Introducción

Dos tipos de datos que nos ayudan agrupar son los structs y los arrays.

Structs

Ejemplo de definición y declaración de un struct:

```
#include<stdio.h>

main(){

    struct fraccion{
        int numerador;
        char caracter;
    }; //declaramos el nombre del struct con dos miembros: un int y un char

    struct fraccion f; //definimos y declaramos f: el struct fraccion
    f.numerador = -5; //inicializamos al miembro int
    f.caracter = 'E'; //inicializamos al miembro caracter
    printf("struct fraccion numerador: %d\n", f.numerador);
    printf("scruct fraccion caracter: %c\n", f.caracter);
}
```

Podemos copiar dos structs con el símbolo de = :

```
#include<stdio.h>

main(){

    struct fraccion{
        int numerador;
        char caracter;
    }; //declaramos el nombre del struct con dos miembros: un int y un char

    struct fraccion f1,f2; //definimos y declaramos f: el struct fraccion
    f1.numerador = -5; //inicializamos al miembro int
    f1.caracter = 'E'; //inicializamos al miembro caracter
    f2 = f1;
    printf("struct fraccion numerador: %d\n", f2.numerador);
    printf("scruct fraccion caracter: %c\n", f2.caracter);
}
```

Arrays

Ejemplo sencillo de definición y declaración de un arreglo de enteros:

```
#include<stdio.h>

main(){
    int arreglo1[5]; //declaración y definición
    int arreglo2[3] = {0}; //inicializamos con el valor de cero al arreglo2
    int i;
    //inicialización del arreglo1:
```

```
    for(i=0;i<5;i++){
        arreglo1[i] = i;
    }
    //Imprimimos al arreglo1

    for(i=0;i<5;i++){
        printf("arreglo1[%d] = %d\n",i,arreglo1[i]);
    }

    printf("----------------\n");
    //Imprimimos al arreglo2:
    for(i=0;i<3;i++){
        printf("arreglo2[%d] = %d\n",i,arreglo2[i]);
    }

}
```

Al definir y declarar el arreglo1 de tamaño 5, son designados 5 bloques de memoria contiguos de tamaño int (=4 bytes) que en total son 20 bytes

Podemos obtener la longitud de un arreglo con la función `sizeof`

```
#include<stdio.h>
main(){
    int arreglo[7];
    printf("Tamaño en bytes del arreglo: %ld\n", sizeof(arreglo));
    printf("Tamaño en bytes de la posición 0 de arreglo: %ld\n", sizeof(arreglo[0]));
    printf("Longitud de arreglo: %ld\n", sizeof(arreglo)/sizeof(arreglo[0]));
}
```

Ejemplo para definición, declaración de un arreglo multidimensional.

Observa el número de bytes alojados para cada estructura de datos:

```
#include<stdio.h>
main(){
int arreglo_multidimensional[4][3];
int numero_renglones, numero_columnas;
printf("Total de bytes para arreglo_multidimensional :%ld\n", sizeof(arreglo_multidimensional));
printf("Total de bytes para arreglo_multidimensional[0]: %ld\n",sizeof(arreglo_multidimensional[0]));
printf("Total de bytes para arreglo_multidimensional[0][0]: %ld\n", sizeof(arreglo_multidimensional[0][0
}
```

Apuntadores

Un apuntador es una variable que contiene el address de una variable.

Ejemplo para definición y declaración de un apuntador hacia una variable `int`.

```
#include<stdio.h>
main(){

    int *p; //definición y declaración de un apuntador hacia una
            //variable tipo int.
}
```

No hay restricción para el lugar donde quisiéramos poner el operador `*`:

```
#include<stdio.h>
main(){

    int *p1; //definición y declaración de un apuntador hacia una
            //variable tipo int.
    int * p2;
    int* p3;
}
```

Observemos su tamaño:

```
#include<stdio.h>
main(){

    int *p; //definición y declaración de un apuntador hacia una
            //variable tipo int.

    printf("Total de bytes para apuntador: %ld\n", sizeof(p));
}
```

Con el operador `&` podemos obtener el address de una variable:

```
#include<stdio.h>
main(){

    int variable;
    printf("Address de variable int: %p\n", &variable);
}
```

Es fundamental `inicializar` a un apuntador. Una forma es con el operador `&` y recordando que el valor de un apuntador es un addresss.

