Abstracción del Control de Flujo de Ejecución

Iteraciones

Esp. Ing. José María Sola, profesor.

Revisión 1.2.0
2017-05-07

Tabla de contenidos

1. Iteraciones 1
    1.1. Introducción 1
        1.1.1. Caso de Estudio 2
        1.1.2. Estructura del Artículo 2
    1.2. Primer Acercamiento a una Resolución 3
        1.2.1. Diagrama de Flujo 4
        1.2.2. Diagrama N-S 4
        1.2.3. Pseudocódigo 4
        1.2.4. C++ 5
        1.2.5. Assembler 5
        1.2.6. C++ Autogenerado desde Diagrama N-S 5
    1.3. Sentencia Go-To 6
        1.3.1. Diagrama de Flujo 7
        1.3.2. Diagrama N-S 7
        1.3.3. Pseudocódigo 7
        1.3.4. C++ 8
        1.3.5. Assembler 8
        1.3.6. C++ Autogenerado desde Diagrama N-S 8
    1.4. Abstracción mediante Estructuras de Iteración 9
    1.5. Sentencia Do-While 9
        1.5.1. Diagrama de Flujo 9
        1.5.2. Diagrama N-S 10
        1.5.3. Pseudocódigo 10
        1.5.4. C++ 11
        1.5.5. Assembler 11
        1.5.6. C++ Autogenerado desde Diagrama N-S 11
    1.6. Sentencia While 12
        1.6.1. Diagrama de Flujo 12
        1.6.2. Diagrama N-S 12
        1.6.3. Pseudocódigo 13
        1.6.4. C++ 13
        1.6.5. Assembler 13
        1.6.6. C++ Autogenerado desde Diagrama N-S 14
    1.7. Sentencia For 14

1.7.1. Diagrama de Flujo 14
1.7.2. Diagrama N-S 15
1.7.3. Pseudocódigo 15
1.7.4. C++ 15
1.7.5. Assembler 15
1.7.6. C++ Autogenerado desde Diagrama N-S 16
1.8. Recursividad 16
1.8.1. Diagrama N-S 16
1.8.2. Pseudocódigo 16
1.8.3. C++ 17
1.8.4. Assembler 17
1.8.5. C++ Autogenerado desde Diagrama N-S 18
1.9. Síntesis 19
1.9.1. Sentencia Do-While 19
1.9.2. Sentencia While 19
1.9.3. Sentencia For General 20
1.9.4. Sentencia For Iterador 21
1.9.5. Sentencia For Iterador de a Paso Mayor a Uno 22
1.10. Ejercicios Propuestos 23

1

Iteraciones

1.1. Introducción

La *repetición* es fundamental para la definición de algoritmos. La *programación con estructuras de control de flujo de ejecución* define tres estructuras:

- Secuencia.
- Selección.
- Iteración.

La *iteración* es un mecanismo para la repetición. Abstrae los *saltos condicionales* y *saltos incondicionales*. Hoy dos patrones básicos de iteración:

- una-ó-más-repeticiones.
- cero-ó-más-repeticiones.

Los lenguajes disponibilizan estos dos patrones en varias estructuras de control, C++ lo hace con tres estructuras:

- Sentencias Do-While.
- Sentencias While.
- Sentencias For.

Otro mecanismo para la repetición es la *recursividad*.

1.1.1. Caso de Estudio

Este texto se basa en un caso de estudio que es la resolución de un problema simple:

> Enviar por la salida estándar repetidas veces una misma frase.

Esta es una generalización del problema particular que todas las semanas[1] se enfrenta Bart Simpson[2], y al que Jason Fox dió una solución[3]. En nuestro caso en particular, la frase va a ser simplemente "Hola" y la cantidad de veces 42.

Nuestra solución se basa en un algoritmo que cuenta las veces que envía la frase y finaliza cuando la cuenta llega a 42. Vamos a repasar diferentes variantes del algoritmo, y también diferentes representaciones.

Todas las variantes del algoritmo utilizan una variable para contar, por eso el *léxico* es el mismo:

$$i \in \mathbb{N}$$

ó bien en C++

```
unsigned i;
```

Una opción también válida es utilizar Enteros ($\mathbb{Z}$) en vez de Naturales ($\mathbb{N}$), pero como el iterador o contador no va a ser menor a cero, elegimos un tipo de dato que restrinja los valores. En C++, no solo nos beneficia la restricción que da `unsigned` por sobre `int`, `unsigned` tiene un rango de valores no negativos que es el doble de `int`.

