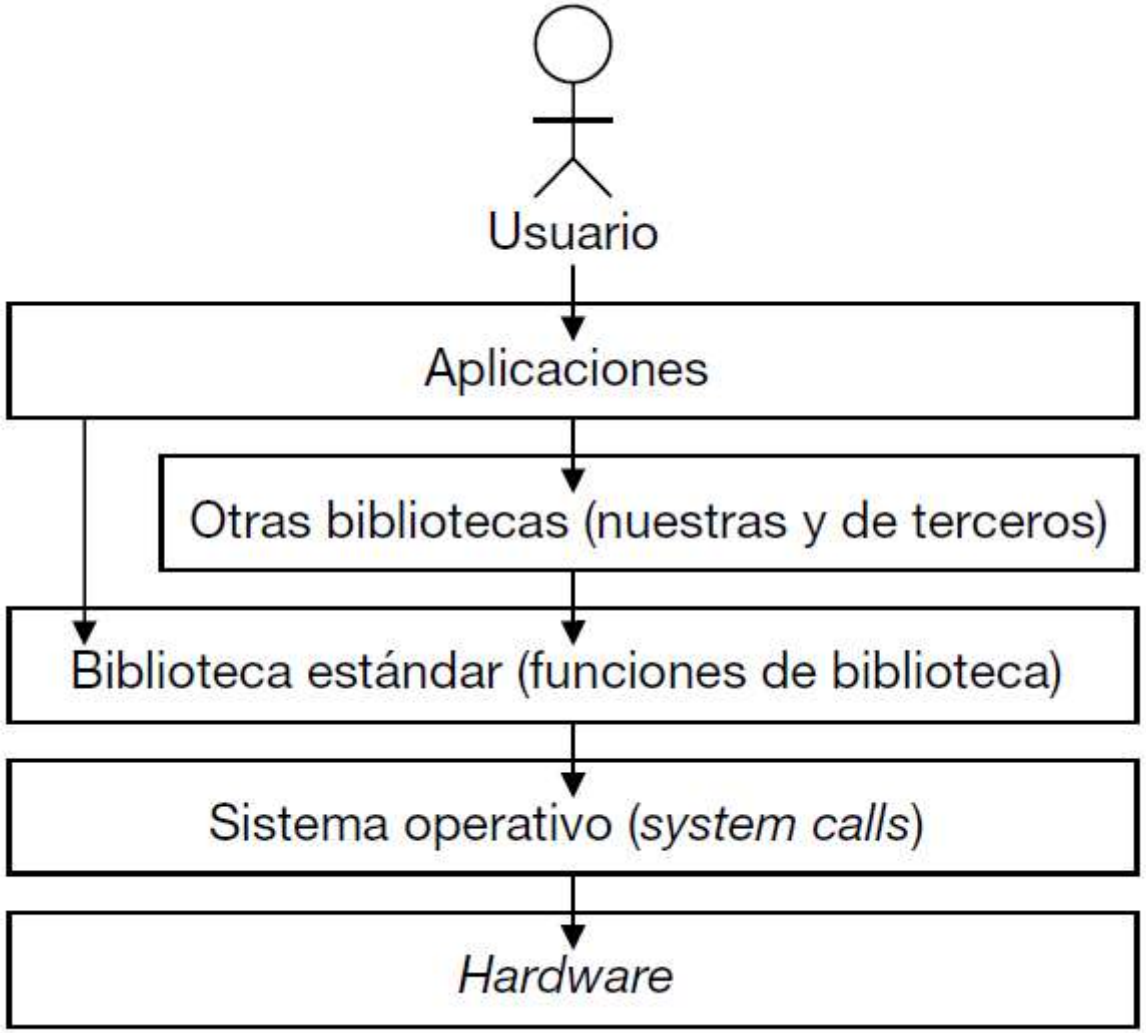
1.5 Tipo de dato abstracto

1.5.1 Capas de abstracción

- Cuando se programan aplicaciones, esto se realiza sobre una capa de abstracción que permite al programador asumir cuestiones de plataforma que, en la mayoría de los casos, resultan totalmente ajenas al contexto en que se encuentra el programador.
- Funciones como printf, scanf, fread, fwrite, etc. Se incluyen como parte de la biblioteca estándar del lenguaje de programación.

1.5.1 Capas de abstracción



1.5.2 Tipo de dato Abstracto (T.D.A.)

- Cuando se habla de tipos de datos se tiene que considerar que cada elemento de un tipo de dato debe tener un conjunto de posibles valores asociados y un conjunto de operaciones aplicables
- El concepto de TAD es una extensión del concepto de Tipo de datos en el cual se permite al programador diseñar sus tipos propios para encapsular la lógica algorítmica y proveer abstracción a las capas de software de más alto nivel.