Observa que `variable` es tipo `int`, por lo que al declarar y definir a un apuntador, es necesario considerar esto último.

```
#include<stdio.h>
main(){
    int variable;
    int *p;
    p = &variable; //inicializamos al apuntador p
    printf("Address de variable int: %p\n", &variable);
    printf("Address de apuntador: %p\n", p);
}
```

Otra forma de `inicializar` a un apuntador:

```
#include<stdio.h>
main(){
    int variable;
    int *p = &variable; //inicializamos al apuntador p
    printf("Address de variable int: %p\n", &variable);
    printf("Address de apuntador: %p\n", p);
}
```

Para un apuntador `inicializado` es posible aplicar el operador `*`, que aplicado a un apuntador, se accede al objeto al que el apuntador apunta:

```
#include<stdio.h>
main(){
```

```
    int variable = 5;
    int *p = &variable; //inicializamos al apuntador p
    printf("Address de variable int: %p\n", &variable);
    printf("Address de apuntador: %p\n", p);
    printf("Accedemos al objeto con *p: %d\n", *p);
}
```

La operación `*p` se conoce como `dereference p`.

Otro ejemplo:

```
#include<stdio.h>
main(){
    int a = -1, b = 4, arreglo[5];
    int *p;
    p = &b;
    a = *p;
    *p = -321;
    arreglo[0] = -2;
    p = &arreglo[0];
    printf("Valor de a: %d\n", a);
    printf("Valor de b: %d\n", b);
    printf("Valor de arreglo[0]: %d\n", arreglo[0]);
    printf("Valor de *p: %d\n", *p);
}
```

El nombre de un arreglo funciona como un apuntador:

```
#include<stdio.h>
main(){
    int arreglo[5];
    printf("Nombre del arreglo: %p\n", arreglo);
}
```

El nombre `arreglo` apunta al base address del arreglo. En este base address se guardará un `int`.

Entonces:

```
#include<stdio.h>
main(){
    int arreglo[5];
    *arreglo = 8;
    printf("arreglo[0]: %d\n", arreglo[0]);
}
```

imprime 8.

También podemos obtener la dirección de memoria de un arreglo con el operador `&`:

```
#include<stdio.h>
main(){
    int arreglo[5];
    printf("Dirección de memoria: &arreglo[0]: %p\n", &arreglo[0]);
    printf("Dirección de memoria: arreglo: %p\n", arreglo);

}
```

Los apuntadores son variables y pueden ser utilizados sin realizar un `dereference`. Por ejemplo, es posible asignar a un apuntador el valor de otro apuntador con `=`:

```
#include<stdio.h>
main(){
    int *apuntador1, *apuntador2;
    int variable = -5;
    apuntador1 = &variable;
    apuntador2 = apuntador1;
    *apuntador2 = 10;
    printf("Valor de variable: %d\n", variable);
}
```

Si tenemos un arreglo definido y declarado, los operadores + y - los podemos usar para los apuntadores. Por ejemplo:

```
#include<stdio.h>
main(){
    int arreglo[3];
    int *apuntador;
    apuntador = arreglo;
    printf("Dirección de memoria de arreglo: %p\n", arreglo);
    printf("Dirección de memoria de apuntador: %p\n", apuntador);
    apuntador = apuntador + 1;
    printf("Dirección de memoria de apuntador+=1 : %p\n", apuntador);
    apuntador = apuntador -1;
    printf("Dirección de memoria de apuntador-=1: %p\n", apuntador);
}
```

Para este ejemplo anterior, tenemos que la operación `apuntador + 1` devuelve un apuntador del mismo tipo y apunta hacia un objeto que está un bloque de tamaño `int` a la `derecha` del base address del arreglo. Se tiene un análogo comportamiento para la operación `apuntador - 1`.