1.1.2. Estructura del Artículo

La primera parte presenta la resolución del problema con seis variantes: *Repetición de código*, *Go-To*, *Do-While*, *While*, *For*, y *Recrusividad*. Por cada variante se presenta seis representaciones.

[1] https://en.wikipedia.org/wiki/The_Simpsons_opening_sequence#Chalkboard_gag
[2] https://simpsonswiki.com/wiki/File:ChalkboardGag7F11.png
[3] http://www.gocomics.com/foxtrot/2003/10/03/

1. *Diagrama de Flujo*: Es una representación visual y de alto nivel del algoritmo donde el control del flujo de ejecución se explicita mediante flechas y saltos.
2. *Diagrama Nassi-Shneiderman (NS)*: También es una representación visual y de alto nivel del algoritmo donde el control de flujo de ejecución se abstrae de los saltos, y se vale de *estructuras de control de flujo*.
3. *Pseudocódigo*: Es una representación textual que también es de alto nivel y que abstrae el control de flujo mediante *estructuras de control de flujo de ejecución*. Las estructuras se representan textualmente con palabras que indican su inicio y, opcionalmente, su fin. El cuerpo de la estructura se denota con *indentación* (sangría). El texto es en general informal, puede estar escrito en una mezcla de lenguaje natural y matemático, y busca evitar palabras propias de un lenguaje de programación en particular.
4. *Código en Lenguaje en C++*. Esta representación también de alto nivel y textual pero que es procesable por un *compilador* para generar una representación de bajo nivel ejecutable por la máquina.
5. *Código en Lenguaje Ensamblador*: Es la representación textual de bajo nivel que tiene una relación, prácticamente, uno-a-uno con el código binario que puede ejecutar la máquina. Este lenguaje no posee *estructuras de control de flujo de ejecución*, incluye saltos condicionales y saltos incondicionales, como también uso de registros. El *ensamblador* es quien procesa esta representación y genera el código binario.
6. *Código en C++ Autogenerado*: Esta representación es generada automática a partir de los diagramas NS mediante la aplicación Structorizer[4], existen otras aplicaciones similares con la misma funcionalidad.

La segunda parte es una síntesis, y la última, ejercicios propuestos.

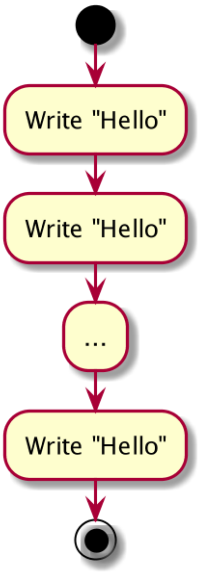
1.2. Primer Acercamiento a una Resolución

El problema es claro, y tiene una solución inicial trivial:

> El algoritmo tienen tantos pasos como veces necesitamos mostrar la frase, cada paso consta de la instrucción para mostrar la frase.

[4] http://structorizer.fisch.lu

1.2.1. Diagrama de Flujo



1.2.2. Diagrama N-S

1.2.3. Pseudocódigo

```
 1: Write "Hello".
 2: Write "Hello".
 3: Write "Hello".
...
42: Write "Hello".
```

1.2.4. C++

```
#include <iostream>

int main() {
 std::cout << "Hello\n";
 std::cout << "Hello\n";
 std::cout << "Hello\n";
 // ...
 std::cout << "Hello\n";
}
```

1.2.5. Assembler

```
 mov rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 lea rsi, [rip + L_.str]
 call __ZNSt3__1lsINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
 mov rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 lea rsi, [rip + L_.str]
 mov qword ptr [rbp - 8], rax ## 8-byte Spill
 call __ZNSt3__1lsINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
 mov rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 lea rsi, [rip + L_.str]
 mov qword ptr [rbp - 16], rax ## 8-byte Spill
 call __ZNSt3__1lsINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
 mov rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 lea rsi, [rip + L_.str]
 mov qword ptr [rbp - 24], rax ## 8-byte Spill
 call __ZNSt3__1lsINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
```

1.2.6. C++ Autogenerado desde Diagrama N-S

```
#include <iostream>

int main(void)
{
 std::cout << "Hello" << std::endl;
 std::cout << "Hello" << std::endl;
 std::cout << "Hello" << std::endl;
 // ...
 std::cout << "Hello" << std::endl;
```

```
 return 0;
}
```

La solución es *eficaz*, ya que realiza lo que se esperaba. También es *eficiente*; desde el punto de vista del *tiempo* no insume tiempo en operaciones innecesarias, es más, probablemente ésta sea la versión más eficiente en cuanto a tiempo que analicemos. También es eficiente del punto de vista del espacio, ya que no insume espacio para variables, aunque la longitud del programa está en función lineal a la cantidad de veces que necesitamos repetir el mensaje, en nuestro caso, 42 es un número relativamente bajo.

