Instituto Politécnico Nacional
La Técnica al Servicio de la Patria

Unidad Profesional Interdisciplinaria de Ingeniería Campus Tlaxcala UPIIT

Fundamentos de Programación

Esaú Eliezer Escobar Juárez

Tipos de datos estructurados

- Tipo de datos que no son simples
- Simples
  - Números
  - Letras
  - Verdadero/falso
- Estructurados o compuestos
  - Combinaciones de tipos simples
  - Los arreglos son tipo de datos estructurados

Tipos de datos definidos por el usuario

- El usuario puede definir sus propios tipos de datos
  - Mayor claridad.
  - Aumenta el significado semántico del código.
  - Simplificar declaración de variables.

Tipos de datos definidos por el usuario

- Typedef
  - Define un nuevo nombre para un tipo de dato.
  - El nombre original sigue siendo válido.

```
typedef <tipo> <nuevo nombre>;
```

```
typedef int positivo;
```

Tipos de datos definidos por el usuario

```
typedef int positivo;
typedef int negativo;

int main(){

  positivo a,b;
  negativo c,d;

  a=1;
  b=2;

  c=-a;
  d=-b;

  printf("%d %d %d %d\n",a,b,c,d);
}
```

Tipos de datos definidos por el usuario

- Otra forma de definir tipos de datos es componer varios datos simples en uno solo.

- Esto se denonima estructura.
  - Una estructura es un tipo de dato que contiene un conjunto de valores.
  - Estos valores pueden ser de distinto tipo.
  - Generalmente, se refiere a un concepto más complejo que un número o una letra.

Estructuras

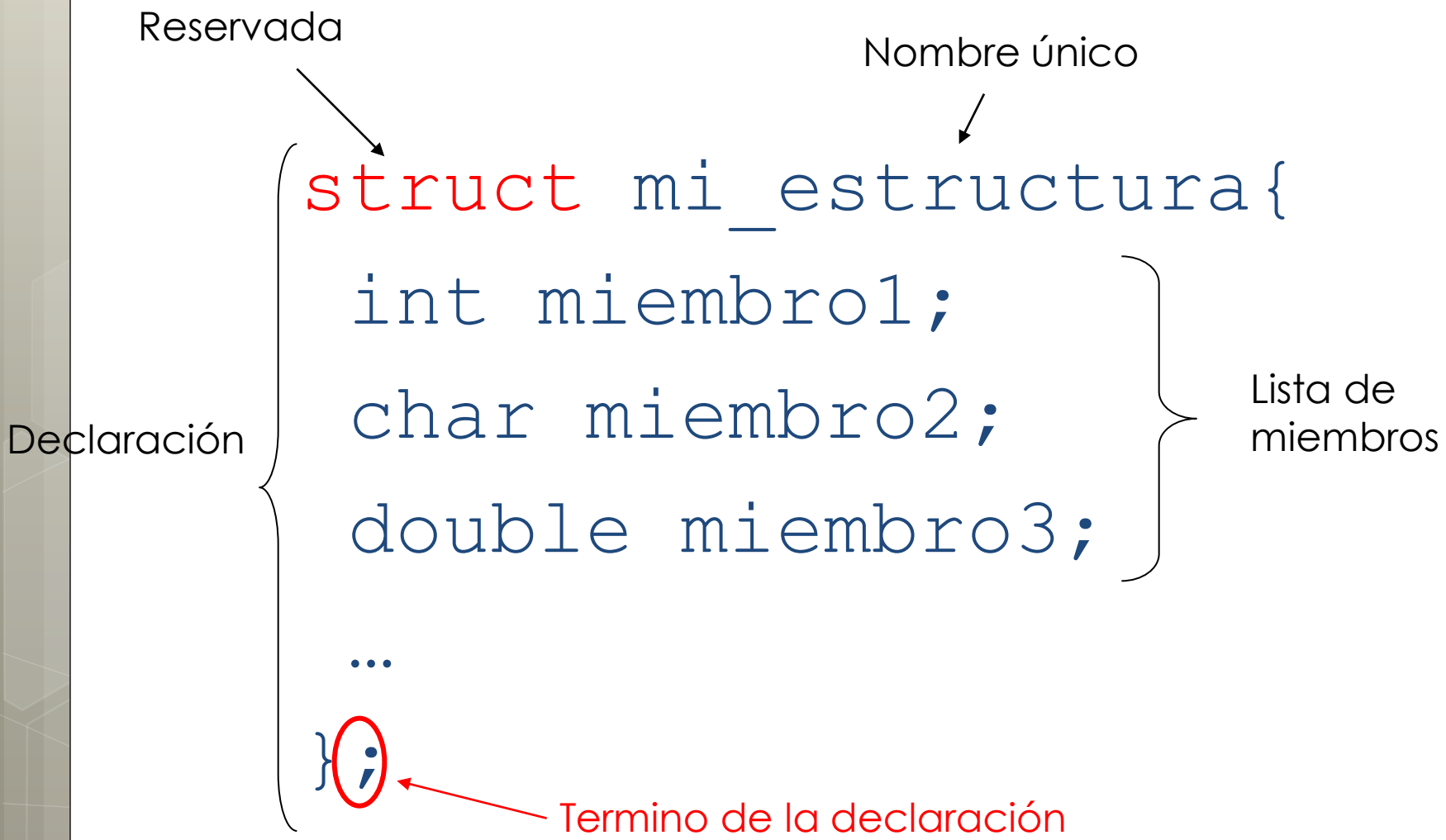
- Es una colección de variables que se referencia bajo un nombre en común.
- Cada una de estas variables se denominan “miembros” de la estructura. Otras denominaciones son:
  - campo
  - elemento
  - atributo

