (s-1)/2;

}

x[s]= val;

}

}

void heapsort (int x[], int n)

{

int i, s, f, ivalue;

for(i=n-1; i>0; i--)

{

ivalue= x[i];

x[i]= x[0];

f= 0;

if(i==1)

s= -1;

else

s=1;

if(i>2 && x[2] > x[1])

s=2;

while (s>=0 && ivalue < x[s])

{

x[f] = x[s];

f = s;

s = 2 \* f + 1;

if(s+1<=i && x[s] < x[s+1])

s++;

if(s>i-1)

s=-1;

}

x[f] = ivalue;

}

}

Quicksort

#include <stdio.h>

#define SIZE 10

int Particion(int Arreglo[], int l, int r);

void QuickSort(int Arreglo[], int l, int r);

int main(void)

{

int Arreglo[SIZE], i;

for(i=0; i<SIZE;i++)

{

printf("Ingrese el elemento %d:\n",i+1);

scanf(" %d", &Arreglo[i]);

}

QuickSort(Arreglo,0,9);

printf("Despues del QuickSort de elementos\n");

for(i=0; i<SIZE;i++)

printf("%d\t",Arreglo[i]);

printf("\n");

getchar();

}

void QuickSort(int Arreglo[], int l, int r)

{

int j;

if(l < r) //dividir

{

j= Particion(Arreglo,l,r);

QuickSort(Arreglo,l,j-1);

QuickSort(Arreglo,j+1,r);

}

}

int Particion(int Arreglo[], int l, int r)

{

int pivote, i,j,t;

pivote = Arreglo[l];

i = l;

j = r+1;

while(1)

{

do

++i;

while((Arreglo[i] <= pivote) && (i <= r));

do

--j;

while(Arreglo[j]> pivote);

if(i >= j)

break;

t= Arreglo[i];

Arreglo[i]= Arreglo[j];

Arreglo[j]= t;

}

t= Arreglo[l];

Arreglo[l]= Arreglo[j];

Arreglo[j]= t;

return j;

}

Máquinas de Estado.

Local.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Archivo: local.h           \*

\* Descripcion: Archivo de encabezado para maquina de estados \*

\*    Aqui se definen todos los tipos de datos y  \*

\*    constantes que se utilizan en el programa \*

\*                \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CONSTANTES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* En esta seccion se declaran todas las   \*

\* las constantes que se necesiten en el programa \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define BUFFER 128

#define L\_EVENT\_ARGS 21

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ESTRUCTURAS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* En esta seccion se declaran todas las  \*

\* estructuras que se necesiten en el programa  \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Este tipo de datos se utiliza para leer la informacion

// que teclea el usuario

struct event\_s

{

int etype;

char args[L\_EVENT\_ARGS];

};

typedef struct event\_s EVENT;

// Este tipo de datos se utiliza para implementar

// la tabla auxiliar

struct aux\_table\_s

{

int (\*action)();

int moreacts;

int nextstate;

};

typedef struct aux\_table\_s AUX\_TAB;

// Este tipo de datos se utiliza para implementar

// la tabla de accion

struct action\_tab\_s

{

int event;

int (\*action)();

int moreacts;

int nextstate;

};

typedef struct action\_tab\_s ACTION\_TAB;

// Este tipo de datos se utiliza para implementar

// la tabla de estados

struct state\_tab\_s

{

int state;

int start;

int end;

};

typedef struct state\_tab\_s STATE\_TAB;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GLOBALES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* En esta seccion se pueden declarar todas  \*

\* variables globales que se necesiten en el programa \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

FILE \*fp;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EVENTOS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* En esta seccion se declarar como constantes   \*

\* los eventos (entradas) que recibe el programa \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define ENTRADA\_0 0

#define ENTRADA\_1 1

#define ENTRADA\_3 3

#define ENTRADA\_5 5

#define asterisco -1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ESTADOS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* En esta seccion se declarar como constantes  \*

\* los estados de que consta el programa  \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define Inicio 0

