Programación Imperativa

Lenguaje C
Assert - Arreglos - Estructuras

Algoritmos y Estructuras de Datos I

Martín Ariel Domínguez

Lenguaje C

- Es un lenguaje imperativo
- Es destacado por su nivel de eficiencia
- Es débilmente tipado de medio nivel.
- Es “Case Sensitive”.
- Se adopta el estandar (ISO/IEC 9899:1990) que lo hace multiplataforma

Asignación múltiple

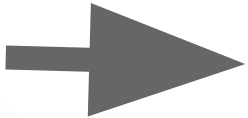
¡En lenguaje C, no existe la asignación múltiple!
Debemos tener cuidado cómo traducimos para que se respete la semántica del programa,.
Es decir, tengan estados equivalentes (el valor de las variables sean los mismos).

Formalismo

$(x \mapsto X_0,\ y \mapsto Y_0)$

$x,y := x+(2*y), y+x;$

$(x \mapsto X_0+(2*Y_0),\ y \mapsto X_0+Y_0)$



Lenguaje C

Debo usar una, o más, variables auxiliares

```
...
    int x,y,xaux;
    x = pedirEntero ();
    y = pedirEntero ();
    xaux=x;
    x=x+(2*y);
    y=xaux+y;
...
```

NOTA: Los puntos suspensivos indican más código.

Asignación múltiple

¿Qué sucede si no uso una variable auxiliar?
Y lo traduzco simplemente poniendo una línea debajo de la otra.

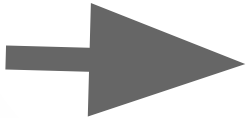
Formalismo

$(x \mapsto X_0,\ y \mapsto Y_0)$

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

$(x \mapsto X_0+(2*Y_0),\ y \mapsto X_0+Y_0)$

Traducción Incorrecta



¡Los estados son distintos!

Lenguaje C

```
...
    int x,y;
    x = pedirEntero ();
    y = pedirEntero ();
    x=x+(2*y);
    y=x+y;
...
```

$(x \mapsto X_0+(2*Y_0),\ y \mapsto X_0+(2*Y_0)+Y_0)$

Alcance de una variable

- ¿Dónde es válida un variable en un programa en C?. (Scope)
- La variable puede utilizarse dentro del bloque de código en el que fue definida.
- En el ejemplo, la variable `x` definida en la función `main`, es otra variable que la de la función `f`.

```
int f(int a){
    int x=2;
    printf("En f->x=%d\n",x);
    printf("En f->a=%d\n",a);
    return x;
 }
int main(void){
    int x=3;
    printf("En main->x=%d\n",x);
    x = f(4);
    return 0;
}
```

Lenguaje C - Introducción

- En haskell para ejecutar un programa
  - Interprestado (ghci)
  - Compilado (ghc)
- Para el lenguaje C, sólo se puede compilarse un programa para ejecutarlo utilizado el comando de consola: **`gcc`**

```
$>gcc -Wall -Wextra -std=c99 hola.c -o holaPrograma
```

Librería assert

- Esta librería implementa "assert" para determinar el cumplimiento de asunciones en el programa.
- Evalúa el cumplimiento del predicado que se le da como argumento y si no se cumple, se aborta el programa.

```
#include <assert.h>
...
assert (i != 4);
```

assert library

- En caso de ingresar datos que incumplen la aserción:

  ```
  "main: main.c:7: main: Assertion `i != 4' failed. Aborted"
  ```

- Puede usarse para comprobar que una pre-condición se satisface antes de llamar a una función
- Puede usarse para comprobar que una postcondición se satisface después de llamar a una función
- Para cualquier otra suposición que estemos seguros que se tenga que cumplir (caso contrario, el programa no tiene sentido que continúe)

Ejemplo de traducción con assert

```
|[ var   x : Num
   con   X : Num
   { X > 0  ∧  x = X }
   do  x < 10 →x := x +
1
   od
   { x >= 10 }
]|
```

```
#include <assert.h>
#include <stdio.h>
int x;
int xInput;
printf("Ingrese un entero positivo\n");
  scanf("%d",&xInput);
  x = xInput;
  assert(xInput > 0  && x == xInput);
  while ( x < 10){
    x = x + 1;
  }
  assert( x >= 10 );
```