Declaración de estructuras

- La definición de una estructura se realiza fuera de cualquier función, generalmente en la parte superior del archivo.

- Para definir una estructura requerimos:
  - Un nombre
  - Una lista de miembros
    - Nombre
    - Tipo

Declaración de estructuras



Declaración de estructuras

- La declaración de una estructura no crea variables.
- Solo se define el nombre y sus miembros.
- Debe estar definida para poder ser utilizada (posición en el código).

Uso de estructuras

- Una vez que se ha declarado la estructura puede ser utilizada.
- Para utilizarla hay que definir variables del tipo “estructura”.
- Para definir estas variables se utiliza la siguiente sintaxis:

```
struct nombre_estructura nombre_variable;
```

Uso de estructuras

```
struct mi_estructura{
  int miembro1;
  char miembro2;
  double miembro3;
};
…

struct mi_estructura e1;

struct mi_estructura e2;
```

Dos variables del tipo mi_estructura

Operaciones con estructuras

- Lo realmente útil no es la estructura, sino sus miembros.
- Para acceder a los valores de los miembros de una variable de tipo estructura se utiliza el operador unario “.”.
- Cada miembro es una variable común y corriente.

Operaciones con estructuras

```
struct mi_estructura{
  int miembro1;
  char miembro2;
  double miembro3;
};
…
struct mi_estructura e1;
e1.miembro1=1024;
e1.miembro2='x';
e1.miembro3=12.8;
```

Operaciones con estructuras

```
struct mi_estructura e1;
printf("m1=%d, m2=%d, m3=%d\n",
  e1.miembro1=1024,
  e1.miembro2='x',
  e1.miembro3=12.8);
```

Ejemplo

- Sin estructuras

```
char nombreAlumno [64];
int edadAlumno;
double promedioAlumno;
```

- Con estructuras

```
struct alumno{
        char nombre[64];
        int edad;
        double promedio;
};
```

Operaciones con estructuras

- Operador de asignación
  - Copia una variable de estructura a otra (miembro por miembro)
- Operadores de comparación
  - No tiene sentido a nivel de estructuras, solo a nivel de miembros.

