Resumen C

- Es un lenguaje imperativo, eficiente y débilmente tipado.

¿Qué quiere decir esto?

Es **imperativo** porque los programas se escriben como una secuencia de instrucciones que modifican el estado del sistema.

Es **eficiente** porque se compila directamente en código máquina y utiliza los recursos del sistema de manera óptima.

Es **débilmente tipado** porque permite conversiones implícitas entre tipos de datos con menos restricciones, lo cual puede ser tanto una ventaja como una desventaja dependiendo del contexto.

Tipos:

- `char`: carácter.
- `int`: entero.
- `float`: real de simple precisión.
- `double`: real de doble precisión.
- Booleanos usando la librería `<stdbool.h>`.

Operadores Básicos

- **Aritméticos:** `+`, `-`, `*`, `/`, `%`.
- **Lógicos:** `&&`, `||`, `!`.
- **Comparación:** `>`, `>=`, `<`, `<=`, `==`, `!=`.

Funciones en C

- **Definición y Uso:**

Estructura de una función:
c
Copiar código

```
TIPO_DEVOLUCION NOMBRE(TIPO_1 ARG_1, TIPO_2 ARG_2) {

 // cuerpo de la función

  return valor;

}
```

Asignación Múltiple en C

- **No existe la asignación múltiple directa en C.**
    - Debe usarse una o más variables auxiliares para mantener la semántica del programa.

Ejemplo con variable auxiliar:

```
int x, y, xaux;

x = pedirEntero();

y = pedirEntero();

xaux = x;

x = x + (2 * y);
```

    - `y = xaux + y;`

Librería `assert`

- **Uso de `assert`:**
    - Verifica que una condición se cumpla y aborta el programa si no se cumple.

Librería `limits.h`

- **Valores máximos y mínimos de tipos básicos en C:**
    - `CHAR_MIN`, `CHAR_MAX`, `INT_MIN`, `INT_MAX`.

Arreglos (Arrays)

- **Declaración y Uso:**

Declaración de un arreglo de 5 elementos:

```
int a[5];
```

Inicialización del segundo elemento:

```
a[1] = 10;
```

Lectura del segundo elemento:

```
int x = a[1];
```

Ejemplo de Uso de Arreglos

Inicialización de un arreglo en un bucle:

```
#define N 5

int main(void) {

    int a[N];

    int i;

    i = 0;

    while(i < N) {

        a[i] = 0;

        i = i + 1;

    }

    return 0;
```

- }

Estructuras (Structs)

- **Uso de `struct` para agrupar datos de diferentes tipos:**

Definición de una estructura `par`:

```
struct par {

    int fst;

    int snd;

};
```

    -

Declaración e inicialización:

```
struct par dupla;

dupla.fst = 3;

dupla.snd = 2;
```

-

Lectura de valores:

```
int x = dupla.fst;

int y = dupla.snd;
```

Uso de `typedef` con Structs

- **Definición de un tipo con `typedef`:**

Ejemplo:

```
typedef struct {
    int fst;
    int snd;
} par;

int main(void) {
    par dupla;
    dupla.fst = 3;
    dupla.snd = 2;
    return 0;
}
```

Definición de sinónimos para tipos:

Ejemplo:

```
typedef char letra;
```

HASKELL

Haskell es un lenguaje de programación que es **tipado** y con **tipado estático**. Esto significa que cada valor y función tienen un tipo específico que se conoce y verifica antes de ejecutar el programa, ayudando a prevenir errores y haciendo que el código sea más seguro y fácil de entender.

- **Tipado**: Cada dato en el programa tiene un tipo asociado (números, texto, listas, etc.).
- **Estático**: El tipo se comprueba antes de que el programa se ejecute.

Inferencia de tipos: Haskell puede inferir el tipo de una expresión, lo que significa que puedes definir una función sin declarar su tipo y el compilador determinará el tipo automáticamente.

