Clase 7. Estructuras, typedef, y listas ligadas.

Tarea 4

Tarea 4

prog4.1 Realice un programa que lea y escriba un tipo de dato generico en una estructura. La estructura puede almacenar: 1) Una variable tipo enum que indique si el dato es: a) una cadena de caracteres, b) un entero, c) un flotante/real.
2) Una función que lea un valor de la entrada estándar o de archivo, requiera memoria dinámica para la estructura, identifique que tipo de valor es y lo almacene en el lugar correspondiente: si tiene punto decimal o exponente lo toma como flotante, sino tiene punto decimal ni exponente es entero, si tiene al menos un caracter es cadena de caracteres. Ejemplo:
3.17 es flotante
5 es entero
5.0 es flotante
5e1 es flotante
hola476 es cadena de caracteres
La función devuelve el apuntador a la memoria requerida. (No olvide liberar la memoria). 3) Imprima a salida estándar o archivo el contenido del dato con una leyenda. Ej.: `El dato es de tipo ENTERO y su valor es de 5.`
Nota: Puede suponer que la longitud máxima de la palabra es de 128 carácteres, Utilice typedef para definir el tipo de la estructura, y un tipo que sea apuntador al tipo de la estructura.

struct

struct

Un tipo de dato compuesto tipo struct, es un tipo definido por el usuario, que nos permite *empaquetar* bajo un mismo nombre de variable un conjunto de datos de diferente tipo.
Ejemplo:

```
1 struct NOMBREDELTIPO{
  tipo_del_miembro1 miembro1;
3 tipo_del_miembro2 miembro2;
  tipo_del_miembro3 miembro3;
5 };
```

struct

Para declarar una variable de la estructura (y un tipo enum es similar) se tiene que anteponer struct.
Ejemplo:

```
1  struct ESTRUCTURA{
   char color;
3  double x,y;
   int pos[2];
5  };

7  int main()
   {
9    struct ESTRUCTURA memoriaEstructura;
   return 0;
11 }
```

struct

Para acceder a los miembros de la estructura se utiliza el operador **(.)**. Que es un punto entre el nombre de la estructura y el miembro al que se desea acceder.
Ejemplo:


```c
struct ESTRUCTURA{
char color;
double x,y;
int pos[2];
};
int main()
{
   struct ESTRUCTURA memEstruct;
   memEstruct.color='b';
   memEstruct.x=7.8;
   memEstruct.y=4.9;
   memEstruct.pos[0]=9;
   memEstruct.pos[1]=-23;
   printf("color=%c\n",memEstruct.color);
   printf("x=%g\n",memEstruct.x);
   printf("y=%g\n",memEstruct.y);
   printf("pos[0]=%d\n",memEstruct.pos[0]);
   printf("pos[1]=%d\n",memEstruct.pos[1]);
return 0;
}
```

memoria en la estructura

La memoria de la estructura está en el mismo bloque, por lo que puede utilizarse para eficientar un programa (poner junta la memoria de datos que se ocupan al mismo tiempo).


```
struct ESTRUCTURA{
char color;
double x,y;
int pos[2];
};
int main()
{
   struct ESTRUCTURA est;
   printf("&est=%p\n",&est);
   printf("&color=%c\n",&est.color);
   printf("&x=%g\n",&est.x);
   printf("&y=%g\n",&est.y);
   printf("&pos[0]=%d\n",&est.pos[0]);
   printf("&pos[1]=%d\n",&est.pos[1]);
return 0;
}
```

```
&est=0x7fff5a5ab550
&color=0x7fff5a5ab550
&x=0x7fff5a5ab558
&y=0x7fff5a5ab560
&pos[0]=0x7fff5a5ab568
&pos[1]=0x7fff5a5ab56c
```

struct

Otra forma de declarar una variable es escribirla antes del punto y coma de la definición del tipo. Ejemplo:


```
struct ESTRUCTURA{
char color;
double x,y;
int pos[2];
}memEstruct;
int main()
{
  memEstruct.color='b';
  memEstruct.x=7.8;
  memEstruct.y=4.9;
  memEstruct.pos[0]=9;
  memEstruct.pos[1]=-23;
  printf("color=%c\n",memEstruct.color);
  printf("x=%g\n",memEstruct.x);
  printf("y=%g\n",memEstruct.y);
  printf("pos[0]=%d\n",memEstruct.pos[0]);
  printf("pos[1]=%d\n",memEstruct.pos[1]);
return 0;
}
```