Estructuras y funciones

- Para pasar miembros de una estructura a una función, se utiliza el mismo esquema de las variables comunes.

```
void mostrarNota(int promedio);
int validarNota(int *promedio);
…
struct alumno a1;
if(validarNota(&a1.promedio))
        mostrarNota(a1.promedio);
```

Estructuras y funciones

- Para pasar estructuras completas como parámetros se debe especificar el tipo completo de la estructura en la definición del parámetro.

```
void mostrarAlumno(struct alumno a){
        printf("Nombre: %s, edad: %d, promedio: %d\n",
                a.nombre, a.edad, a.promedio);
}
void inicializarAlumno(struct alumno *a){
        strcpy((*a).nombre,"Pedro");
        (*a).edad=5;
        (*a).promedio=10.0;
}
…
struct alumno a1;
inicializarAlumno(&a1);
mostrarAlumno(a1);
```

Estructuras y funciones

- La notación ‘(*).’ Se puede resumir con ‘->’.
- Agrega claridad al código.
- Se denomina operador “flecha”.

```
void inicializarAlumno(struct alumno *a){
        strcpy(a->nombre,"Pedro");
        a->edad=5;
        a->promedio=10.0;
}
```

Estructuras y funciones

- Para devolver estructuras como resultado de una función, se utiliza el mismo esquema de siempre.
- El resultado se copia a la variable que lo recibe.

```
struct vector{
   double x;
   double y;
};


struct vector sumar(struct vector v1, struct vector v2){
   struct vector vres;
   vres.x = v1.x + v2.x;
   vres.y = v1.y + v2.y;

   return vres;
}
```

Estructuras y funciones

```
int main(){
  struct vector va;
  struct vector vb;
  struct vector vc;

  va.x=0.5;
  va.y=1;
  vb.x=1;
  vb.y=0.5;
  vc = sumar(va,vb);
  printf("res: %.2f,%.2f\n",vc.x,vc.y);

}
```

Estructuras anidadas

- Nada impide que los miembros de una estructura sean a su vez tipos de datos estructurados, es decir:
  - Otras estructuras
  - Arreglos
- Estas estructuras se denominan anidadas.
- Incuso pueden ser estructuras recursivas.

Estructuras anidadas

```
struct punto{
  double x;
  double y;
};

struct circunferencia{
  struct punto centro;
  double radio;
};
```

Estructuras anidadas

```
double perimetro(struct circunferencia c){
  return 2*PI*c.radio;
}

double area(struct circunferencia c){
  return PI*c.radio*c.radio;
}
```

Estructuras anidadas

```
double distancia(struct punto p1, struct punto p2){

  return sqrt( pow(p2.x+p1.x,2) + pow(p2.y+p1.y,2));

}

int intersectan(struct circunferencia c1,struct
  circunferencia c2){

  double dist = distancia(c1.centro, c2.centro);
  printf("%.2f vs %.2f\n",dist,c1.radio+c2.radio);
  return (dist < c1.radio+c2.radio);

}
```

Estructuras anidadas

```
int main(){
  struct circunferencia ca;
  struct circunferencia cb;

  ca.centro.x=0;
  ca.centro.y=0;
  ca.radio = 1;

  cb.centro.x=1.9;
  cb.centro.y=0;
  cb.radio = 1;

  printf("p:%.2f, a:%.2f, int?%s\n",perimetro(ca),
              area(ca),(intersectan(ca,cb)?"Si":"No"));

}
```

Arreglos de estructuras

- Se puede crear arreglos cuyos elementos sean variables de estructura.
- Se definen de manera similar al caso común.

```
tipo arreglo[N]
```

```
struct estructura arreglo[N];
```

Arreglos de estructuras

```
struct alumno{
  int grupo;
  int promedio;
};

int main(){
  int i, suma=0;
  struct alumno alumnos[N];
  double promedio;
  for(i=0;i<N;i++){
    printf("Ingrese grupo y promedio: ");
    scanf("%d %d",&alumnos[i].grupo,&alumnos[i].promedio);
  }
  for(i=0;i<N;i++)
    suma+=alumnos[i].promedio;
  promedio = (1.0*suma)/N;
  printf("Promedio del curso: %.1f",promedio);
}
```

Declaración de un tipo estructura

- Utilizamos typedef

```
typedef struct alumno Estudiante;
```

- Con el tipo definido podemos declarar variables, arreglos o apuntadores.

```
Estudiante a1;
Estudiante alumnos[20];
Estudiante *p_alumnos;
```