Las estructuras o registros

Una estructura o registro es un dato de tipo compuesto, es decir que está formado por un conjunto de elementos llamados campos, no necesariamente del mismo tipo y que permiten almacenar una serie de datos relacionados entre sí bajo un nombre común.

El tipo de datos registro es el que sirve entonces para guardar información de distinto tipo, en general relacionados lógicamente en una estructura única o bajo un mismo identificador. De aquí, sus características básicas:

- Tiene un identificador que representa la estructura.
- Se divide en campos.
- Cada campo puede ser de cualquier tipo primitivo.
- Cada campo tiene su identificador.

Definición de una estructura

El ámbito para definir una estructura de tipo registro es, o bien entre las directivas de preprocesamiento y la cabecera de la función (main) o dentro del cuerpo de la función al momento de declarar las variables.

La forma general de definir una estructura es la siguiente:

```
struct<Identificador_Etiqueta> //Nombre la estructura
    {
        //Campos de la estructura
        <Especificador_de_tipo_1><Identificador_1>;
        <Especificador_de_tipo_2><Identificador_2>;
        <Especificador_de_tipo_3><Identificador_3>;
        ...
        <Especificador_de_tipo_4><Identificador_n>;
    };
```

Nos referimos a definición de estructura porque en este punto del código no se han declarado variables. Solo se ha definido la forma de la estructura que pasará a ser un especificador de tipo o un nuevo tipo de dato.

Declaración de una variable

Como mencionamos anteriormente, cuando se define una estructura esencialmente se está definiendo un tipo de dato compuesto de diferentes elementos o campos. No existe realmente aún una variable de ese tipo hasta que se declara.

La forma general de declarar una variable de este tipo es:

```
struct<Identificador_Etiqueta><Identificador_variable>;
```

De esta forma se declara una variable de tipo <Identificador_Etiqueta> con el nombre o identificador de variable que hayamos designado.

Para aclarar mejor esto veamos un ejemplo concreto:

```c
#include <stdio.h>
#include <stdlib.h>
struct datos//Definición de la estructura
{
    //Campos de la estructura
    int Legajo;
    int edad;
};
int main()
{
    struct datos alumno;//Declaración de una variable de tipo "datos"
    ...
return 0;
}
```

Se define una estructura denominada "datos", y luego se declara una variable "alumno" que tiene la estructura definida anteriormente.

Como pueden observar la estructura está definida antes de main(), y en la primera línea de código dentro del cuerpo de la función se define la variable, utilizando el nuevo tipo de dato, en este caso denominado "datos".

Segun lo expresado anteriormente podríamos definir la estructura al momento de declarar las variables es decir dentro de la función main, por ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    struct datos //Definición de la estructura "datos"
    {
        //Campos de la estructura
        int Legajo;
        int edad;
    } alumno; //Declaración de una variable alumno de tipo "datos"
    ...
return 0;
}
```

Si bien los dos ámbitos son válidos, no se suele definir la estructura en el propio cuerpo de main dado que el alcance de dicha definición se limita al cuerpo de la función principal.

Por lo general necesitamos aquella estructura como tipo de dato en otras funciones; por lo cual es mejor definirla fuera de main y que dicha estructura quede disponible como tipo de dato, para ser utilizada por cualquier otra función.

Uso de typedef

El lenguaje C permite definir un nuevo nombre de tipo de dato usando la palabra reservada *typedef*. Es muy común asignarle un alias o sinónimo al nombre de la estructura, para evitar poner la palabra reservada *struct* cada vez que utilizamos una variable de este tipo.

Veamos un ejemplo con este uso:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct
{
    //Campos de la estructura
    int Legajo;
    int edad;
}datos;//Definición de un "nuevo" tipo de datos estructura
int main()
{
    datos alumno;//Declaración de una variable de tipo "datos"
    ...
return 0;
}
```

Note que en este caso no necesito utilizar la palabra reservada struct en la declaración de la variable.

Operaciones con estructuras

Inicialización

Al momento de declarar las variables podemos inicializar los campos, veamos el siguiente ejemplo:

```c
...
//Definición de la estructura "datos"
struct datos
{
    //Campos de la estructura
    int Legajo;
    int edad;
};
int main()
{
    //Declaracion e inicializacion de una variable
    struct datos alumno = {99050, 25};
    return 0;
}
```

Es importante destacar que, si se inicializa una variable de esta forma, es necesario colocarles un valor a todos los campos de la estructura.

Acceso a los campos

El mecanismo de acceso a cada dato es mediante el nombre de la variable declarada previamente del tipo de estructura deseado, seguido del operador punto (.) y el nombre del campo que se quiere acceder.

Veamos el siguiente ejemplo:

```c
#include <stdio.h>
#include <stdlib.h>

//Definición de la estructura "datos"
struct datos
{
    //Campos de la estructura
    char nombre[10];
    int edad;
};

int main()
{
   //Declaracion e inicializacion de una variable
   struct datos alumno = {"Carlos", 25};

   //Muestra los datos por pantalla
   printf(" %s tiene %d a%cos", alumno.nombre, alumno.edad, 164);
   return 0;
}
```

Veamos un ejemplo completo, en donde se lee una estructura y se muestra por pantalla, utilizando funciones de entrada/salida estándar. En este caso solo leemos un dato y lo mostramos.