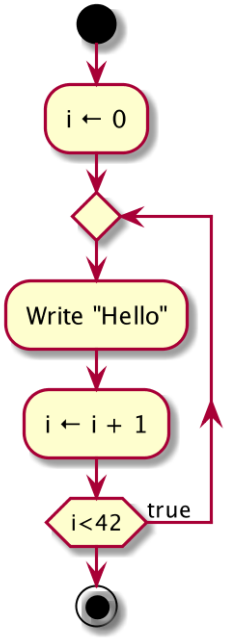
Aunque la *eficacia* y la *eficiencia* son características fundamentales, hay otros atributos de calidad[5] que también debemos considerar. ¿Cuál es el grado de *mantenibilidad*? ¿Y el de *claridad*? Podemos acordar que es bajo en ambos casos. Necesitamos otra estrategia.

1.3. Sentencia Go-To

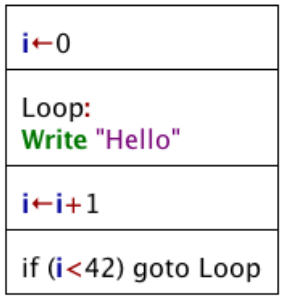
Esta estrategia se basa en identificar la instrucción que se repite y volver a esa instrucción tantas veces como sea necesario.

[5] https://en.wikipedia.org/wiki/List_of_system_quality_attributes

1.3.1. Diagrama de Flujo



1.3.2. Diagrama N-S



1.3.3. Pseudocódigo

```
1: i ← 0.
2: Write "Hello".
3: Incrementar i.
4: Si es menor a 42 entonces ir a línea 2.
```

1.3.4. C++

```
#include <iostream>

int main(void){
 unsigned i;

 i = 0;
Loop:
 std::cout << "Hello\n";
 ++i;
 if(i<42) goto Loop;
}
```

1.3.5. Assembler

```
 mov dword ptr [rbp - 8], 0
LBB0_1:                                 ## =>This Inner Loop Header:
 Depth=1
 mov rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 lea rsi, [rip + L_.str]
 call __ZNSt3__1lsINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
 mov ecx, dword ptr [rbp - 8]
 add ecx, 1
 mov dword ptr [rbp - 8], ecx
 cmp dword ptr [rbp - 8], 42
 mov qword ptr [rbp - 16], rax ## 8-byte Spill
 jae LBB0_3
## BB#2:                                ##   in Loop: Header=BB0_1
 Depth=1
 jmp LBB0_1
LBB0_3:
```

1.3.6. C++ Autogenerado desde Diagrama N-S

```
#include <iostream>

int main(void)
{
 int i;

 i = 0;
```

```
 Loop:;
 std::cout << "Hello" << std::endl;
 i = i+1;
 if (i<42) goto Loop;
}
```

1.4. Abstracción mediante Estructuras de Iteración

Se eliminan los saltos condicionales y los saltos incondicionales. Hay dos patrones:

- una-ó-más-repeticiones.
- cero-ó-más-repeticiones.

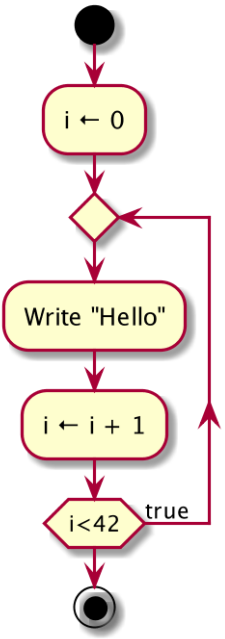
Y tres tipos de sentencia de C++ que implementan los patrones:

- Sentencias Do-While.
- Sentencias While.
- Sentencias For.

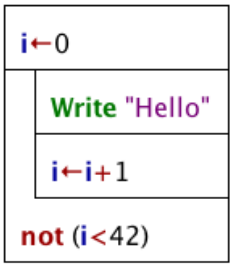
1.5. Sentencia Do-While

1.5.1. Diagrama de Flujo

El diagrama de flujo para `do-while` es idéntico al de `goto`.



