Introducción

Dos tipos de datos que nos ayudan a agrupar datos: structs y arrays

Structs

Ejemplo de definición y declaración de un struct:

```
#include<stdio.h>

main(){
        struct fraccion{
            int numerador;
            char denominador;
        }; //declaramos el nombre del struct con dos miembros: un int y un char

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

Podemos hacer copias entre structs con el símbolo de =:

```
#include<stdio.h>

main(){
        struct fraccion{
            int numerador;
            char denominador;
        }; //declaramos el nombre del struct con dos miembros: un int y un char

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

}
```

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 al arreglo2 con ceros
    int i;
    //inicializamos al arreglo1:
        for(i=0;i<5;i++)
            arreglo1[i]=i;
```

```
    //imprimimos arreglo1:
        for(i=0;i<5;i++)
            printf("arreglo1[%d]=%d\n",i,arreglo1[i]);
    printf("----------------\n");
    //imprimimos 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 de arreglo: %ld\n", sizeof(arreglo));
    printf("Tamaño en bytes de arreglo posición 0: %ld\n", sizeof(arreglo[0]));
    printf("Longitud de arreglo: %ld\n", sizeof(arreglo)/sizeof(arreglo[0]));

}
```

Arreglo multidimensional

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];
    printf("Total de bytes alojados para arreglo_multidimensional %ld\n",sizeof(arreglo_multidimensional
    printf("Total de bytes alojados para arreglo_multidimensional[0] %ld\n",sizeof(arreglo_multidimensi
    printf("Total de bytes alojados para arreglo_multidimensional[0][0] %ld\n", sizeof(arreglo_multidim
    printf("Número de renglones: %ld\n", sizeof(arreglo_multidimensional)/sizeof(arreglo_multidimensiona
    printf("Número de columnas: %ld\n", sizeof(arreglo_multidimensional[0])/ sizeof(arreglo_multidimensi
}
```

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;// int * p // int* p
}
```

Observemos el número de bytes designados para guardar a un apuntador:

```
#include<stdio.h>
main(){
    int *p;
    int c;
    double *p2;
```

```
    printf("Total de bytes para apuntador: %ld\n",sizeof(p));
    printf("sizeof double p2: %ld\n",sizeof(p2));
    printf("Total de bytes para int:%ld\n",sizeof(c));
}
```

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

```
#include<stdio.h>
main(){
    int variable;
    printf("address de variable: %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 address.

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;
    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;
    int *p = &variable;
    printf("Address de variable int: %p\n", &variable);
    printf("Address de apuntador: %p\n", p);
    variable = -3;
    printf("Valor de variable: %d\n", variable);
    printf("Valor de *p: %d\n", *p);
}
```

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

Otro ejemplo. Qué se imprime en las líneas con `printf` ? :

```
#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 de arreglo: %p\n", arreglo);
    printf("Posición cero de arreglo: %p\n", &arreglo[0]);
}
```

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]);

}
```

Podríamos imprimir las direcciones de memoria de arreglo:

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