Funciones que reciben y devuelven estructuras


```
1  struct ESTRUCTURA{
   char color; double x[2];
3  }var;
   struct ESTRUCTURA funcion(struct ESTRUCTURA var1)
5  {
     var1.color='x';
7    var1.x[0]=3.1415; var1.x[1]=1.3;
     printf("Memoria de funcion=%p\n",&var1);
9    return var1;
   }
11 int main()
   {
13   struct ESTRUCTURA var2= funcion(var2);
     printf("Memoria en main=%p\n",&var2);
15 return 0;
   }
```

Salida:

```
Memoria de funcion=0x7ffff72cdc90
Memoria en main=0x7ffff72cdcb0
```

typedef

typedef Es una palabra reservada, dedicada exclusivamente para darle un nuevo nombre a un tipo. En el caso de estructuras (o de apuntadores) es muy util para hacer un tipo nuevo que nos permita simplificar la programacion. Usualmente estos nuevos tipos se definen en mayusculas (par que el programador sepa que son un tipo definido por el usuario). Ejemplo.

```
enum TIPO{A,B,C,D};
struct ESTRUCTURA{
char color; double x[2];};
typedef unsigned char BYTE;
typedef enum TIPO TIPOENUM1;
typedef enum{one,two} TIPOENUM2;
typedef struct ESTRUCTURA TIPOE1;
typedef struct {
char color; float x[2];
}TIPOE2;
```

```
int main()
{
    BYTE c;
    TIPOENUM1 x;
     TIPOENUM2 y;
    TIPOE1 est1;
     TIPOE2 est2;
return 0;
}
```

Cantidad de memoria en el struct


```
struct ESTRUCTURA{
char color; double x[2]; };
typedef struct ESTRUCTURA TIPOEST1;
int main(int argc, char *argv[])
{

  TIPOEST1 e;
  printf("sizeof(TIPOEST1)=%d\n",sizeof(TIPOEST1));
  printf("&color=%p &x[0]=%p &x[1]=%p\n",&e.color,&e.x
    [0],&e.x[1]);
  return 0;
}
```

Salida:

```
sizeof(TIPOEST1)=24
&color=0x7fffcd268060 &x[0]=0x7fffcd268068 &x[1]=
```

¿Porqué son 24 bytes?

struct y apuntadores

La memoria para una struct se requiere de la misma forma que para cualquier otra variable, el inicio de la memoria se puede almacenar en un apuntador: Se puede acceder a los miembro por medio del operador $->$.

```
1  #include <stdio.h>
   #include <stdlib.h>
3  typedef struct {char color; float x[2];}TIPOEST;
   typedef TIPOEST* PTREST;
5  int main()
   {
7      PTREST ptr=NULL;
       ptr=(PTREST)malloc(sizeof(TIPOEST));
9      ptr->color='w';
       ptr->x[0]=8.93;
11     ptr->x[1]=-6.75;
       printf("color=%c x[0]=%")
13 return 0;
   }
```

Aritmética de apuntadores en estructuras

Al declarar el tipo de la estructura el compilador sabe cuanto debe de recorrer al aumentar uno el apuntador. En el ejemplo de abajo (ejecutenlo) la estructura es de tamaño de 24 bytes, el apuntador var2 comienza en cero y cuando le sumamos 1 se va a 0x18 (recuerde que es hexadecimal, entonces es 24 en decimal).

```
1 struct ESTRUCTURA{
  char color; double x[2];
3 }var;

5 int main()
  {
7   struct ESTRUCTURA* var2= NULL;
    printf("sizeof(estructura)=%d Memoria en main=%p %p\n
      ",sizeof(struct ESTRUCTURA),var2,var2+1);
9 return 0;
  }
```