1.5.2. Diagrama N-S



1.5.3. Pseudocódigo

```
i ← 0.
Hacer:
    Write "Hello".
    Incrementar i.
Mientras sea menor a 42.
```

1.5.4. C++

```
#include <iostream>

int main() {
 unsigned i;

 i=0;
 do{
  std::cout << "Hello\n";
  ++i;
 }while(i<42);
}
```

1.5.5. Assembler

```
 mov dword ptr [rbp - 8], 0
LBB0_1:                                 ## =>This Inner Loop Header:
 Depth=1
 mov rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 lea rsi, [rip + L_.str]
 call __ZNSt3__1lsINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
 mov ecx, dword ptr [rbp - 8]
 add ecx, 1
 mov dword ptr [rbp - 8], ecx
 mov qword ptr [rbp - 16], rax ## 8-byte Spill
## BB#2:                                ##   in Loop: Header=BB0_1
 Depth=1
 cmp dword ptr [rbp - 8], 42
 jb LBB0_1
```

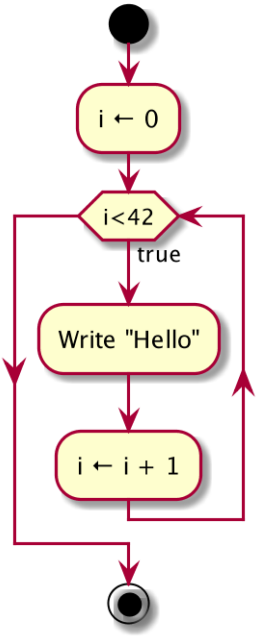
1.5.6. C++ Autogenerado desde Diagrama N-S

```
#include <iostream>

int main(void)
{
 int i;

 i = 0;
 do {
  std::cout << "Hello" << std::endl;
```

```
  i = i+1;
 } while (!(! (i<42)));
}
```

1.6. Sentencia While

1.6.1. Diagrama de Flujo



1.6.2. Diagrama N-S

1.6.3. Pseudocódigo

```
i ← 0.
Mientras sea menor a 42:
    Write "Hello".
    Incrementar i.
```

1.6.4. C++

```
#include <iostream>

int main() {
 unsigned i;

 i=0;
 while(i<42){
  std::cout << "Hello\n";
  ++i;
 }
}
```

1.6.5. Assembler

```
 mov dword ptr [rbp - 8], 0
LBB0_1:                                 ## =>This Inner Loop Header:
 Depth=1
 cmp dword ptr [rbp - 8], 42
 jae LBB0_3
## BB#2:                                ##   in Loop: Header=BB0_1
 Depth=1
 mov rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 lea rsi, [rip + L_.str]
 call __ZNSt3__1lsINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
 mov ecx, dword ptr [rbp - 8]
 add ecx, 1
 mov dword ptr [rbp - 8], ecx
 mov qword ptr [rbp - 16], rax ## 8-byte Spill
 jmp LBB0_1
LBB0_3:
```

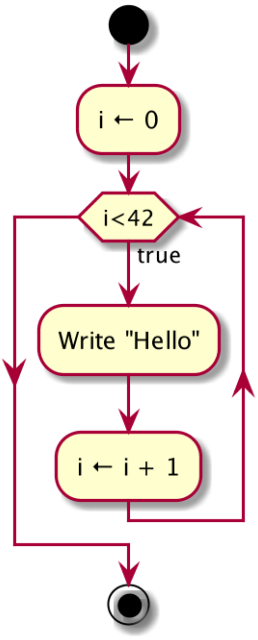
1.6.6. C++ Autogenerado desde Diagrama N-S

```
#include <iostream>

int main(void)
{
 int i;

 i = 0;
 while (i<42) {
  std::cout << "Hello" << std::endl;
  i = i+1;
 }
}
```

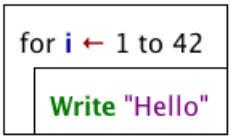
1.7. Sentencia For

1.7.1. Diagrama de Flujo

El diagrama de flujo para `for` es idéntico a `while`.