1.5.2 Tipo de dato Abstracto (T.D.A.)

- Ejemplo de ello son las funciones fopen, fclose, fread, fwrite, componen un TAD que encapsula la problemática relacionada con el manejo de archivos.
- La implementación de los Tipos de Dato Abstractos en el lenguaje de programación **C** se hace a través de **estructuras**. Ya que se pueden combinar datos primitivos con otras estructuras y poder generar diferentes niveles de abstracción

1.5.3 Estructuras

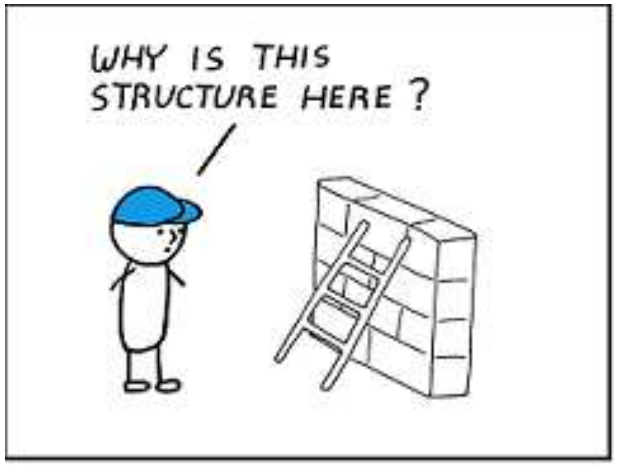
- Una **estructura** es una colección de uno o más elementos denominados miembros, cada uno de los cuales puede ser un tipo de dato diferente.

- Una estructura sirve para definir tipos de datos abstractos

1.5.3 Estructuras

- **Declaración de una estructura**

```
struct <nombre de la estructura> {
    <tipo_dato miembro> <nombre>
    <tipo_dato miembro> <nombre>
    ...
};
```



1.5.3 Estructuras

- El uso de arreglos, apuntadores y estructuras facilitan la formación de nuevos tipos de datos, y tipos de datos más complejos como listas ligadas, las colas, pilas y árboles.

- Los miembros de una estructura deben tener nombres únicos pero dos estructuras diferentes pueden contener miembros con el mismo nombre.

1.5.2 Estructuras

Operaciones

Las operaciones válidas que se pueden realizar cuando se utilizan estructuras son:

- ✓ Asignación de valores a los miembros de estructuras
- ✓ Duplicar estructuras.

Con las estructuras **NO** es posible hacer operaciones aritméticas ni comparaciones porque los miembros de las estructuras no están necesariamente almacenados en bytes de memoria consecutivos.

Ejemplo

```
struct alumno {
    char *nombre;
    int edad;
};

struct alumno alu1;
struct alumno grupo[45];
struct alumno *point;

struct alumno {
    char *nombre;
    int edad;
}alu1, grupo[45], *point;
```

1.5.3 Estructuras

- **Inicializar estructuras**

Se inicializan de manera similar a los arreglos.

```
struct Alumno alu1 = {"pepe", 29};
```

SI en la lista aparecen menos inicializadores que en la estructura, los miembros restantes automáticamente quedarán inicializados a **0** o a **NULL.**

1.5.3 Estructuras

Acceso a los miembros de una estructura

Para tener acceso a miembros de estructuras se utilizan dos operadores:

El operador miembro o punto (.)

Tiene acceso a un miembro de estructura mediante el nombre de la variable de estructura.

El operador de apuntador o flecha (->)

Tiene acceso a una característica de una estructura de manera indirecta (con un apuntador a la misma.)

Ejemplo

```c
#include <stdio.h>

struct Alumno{
    int numCuenta;
    char *nombre;
    char apellido[20];
    float promedio;
};

int main(){
    struct Alumno alumno1;
    alumno1.numCuenta = 30231234;
    alumno1.nombre = "Bart";
    strcpy(alumno1.apellido,"Simpson");
    printf("Nombre: %s \n",alumno1.nombre);
    printf("Apellido: %s  \n",alumno1.apellido);
}
```

Ejemplo

```c
#include <stdio.h>

struct Alumno{
    ...
}alumno1,alumno2;

int main(){
    struct Alumno *ap;
    ap = &alumno1;
    ap->nombre = "Edgar";
    ap->apellido = "Tista";
    printf("Nombre: %s \n",ap->nombre );
    printf("Apellido: %s  \n",alumno1.apellido);
}
```

1.5.2 Estructuras

- Una estructura convencional no contiene un miembro de su mismo tipo.
- Cuando se requiere generar una estructura que se contenga a sí misma, es posible hacerlo mediante un apuntador.
- Una estructura de este tipo se conoce como *estructura autorreferenciada*.

typedef

- La palabra typedef proporciona un mecanismo para la creación de sinónimos o alias para tipos de datos.
- Los nombres de los tipos de estructura se definen utilizando esta palabra reservada a fin de crear nombres de tipo más breves.

Ejemplo:

```
typedef int Entero;
Entero a = 10;
Entero b = 20;
```

typedef - Estructuras

```
struct alumno{
    ...
};

typedef struct alumno Alumno;

int main(){
    Alumno alumno1;
    Alumno alumno2;
    alumno1.nombre = "Bart";
    alumno1.apellido = "Simpson";
    printf("Name: %s \n",alumno1.nombre);
    printf("LastName : %s \n",alumno1.apellido);
}
```

typedef - Estructuras

```
typedef struct {
    int numCuenta;
    char *nombre;
    ...
}Alumno;

main(){
    Alumno alumno1;
    Alumno alumno2;
    alumno1.nombre = "Rafa";
    alumno1.apellido = "Gorgori";
    printf("Name: %s \n",alumno1.nombre);
    printf("LastName : %s  \n",alumno1.apellido);
}
```

1.5.3.1 Estructuras y funciones

- Es una buena practica de programación que el manejo de estructuras se haga a través de funciones, con lo cual se puede reutilizar código, hacer los programas más modulares y facilitar la creación de estructuras de datos más complejas.
- Lo anterior no solo aplica a estructuras, de manera general, el manejo de funciones permite optimizar códigos de aplicaciones complejas.