#define EdoA 1

#define EdoB 2

Tables.h

\* \*

\* Archivo: tables.h \*

\* Descripcion: Tablas para la máquina de estados y funciones externas \*

\* Aqui se definen todos las tablas que se utilizan \*

\* en el programa \*

\* \*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ARCHIVOS DE INCLUSION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "local.h"

#include <stdio.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNCIONES EXTERNAS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* En esta seccion se declaran los encabezados de \*

\* todas las funciones que se utilizan en las \*

\* tablas del programa. Todas las funciones deben \*

\* ser de tipo entero y no deben recibir parametros \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern int msg\_aEdoA(void);

extern int msg\_aEdoB(void);

extern int msg\_SalA(void);

extern int msg\_SalB(void);

extern int nul(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TABLAS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* En esta seccion se declaran las tablas \*

\* que se van a usar en el programa \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TABLA DE ACCION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ACTION\_TAB action\_table[]={

/\*etype accion bandera sigEdo. \*/

{ ENTRADA\_0, msg\_aEdoA, -1, EdoA},

{ ENTRADA\_1, msg\_aEdoB, -1, EdoB},

{ asterisco, nul, -1, Inicio},

{ ENTRADA\_3, msg\_SalA, -1, Inicio},

{ asterisco, nul, -1, EdoA},

{ ENTRADA\_5, msg\_SalB, -1, Inicio},

{ asterisco, nul, -1, EdoB},

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TABLA AUXILIAR \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ //No borrar aunque no se utilice.

AUX\_TAB aux\_table[]={

/\* accion bandera sigEdo \*/

{ nul, 0, 0},

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TABLA DE ESTADOS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

STATE\_TAB state\_table[]={

/\*estado inicio fin \*/

{Inicio, 0, 2},

{EdoA, 3, 4},

{EdoB, 5, 6},

};

Principal.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*                \*

\* Archivo: state\_mach1.c         \*

\* Descripcion: Codigo Fuente para implementar el programa  \*

\*    de maquina de estados.        \*

\*                \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ARCHIVOS DE INCLUSION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "tables.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* VARIABLES GLOBALES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

EVENT event;

int state;

char buf[BUFFER];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PROTOTIPOS DE FUNCION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initialise(void);

void getevent(void);

int msg\_aEdoA(void);

int msg\_aEdoB(void);

int msg\_SalA(void);

int msg\_SalB(void);

int nul(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNCION PRINCIPAL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char \*\*argv)

{

int actx, auxx, outcome;

initialise();

while (1) { /\* loop infinito para la MFE \*/

getevent(); // leer el evento

for ((actx = state\_table[state].start);(action\_table[actx].event != event.etype) && (actx < state\_table[state].end);actx++)

;

outcome = (\*(action\_table[actx].action))();

if(action\_table[actx].moreacts == -1)

state = action\_table[actx].nextstate;

else {

auxx = action\_table[actx].moreacts + outcome;

while (auxx != -1){

outcome = (\*(aux\_table[auxx].action))();

if (aux\_table[auxx].moreacts == -1 ){

state = aux\_table[auxx].nextstate;

auxx = -1;

}

else

auxx = aux\_table[auxx].moreacts + outcome;

}

}

} /\* while(1) \*/

}

void initialise(void)

{

state = 0;

}

void getevent(void)

{

char \*ptmp;

ptmp = &buf[2];

#ifdef DEBUG

printf("wait event \n");

#endif

gets(buf);

switch (buf[0])

{

case 'A' :

event.etype=ENTRADA\_0;

break;

case 'B' :

event.etype=ENTRADA\_1;

break;

     case 'a' :

event.etype=ENTRADA\_3;

break;

case 'b' :

event.etype=ENTRADA\_5;

break;

    default:

     event.etype=-1;

     break;

}//switch

}// getevent

/\* FUNCIONES DE IMPLEMENTACION \*/

int msg\_aEdoA(void)

{

printf("Entró al estado A\n");

}//funcion1

int msg\_aEdoB(void)

{

printf("Entró al estado B\n");

}//funcion2

int msg\_SalA(void)

{

printf("Salió al estado A\n");

}//funcion3

int msg\_SalB(void)

{

printf("Salió del estado B\n");

}//funcion4

int nul(void)

{

printf("No hace nada jeje\n");

}//nul