1.7.2. Diagrama N-S



1.7.3. Pseudocódigo

```
Repetir 42 veces:
    Write "Hello".
```

1.7.4. C++

```
#include <iostream>

int main() {
 for(unsigned i=0; i<42; ++i)
  std::cout << "Hello\n";
}
```

1.7.5. Assembler

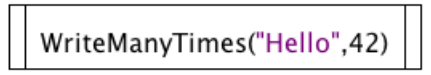
```
 mov dword ptr [rbp - 8], 0
LBB0_1:                                 ## =>This Inner Loop Header:
 Depth=1
 cmp dword ptr [rbp - 8], 42
 jae LBB0_4
## BB#2:                                ##   in Loop: Header=BB0_1
 Depth=1
 mov rdi, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 lea rsi, [rip + L_.str]
 call __ZNSt3__1lsINS_11char_traitsIcEEEERNS_13basic_ostreamIcT_EES6_PKc
 mov qword ptr [rbp - 16], rax ## 8-byte Spill
## BB#3:                                ##   in Loop: Header=BB0_1
 Depth=1
 mov eax, dword ptr [rbp - 8]
 add eax, 1
 mov dword ptr [rbp - 8], eax
 jmp LBB0_1
```

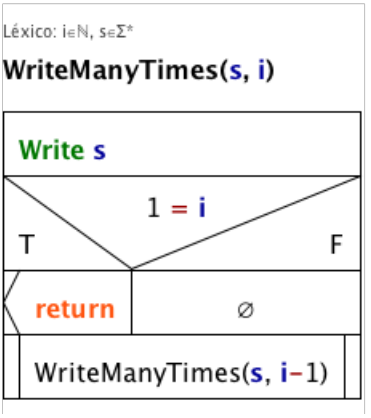
1.7.6. C++ Autogenerado desde Diagrama N-S

```
#include <iostream>

int main(void)
{
 int i;

 for (i = 1; i <= 42; i += (1)) {
  std::cout << "Hello" << std::endl;
 }
}
```

1.8. Recursividad

1.8.1. Diagrama N-S





1.8.2. Pseudocódigo

```
Progama principal:
    WriteManyTimes("Hello", 42).

WriteManyTimes(s, i):
    Si i no es cero:
```

```
        Write s.
        WriteManyTimes(s, i-1).
```

1.8.3. C++

```
#include <iostream>

void WriteManyTimes(std::string, unsigned);

int main() {
 WriteManyTimes("Hello\n", 42);
}

void WriteManyTimes(std::string s, unsigned i){
 std::cout << s;
 if( 1 == i ) return;
 WriteManyTimes(s, i-1);
}
```

1.8.4. Assembler

Esta versión es más extensa porque utiliza optimiza la *tail-call* (llamada final).

```
## BB#0:
 push rbp
Ltmp9:
 .cfi_def_cfa_offset 16
Ltmp10:
 .cfi_offset rbp, -16
 mov rbp, rsp
Ltmp11:
 .cfi_def_cfa_register rbp
 sub rsp, 80
 mov rax, qword ptr [rip + __ZNSt3__14coutE@GOTPCREL]
 mov dword ptr [rbp - 4], esi
 mov qword ptr [rbp - 56], rdi ## 8-byte Spill
 mov rdi, rax
 mov rsi, qword ptr [rbp - 56] ## 8-byte Reload
 call
 __ZNSt3__1lsIcNS_11char_traitsIcEENS_9allocatorIcEEEERNS_13basic_ostreamIT_T0_EES
 mov ecx, 1
 cmp ecx, dword ptr [rbp - 4]
 mov qword ptr [rbp - 64], rax ## 8-byte Spill
```

```
 jne LBB1_2
## BB#1:
 jmp LBB1_4
LBB1_2:
 lea rax, [rbp - 32]
 mov rdi, rax
 mov rsi, qword ptr [rbp - 56] ## 8-byte Reload
 mov qword ptr [rbp - 72], rax ## 8-byte Spill
 call
 __ZNSt3__112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEC1ERKS5_
 mov ecx, dword ptr [rbp - 4]
 dec ecx
Ltmp6:
 mov rdi, qword ptr [rbp - 72] ## 8-byte Reload
 mov esi, ecx
 call
 __Z14WriteManyTimesNSt3__112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEEj
Ltmp7:
 jmp LBB1_3
LBB1_3:
 lea rdi, [rbp - 32]
 call
 __ZNSt3__112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEED1Ev
LBB1_4:
 add rsp, 80
 pop rbp
 ret
```

