Abstracción del Control de Flujo de Ejecución

Secuencias

Esp. Ing. José María Sola, profesor.

Revisión 1.0.0
2018-04-15

Tabla de contenidos

1. Introducción a Secuencias 1
2. Caso de Estudio 3
    2.1. Diagrama de Flujo 4
    2.2. Diagrama N-S 4
    2.3. Pseudocódigo 4
    2.4. C++ 4
    2.5. C 5
    2.6. Assembler 5
    2.7. C++ Autogenerado desde Diagrama N-S 6
3. Síntesis 7
    3.1. Lenguajes Formales 7
        3.1.1. Expresión Regular 7
        3.1.2. RegEx 7
        3.1.3. Gramática 7
        3.1.4. Autómata Finito 7
    3.2. Diagrama de Flujo 8
    3.3. Diagrama N-S 8
    3.4. C++ 8
4. Ejercicios Propuestos 9

1

Introducción a Secuencias

La *secuencia* es la más simple y prevalente estructura de control. Es por eso que a veces no se la identifica correctamente o se la pasa por alto. La secuencia establece el orden de los pasos de un algoritmo.

Como caso de estudio simple, escribamos un *algoritmo* que resuelve el siguiente problema:

> Salir de una habitación cuya puerta está cerrada y dejar la puerta como estaba.

Una posible solución es:

> 1. Girar el picaporte y tirar para abrir la puerta.
> 2. Pasar por la puerta.
> 3. Tirar del picaporte para cerrar la puerta.

Analicemos la solución.

Ejecutor
: En este caso somos nosotros quienes ejecutamos las acciones, la secuencia establece instrucciones para que las realice una persona, no una computadora.

Precondicones

Para que el algoritmo sea efectivo, hay varias condiciones que asumimos se cumplen. La asunción implica que la verificación no es parte de la secuencia de pasos. Ese tipo de condiciones se las conoce como *precondiciones*. Algunas precondciones son que estemos dentro de la habitación y que la puerta no esté con llave. ¿Qué otras precondiciones encuentra?

Secuencia

El orden de los pasos es fundamental, ya que el siguiente algoritmo no resolvería el problema:

1. Pasar por la puerta.
2. Tirar del picaporte para cerrar la puerta.
3. Girar el picaporte y tirar para abrir la puerta.

Para el caso de las computadoras, las intrucciones están en *memoria principal* y las ejecuta el *procesador*. El orden de la secuencia lo mantiene el registro *Instruction Pointer* (*Puntero a Instrucción*) también llamado *Program Counter* (*Contador del Programa*), que indica la dirección en memoria de la próxima instrucción a ejecutar, y se incrementa luego de que la instrucción es ejecutada. Si la instrucción es una *instrucción de salto* en vez de un icremento puede ser un decremento.

Pero eso es demasiado detalle, demasiado *bajo nivel de abstracción*. En un *alto nivel de abstracción*, la secuencia la establece el orden de escritura de las instrucciones, *de arriba hacia abajo*, y de *izquerida a derecha*, similar a algunos lenguajes naturales como el castellano o inglés.

2

Caso de Estudio

El caso de estudio es la resolución de un problema simple:

> Obtener del usuario dos números y mostrarle la suma.

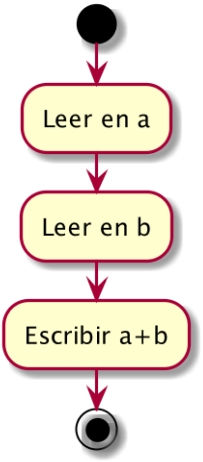
Todas las notaciones del algoritmo utilizan dos variables para almacenar los números ingresados por el usuario, por eso el *léxico* es el mismo:

$$a, b \in \mathbb{Z}$$

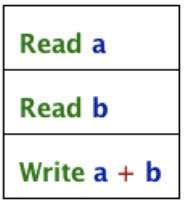
ó bien en C++

```
int a, b;
```

2.1. Diagrama de Flujo



2.2. Diagrama N-S



2.3. Pseudocódigo

```
Leer en a.
Leer en b.
Escribir a+b.
```

2.4. C++

```
/* JMS
 * 20130411
 * Adición
 */

#include <iostream>
```

```
int main(){
 int a, b;
 std::cin >> a;
 std::cin >> b;
 std::cout << a + b;
}
```

2.5. C

```
/* JMS
 * 20180414
 * Adición
 */

#include <stdio.h>

int main(void){
 int a, b;
 scanf("%d", &a);
 scanf("%d", &b);
 printf("%d\n", a+b);
}
```

2.6. Assembler

```
 subq $32, %rsp
 movq __ZNSt3__13cinE@GOTPCREL(%rip), %rdi
 leaq -4(%rbp), %rsi
 callq __ZNSt3__113basic_istreamIcNS_11char_traitsIcEEErsERi
 movq __ZNSt3__13cinE@GOTPCREL(%rip), %rdi
 leaq -8(%rbp), %rsi
 movq %rax, -16(%rbp)          ## 8-byte Spill
 callq __ZNSt3__113basic_istreamIcNS_11char_traitsIcEEErsERi
 movq __ZNSt3__14coutE@GOTPCREL(%rip), %rdi
 movl -4(%rbp), %ecx
 addl -8(%rbp), %ecx
 movl %ecx, %esi
 movq %rax, -24(%rbp)          ## 8-byte Spill
 callq __ZNSt3__113basic_ostreamIcNS_11char_traitsIcEEElsEi
 xorl %ecx, %ecx
 movq %rax, -32(%rbp)          ## 8-byte Spill
 movl %ecx, %eax
```

```
 addq $32, %rsp
 popq %rbp
 retq
```

2.7. C++ Autogenerado desde Diagrama N-S

```
#include <iostream>

int main(void)
{
 // b;
 // a;


 std::cin >>  a;
 std::cin >>  b;
 std::cout << a + b << std::endl;
}
```

3

Síntesis

A continuación se presentan el concepto de secuencia en diferentes representaciones.

3.1. Lenguajes Formales

3.1.1. Expresión Regular

$$ab$$

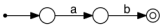
3.1.2. RegEx

```
ab
```

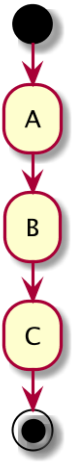
3.1.3. Gramática

$$S \rightarrow aT, T \rightarrow b$$

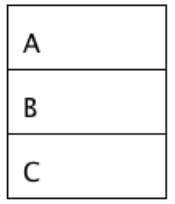
3.1.4. Autómata Finito



3.2. Diagrama de Flujo



3.3. Diagrama N-S



3.4. C++

```
A;
B;
C;
```

4

Ejercicios Propuestos

1. Resuelva el caso de estudio en *Pascal*.
2. Resuelva el caso de estudio en *Basic*.
3. Resuelva el caso de estudio en *SmallItalk*.
4. Resuelva el caso de estudio en *Python*.
5. Resuelva el caso de estudio en *JavaScript*.
6. Resuelva el caso de estudio en *Swift*.