Tipos básicos:

- `Int`, `Integer`: Números enteros.
- `Float`, `Double`: Números de punto flotante.
- `Bool`: Booleanos (True o False).
- `Char`, `String`: Caracteres y cadenas de texto.

Polimorfismo

~ Hay expresiones que pueden tener más de un tipo

Polimorfismo Paramétrico: Funciones que pueden operar sobre cualquier tipo.

- Ejemplo: `id :: a -> a` puede ser `Char -> Char`, `Bool -> Bool`, etc.

Polimorfismo Ad hoc: Una función que puede tener distintos comportamientos dependiendo del tipo concreto con que se use

- En haskell se logra mediante Type Classes

elem :: eq a => a -> [a] -> bool
podrá utilizarse con un tipo que instancie la clase eq.

- Una clase define requisitos que debe satisfacer un tipo
- Por ejemplo, una instancia Eq tendrá definidas las funciones de igualdad y desigualdad
- Utilizando typeclasses podemos definir funciones sobre tipos para los cuales pedimos algunas restricciones
- El comportamiento dependerá de cómo el tipo defina las funciones especificadas en la clase
- Algunas Clases: Eq,Ord,Show,Núm…..

Currificación y Aplicación Parcial

Currificación: Las funciones en Haskell tienen un único parámetro y se pueden aplicar parcialmente.

- Ejemplo: `max :: Ord a => a -> (a -> a)` es una función que toma un valor y devuelve otra función que toma otro valor.

Aplicación Parcial: Uso de una función con menos parámetros de los que requiere.

- Ejemplo: `(max 4)` crea una nueva función que compara cualquier número con 4.

Funciones de Alto Orden

Funciones de Alto Orden: Funciones que pueden tomar otras funciones como parámetros o devolver funciones.

- Ejemplo: `applyTwice :: (a -> a) -> a -> a` aplica una función dos veces a un valor.

Map y Filter

Map: Aplica una función a cada elemento de una lista.

- Ejemplo: `map succ [1, 2, 3, 4]` resulta en `[2, 3, 4, 5]`.

Filter: Filtra elementos de una lista que cumplen con un predicado.

- Ejemplo: `filter even [8, 2, 3, 6, 11]` resulta en `[8, 2, 6]`.

Tipos Algebraicos

Tipos Algebraicos sin Parámetros: Definición de un conjunto de valores posibles.

- Ejemplo: `data Dia = Lunes | Martes | Miércoles | Jueves | Viernes | Sábado | Domingo`.

Tipos Algebraicos con Parámetros: Definición de tipos más complejos con constructores que tienen parámetros.

- Ejemplo: `data Figura = Circulo (Float, Float) Float | Rectangulo (Float, Float) (Float, Float)`.

Funciones con Tipos Algebraicos:

Ejemplo para calcular el área de figuras geométricas:

```
area :: Figura -> Float

area (Circulo _ r) = 3.1416 * r * r

area (Rectangulo (x, y) (w, z)) = (w - x) * (z - y)
```

Recursión y Tipos Recursivos

Tipos Recursivos: Tipos que pueden referirse a sí mismos en su definición.

- Ejemplo: `data Lista a = Vacia | Cons a (Lista a)` define una lista de elementos de tipo `a`.

Ejemplo con Maybe:

- `data Maybe a = Nothing | Just a` se usa para manejar funciones que pueden no devolver un resultado válido en todos los casos.

RESUMEN

- Tipos algebraicos sin parámetros (enumerados)
    - data Dia = Lunes | Martes | Miércoles | Jueves | Viernes | Sábado | Domingo

- Tipos algebraicos con parámetros
    - data Figura = Circulo (Float,Float) Float | Rectangulo (Float,Float) (float, Float)
- Instancias derivadas
- Sinónimos de tipos
    - type Punto = (Float, Float)
- Definición explícita de instancias
-

~ Si no puedo usar deriving y si pattern matching