1.8.5. C++ Autogenerado desde Diagrama N-S

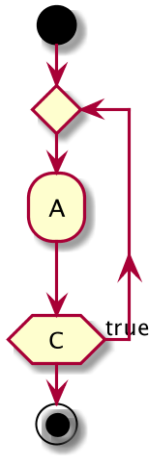
```
#include <iostream>

// Léxico: i∈ℕ, s∈Σ*
// function WriteManyTimes(s, i)
// TODO Revise the return type and declare the parameters!
void WriteManyTimes(/*type?*/ s, /*type?*/ i)
{
 std::cout << s << std::endl;
 if (1 == i) {
  return ;
 }
 WriteManyTimes(s, i-1);
}
```

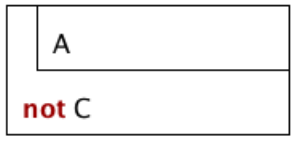
```
// program Recursion
int main(void)
{
 WriteManyTimes("Hello",42);

 return 0;
}
```

1.9. Síntesis

1.9.1. Sentencia Do-While

```
do
 A;
while(C);
```

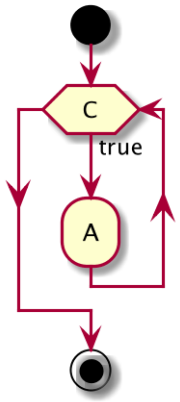


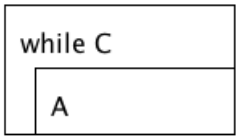


1.9.2. Sentencia While

```
while(C)
```

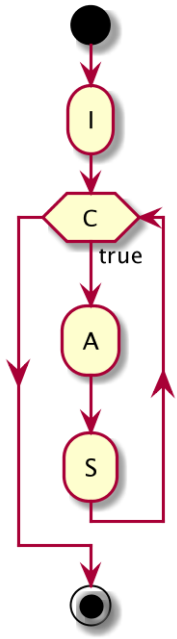
```
 A;
```

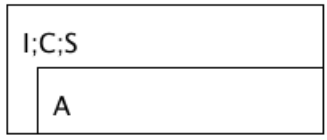




1.9.3. Sentencia For General

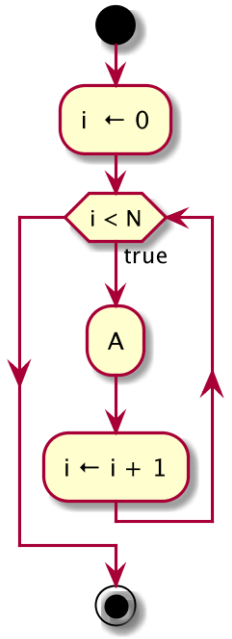
```
for(I;C;S)
 A;
```

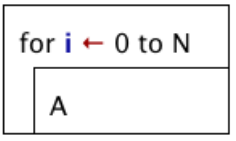




1.9.4. Sentencia For Iterador

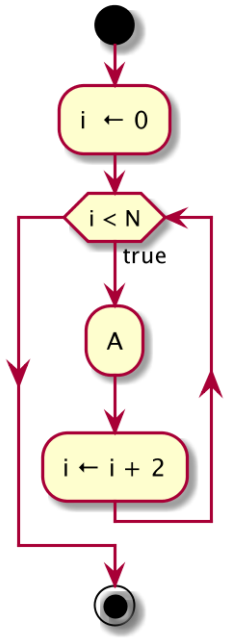
```
for(unsigned i=0; i < N; ++i)
 A;
```

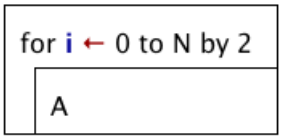




1.9.5. Sentencia For Iterador de a Paso Mayor a Uno

```
for(unsigned i=0; i < N; i+=2)
 A;
```





1.10. Ejercicios Propuestos

1. Analice el programa de Jason[6]:
   a. ¿Es factible una versión que decremente y no incremente?
   b. Escriba diferentes versiones, ya sea en C o en C++, que intenten ser más simples y concisas, que minimicen la cantidad de líneas, de operaciones, y de variables.
   c. Analice el código ensamblador generado por esas versiones y determine cuál es más eficiente.
   d. Elija la versión que considera más clara y justifique su decisión.
2. Investigue las sentencias `break` y `continue`.

[6] http://www.gocomics.com/foxtrot/2003/10/03/

3. Investigue las variantes dde la sentencia `for` en C++.
4. Investigue las estructuras iterativas de *Basic*.
5. Investigue las estructuras iterativas de *Pascal*.
6. Resuelva el caso de estudio en *Python*.
7. Resuelva el caso de estudio en *Smalltalk*.