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct
{
    char Nombre[10];
    int edad;
} datos;
int main()
{  datos persona;
   printf("Ingrese el nombre de la persona ");
   scanf("%s", persona.Nombre);
   printf("Ingrese la edad de la persona ");
   scanf("%d", &persona.edad);
   printf("%s tiene %d a%cos", persona.Nombre, persona.edad, 164);
  return 0;
}
```

Otra operación muy común a realizar con los registros o estructuras es copiar o asignar un registro en otro, esto se puede hacer íntegramente sin mayor problema, teniendo la precaución de que ambas variables tengan la misma estructura.

A continuación, utilizando el código del ejemplo anterior, mostramos cómo sería esta operación:

```c
#include <stdio.h>
typedef struct
{
    char Nombre[10];
    int edad;
} datos;
int main(){
    datos persona, nuevo;
    printf("Ingrese el nombre de la persona ");
    scanf("%s", persona.Nombre);
    printf("Ingrese la edad de la persona ");
    scanf("%d", &persona.edad);
    nuevo = persona; //Copia en nuevo el registro completo persona
    printf("%s tiene %d a%cos", nuevo.Nombre, nuevo.edad, 164);
    return 0;
}
```

Registros anidadas

De la misma forma que ocurre con ciclos, condiciones, etc. También una estructura puede estar dentro de otra estructura y a esto se le conoce como anidamiento o estructuras anidadas.

Veamos un ejemplo:

```
...
struct infopersona //Definición de la primer estructura
{
    char direccion[25];
    char ciudad[20];
    int codigo_postal;
};
struct empleado //Definición de la segunda estructura
{
  char NombrEmpleado[25];
  //Variable DirecEmpleado con"struct"del tipo "estructura
infopersona"
  struct infopersona DirecEmpleado;
  double Salario;
};
int main()
{
    system("cls");
    struct empleado persona;
    ...
    return 0;
}
```

La variable persona es de tipo empleado que a su vez está compuesta por el tipo infopersona.

Para acceder a los distintos campos, el mecanismo es igual que en un struct simple, es decir mediante el nombre de la variable, seguida del operador punto (.) y el nombre del campo al que se quiere acceder. Solo que en este caso si hago referencia a otro struct se necesita a su vez otro nombre de campo también seguido de un punto y el campo correspondiente al que voy a acceder.

Veamos el siguiente ejemplo de anidamiento utilizando las estructuras creadas anteriormente:

```c
#include <stdio.h>
#include <stdlib.h>
struct infopersona
{
    char direccion[30];
    int numero;
    char ciudad[20];
    int codigo_postal;
};
struct empleado
{
    char NombrEmpleado[25];
    struct infopersona DirecEmpleado;
    float SueldoBasico;
};
int main()
{
    struct empleado persona;

    printf("Ingrese el nombre de la persona ");
    scanf("%s", persona.NombrEmpleado);
    printf("Ingrese la direccion de la persona ");
    fflush(stdin);
    gets(persona.DirecEmpleado.direccion);
    printf("Ingrese el salario de la persona ");
    scanf("%f", &persona.SueldoBasico);

    //Muestra por pantalla los datos del empleado
    printf("%s ", persona.NombrEmpleado);
    printf("vive en %s",persona.DirecEmpleado.direccion);
    printf(" tiene un salario de $ %.2f", persona.SueldoBasico);

return 0;
}
```

Estructuras en arrays

Como ya mencionamos en otro apartado, los vectores permiten agrupar varios elementos de un mismo tipo. Cada elemento de un vector es accesible a través de un índice.
Como también se mencionó anteriormente, los registros o estructuras son agrupaciones heterogéneas de datos cuyos elementos (denominados campos) son accesibles mediante identificadores.
Es posible agrupar un conjunto de elementos de tipo estructura en un array y acceder a diferentes elementos de un grupo a través del identificador con su correspondiente índice.

Veamos el siguiente ejemplo que lee y muestra 10 registros correspondientes a legajos y edad de un grupo de alumnos:

```c
#include <stdio.h>
#include <stdlib.h>
#define n 5
typedef struct
{
    int Legajo;
    int edad;
} datos;
int main ()
{
    datos alumnos[n];//Vector de 10 elementos de tipo "datos"
    system("cls");
    for (int i = 0; i < n; i++) //Ciclo para hacer los 10 ingresos
    {
        printf("Ingrese el legajo del alumno ");
        scanf("%d", &alumnos[i].Legajo);
        printf("Ingrese la edad del alumno ");
        scanf("%d", &alumnos[i].edad);
    }
    printf("legajo\tEdad\n ");
   //Ciclo para mostrar los 10 registros
   for (int i = 0; i < n; i++)
    {
        printf("%d\t%d \n ", alumnos[i].Legajo, alumnos[i].edad);
    }
   return 0;
}
```

Note que al nombre de la variable le sigue el índice entre [] que indica que estoy trabajando con un arrays unidimensional seguido de operador punto (.) y luego el nombre del campo al que voy a acceder.

Estructuras con arrays

Así como los campos de la estructura pueden ser de cualquier tipo de dato primitivo, también pueden ser un array. Veamos a continuación un ejemplo con este uso:

```
#include <stdio.h>
#define n 5
typedef struct
{
    int Legajo;
    int notas[n];//Un campo de tipo vector de enteros
} datos;
int main()
{
    datos alumnos;
    printf("Ingrese el legajo del alumno ");
    scanf("%d", &alumnos.Legajo);
    for (int i = 0; i < n; i++) //Ciclo para leer las 10 notas
    {
        printf("Ingrese la nota %d del alumno ", i + 1);
        scanf("%d", &alumnos.notas[i]);
    }
    printf("\nLegajo %d\n ", alumnos.Legajo);
    for (int i = 0; i < n; i++) //Ciclo para mostrar las 10 notas
    {
        printf("Nota %d: %d\n ", i + 1, alumnos.notas[i]);
    }
    return 0;
}
```