Arreglos de estructuras


```
typedef struct {
char color; double x[2];
}EST;
typedef ESTR* PTREST;
int main()
{
   int n=10,m=5;
   PTREST vector=(PTREST)malloc(n*sizeof(EST));
   PTREST *matriz=(PTREST*)malloc(n*sizeof(PTREST));
   for (int i=0; i<n; i++)
       matriz[i]=(PTREST)malloc(m*sizeof(EST));
   vector[1].color='z';
   printf("color[1]=%c\n",vector[1].color);
   matriz[2][3].x[1]=0.328571;
   printf("x[2][3][1]=%lf\n",matriz[2][3].x[1]);
   free(vector);
   for (int i=0; i<n; i++)
       free(matriz[i]);
   free(matriz);
return 0;
}
```

atoi y atof

Las funciones atoi y atof estan en stdlib.h
int atoi (const char * str); convierte una cadena de caracteres en un entero. Descarta los espacios en blanco hasta que encuetra un espacio que no este en blanco, si la entrada no representan un entero el comportamiento no está definido.
double atof (const char* str); convierte una cadena de caracteres en un double.


```
1  int main(int argc, char *argv[])
{
3    typedef char CAR;

5    CAR str[12]={"3.141529e-2"};

7    printf("string=%s double=%lf entero=%d\n",str, atof(
       str),atoi(str));
     CAR str2[12]={"-1358"};
9    printf("string=%s double=%lf entero=%d\n",str2, atof(
       str2),atoi(str2));

11   return 0;
}
```

Tarea 4 Segunda parte

Tarea 4

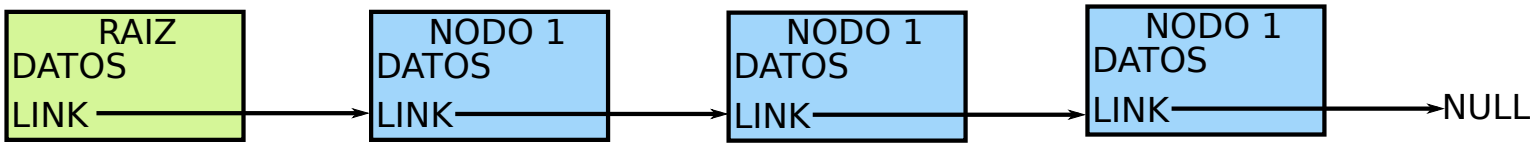
prog4.2 Escriba un programa que dada una lista de datos (un dato es del tipo que se uso en el programa 4.1), requiera y libere la memoria dinámica para esa lista. En el main se declarará solo un apuntador **root** a partir del cual se accedera a toda la lista, cuando root es igual con NULL quiere decir que la lista está vacía. La lista crecerá hasta que se encuentre la paralabra **END**. La función para leer e imprimir la lista debe de ser iterativa.

prog4.3 Escriba un programa que:

1. Despliegue un menú cíclico, con las opciones de a) **PUSH** insertar dato a lista (siempre al final) b) (POP) extraé y eliminar el último dato de la lista, c) (SEEK) búsque y extraiga(elimine) un dato de la lista. d) (DEL) vacie y borre la lista (Comenzando del último elemento) e) (PRINT) Imprima la lista f) (REVERSE) Reconecte la lista en sentido inverso f) salir.
   Siempre hay que liberar la memoria del programa al salir. Los datos en la lista son del mismo tipo que la tarea 4.1. Para la búsqueda implemente una funcion **compara(a,b)** que compara a y b, regresa cero si son iguales, mayor que cero si a es mayor que b y menor que cero si b es mayor que a. Para las palabras utilice el orden dado por las letras casteadas a entero. La comparación entre un flotante y un entero se hace casteando el entero a flotante (o double), la comparacion entre un número y una palabra se hace hace casteando el primer caracter de la palabra a flotante y se aplica lo anterior.

Lista ligada

Una lista ligada es una estructura de datos, que consiste en nodos ligados. Cada nodo contiene uno o mas datos, y una referencia al siguiente nodo. Las listas simplemente ligadas contienen solo una referencia al siguiente nodo.



En C, comunmente, un nodo es una estructura, y la referencia es un apuntador.

Referencias

Estructuras:
http://www.tutorialspoint.com/cprogramming/c_structures.htm
http://www.cprogramming.com/tutorial/c/lesson7.html
http://www.cs.usfca.edu/~wolber/SoftwareDev/C/CStructs.htm
Punteros a estructuras:
http://www.programiz.com/c-programming/c-structures-pointers
http://www.tutorialspoint.com/cprogramming/c_structures.htm
atoi y atof:
http://www.cplusplus.com/reference/cstdlib/atof/
http://www.cplusplus.com/reference/cstdlib/atoi/
Listas:
http://www.cprogramming.com/tutorial/c/lesson15.html