Podemos imprimir las direcciones de memoria de arreglo con esta idea:

```
#include<stdio.h>
main(){
    int arreglo[5];
    int i;
    for(i=0;i<sizeof(arreglo)/sizeof(arreglo[0]);i++)
        printf("posición %d, memoria: %p\n", i, arreglo+i);
}
```

Así, tenemos dos formas de recorrer un arreglo (como mínimo):

```
#include<stdio.h>
main(){
    int arreglo[4] = {-3, 5, 7, 8};
    int i;
    for(i=0;i<sizeof(arreglo)/sizeof(arreglo[0]);i++)
        printf("arreglo[%d]=%d\n", i,arreglo[i]);
    printf("-----------------\n");
    for(i=0;i<sizeof(arreglo)/sizeof(arreglo[0]);i++)
        printf("arreglo[%d]=%d\n", i,*(arreglo + i));
}
```

Es válido restar dos apuntadores que apuntan hacia diferentes direcciones de memoria de un arreglo:

```
#include<stdio.h>
main(){
    int arreglo[3];
```

```
    int *apuntador, *apuntador2;
    apuntador = arreglo;
    apuntador2 = arreglo+3;
    printf("apuntador2 - apuntador: %ld\n", apuntador2-apuntador); // El resultado es la "distancia" en
}
```

También es valido comparar dos apuntadores que apuntan hacia diferentes direcciones de memoria de un arreglo:

```
#include<stdio.h>
main(){
    int arreglo[3];
    int *apuntador, *apuntador2;
    apuntador = arreglo;
    apuntador2 = arreglo+3;
    printf("apuntador < apuntador2: %d\n", apuntador < apuntador2);
    printf("apuntador > apuntador2: %d\n", apuntador > apuntador2);
    printf("apuntador == apuntador2: %d\n", apuntador == apuntador2);
}
```

El nombre de un arreglo como se dijo, es un apuntador al base address del arreglo. Sin embargo no podemos asignar `=`, modificar la dirección de memoria a la que apunta. No son válidos:

```
...
    int arreglo[5];
    int arreglo2[10];
    arreglo = arreglo2;
...
```

```
...

    int arreglo[5];
    arreglo = arreglo + 1;
...
```

Observa que tenemos una restricción para los apuntadores: deben apuntar a un tipo en particular de objeto, es decir, cada apuntador apunta a un tipo de dato específico. (Una excepción es un apuntador hacia `void`, el cual se utiliza para indicar que un apuntador no apunta hacia un tipo en particular de dato, y no puede ser dereferenced).

```
#include<stdio.h>
main(){
    int variable, variable2;
    void *apuntador;
    variable = -10;
    apuntador = &variable;
    printf("Valor de apuntador2: %p\n", apuntador);
    printf("Dirección de memoria variable: %p\n", &variable);
    variable2 = *apuntador; //warning y posible error¡¡¡
    printf("Valor de variable2: %d\n", variable2);
}
```

Pero si podemos convertir "cast" un apuntador de tipo `void` hacia otro tipo, por ejemplo `int`:

```
#include<stdio.h>
main(){
    int variable, *apuntador;
    void *apuntador2;
```

```
    variable = -10;
    apuntador2 = &variable;
    printf("Valor de apuntador2: %p\n", apuntador2);
    printf("Dirección de memoria variable: %p\n", &variable);
    apuntador = (int *)apuntador2; //cast
    printf("Dereferenced apuntador: *apuntador: %d\n", *apuntador);
}
```

Otro ejemplo para "cast" de un apuntador a un tipo de dato `void`:

```
#include<stdio.h>
main(){
    long int variable, variable2; //deben ambas variables ser tipo long int y depende de la arquitectura
                                  //sistema en el que se esté trabajando: 32 o 64 bits
    void *apuntador2;
    variable = -10;
    apuntador2 = (void *)variable; //cast a un apuntador de tipo de dato void
    printf("Valor de apuntador2: %p\n", apuntador2);
    printf("Dirección de memoria variable: %p\n", &variable);
    //las dos líneas anteriores imprimen direcciones de memoria distintas
    variable2 = (long int)apuntador2; //cast
    printf("variable: %ld\n", variable2); //no hay pérdida de información del valor de variable
}
```