Librería `limits.h`

- Esta librería define los valores máximos y mínimos de los tipos básicos de C
- Podemos utilizarla por ejemplo, para cuando en una derivación necesitamos utilizar $-\infty$ o $+\infty$
- Como usarla:

```
#include <limits.h>
...
 printf("The minimum value of INT = %d\n", INT_MIN);
```

Librería `limits.h`

Valores máximos y mínimos de los tipos que utilizamos:

Macro	Value	Description
CHAR_MIN	-128	Define el valor mínimo que puede tomar una variable del tipo `CHAR`
CHAR_MAX	+127	Define el valor máximo que puede tomar una variable del tipo `CHAR`
INT_MIN	-2147483648	Define el valor mínimo que puede tomar una variable del tipo `INT`
INT_MAX	+2147483647	Define el valor máximo que puede tomar una variable del tipo `INT`

Arreglos (Arrays)

- Los arreglos (arrays en inglés) permiten almacenar vectores y matrices.
- Los arreglos unidimensionales sirven para representar vectores
- Los arreglos bidimensionales para matrices de dos dimensiones.

Arreglos (Arrays)

Cuándo se utilizan:

- Es un tipo abstracto de datos que es adecuado para situaciones en las que el acceso a los datos se realice de forma aleatoria e impredecible.
- En caso contrario, cuando los elementos pueden estar ordenados y se va a utilizar acceso secuencial sería más adecuado utilizar una lista.
- Se pueden representar matrices n-dimensionales.

Arreglos: Declaración

- Para declarar un arreglo en el lenguaje "C" se utiliza la siguiente instrucción:

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

NOTA: esta instrucción declara un arreglo de 5 elementos con índices de 0 a 4 .

Arreglos: Inicialización

- Para inicializar o asignarle valor al segundo elemento del arreglo definido previamente se utiliza la siguiente instrucción:

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

Arreglos: Lectura

- Para leer el segundo elemento del arreglo definido previamente se utiliza la siguiente instrucción:

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

Arreglos: Ejemplo de uso

Formalismo Básico

```
|[ var Const N : Int, a
: array [0, N) of Int,
Var i: Int
{ True }
i := 0;
do i < N ->
a.i :=0; i := i+1;
od
{ S }
]|
```

¿Qué hace éste código?

Traducción a C

```
#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 (Struct)

- Las estructuras son utilizadas en la soluciones de algunos problemas, donde es necesario agrupar datos de diferentes tipos. También pueden ser utilizadas para manejar datos que serían muy difícil de describir en los tipos de datos primitivos.
- En el lenguaje "C" se utilizan el comando struct para agrupar diferentes valores de acuerdo a lo que se necesite.
- Para hace una analogía, las **`structs`**, se parecen a una tupla, donde a cada elemento le asigno un nombre, para luego poder referirme más fácilmente.

Estructuras: Definicón

- Supongamos que queremos definir un tipo par, para poder representar un par ordenado. Deberíamos utilizar las siguientes instrucciones.

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

Estructuras: Declaración e Inicialización

- Para declarar una variable que tenga el tipo de la estructura definida previamente y poder asignarle algún valor a los campos se procede como sigue:

```
struct par dupla;
...
dupla.fst=3;
dupla.snd=2;
```

Estructuras: Lectura

- Para leer los valores de la estructura que definimos:

```
...
x= dupla.fst;
y= dupla.snd;
```

Estructuras: Definición

- También podemos escribir el **`struct`**, asociado con **`typedef`**. De esta manera evitamos tener que usar **`struct par`** para la declaración, como sigue:

```
typedef struct {
   int fst;
   int snd;
} par;
int  main (void){
   par dupla;
   dupla.fst=3;
...
```

Sinonimo de tipos

- La sentencia **`typedef`** no solo se aplica a estructuras sino que permite definir sinónimos para cualquier tipo (igual que **`type`** en haskell)
- Se debe indicar primero el tipo original y luego el sinónimo que se va a definir:

  `typedef <tipo_viejo>` `<tipo_nuevo>;`
- Por ejemplo si se quiere definir un nuevo tipo `letra` como un sinónimo de `char`:
  `typedef char` `letra;`