

Estructuras de Control (y su forma en Python y en C)

Clase 5

Introducción a la Computación

Patricia Borensztein

Estructuras de Control

- En el modelo de ejecución secuencial que vimos, el procesador repite:
 - 1) Fetch de instrucción
 - 2) Decodifica y Ejecuta instrucción
 - 3) Incrementa dirección de siguiente instrucción
- Si nuestro programa quisiera calcular el 10 término de la serie de Fibonacci, debería hacer lo siguiente:

```
#include <stdio.h>
int fibo10()
{
    int term, cur=1,prev=1;
    // tercer término
    term=prev+cur;
    prev=cur;
    cur=term;

    // cuarto término
    term=prev+cur;
    prev=cur;
    cur=term;

    // quinto término
    term=prev+cur;
    prev=cur;
    cur=term;
```

```
// sexto término
    term=prev+cur;
    prev=cur;
    cur=term;

    // septimo término
    term=prev+cur;
    prev=cur;
    cur=term;

    // octavo término
    term=prev+cur;
    prev=cur;
    cur=term;

    // noveno término
    term=prev+cur;
    prev=cur;
    cur=term;
```

```
// decimo término
    term=prev+cur;
```

```
    printf ("%d\n", term);
    return 0;
}
```

Ufffffffi!!!

Estructuras de Control

- Es necesario una sentencia que pueda romper el secuenciamiento implícito de las instrucciones, pues ya no queremos que la siguiente instrucción sea la que sigue en secuencia
 - quizás queremos «ir hacia arriba» en el programa, o sea, repetir instrucciones, (repetitivos)
 - o bien quizás queremos ir hacia abajo , o sea, saltarnos instrucciones....(condicionales)
- Para eso, los lenguajes máquina o ensambladores tienen unas instrucciones especiales:
 - Saltar a una dirección determinada (Salto)
 - Saltar a una dirección determinada si se cumple una condición (Salto condicional)

Estructuras de Control

- Con estas dos simples instrucciones que ofrecen los lenguajes de bajo nivel se crearon varios tipos de sentencias de control de secuencia, o de control del flujo del programa:
- Estas son:
 - Sentencias condicionales
 - Sentencias repetitivas

Estructuras de control

- La introducción de las estructuras de control en los lenguajes de programación dio origen al estilo de programación conocido como «programación estructurada».
- El teorema del programa estructurado dice lo siguiente (Böhm-Jacopini) :
 - Todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes:
 - Secuencia
 - Instrucción condicional.
 - Iteración (bucle de instrucciones) con condición al principio.
- Este teorema demuestra que la instrucción GOTO no es estrictamente necesaria y para todo programa existe un programa equivalente que no hace uso de dicha instrucción.

Estructuras de control

- Todas las estructuras de control tienen un único punto de entrada y un único punto de salida. (si no uso GOTO en el cuerpo de la estructura de control)

```
Int función_fea()
{ int i, mi_var=0;

for (i=1;i<1000;i++)
{
    if (mi_var=27) break;
    mivar=mivar+1;
    if (i==45) return (34);
}
}
```

- Este programa tiene un punto de entrada y tres puntos de salida.

Ventajas de la Programación Estructurada

1. Los programas son más fáciles de entender, ya que pueden ser leídos de forma secuencial, sin necesidad de hacer seguimiento a saltos de línea (GOTO) dentro de los bloques de código para entender la lógica.
2. La estructura del programa es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.
3. Reducción del esfuerzo en las pruebas. El seguimiento de los fallos o errores del programa ("debugging") se facilita debido a la estructura más visible, por lo que los errores se pueden detectar y corregir más fácilmente.
4. Reducción de los costos de mantenimiento de los programas.
5. Programas más sencillos y más rápidos (ya que es más fácil su optimización).
6. Los bloques de código son auto explicativos, lo que facilita la documentación.
7. Los GOTO se reservan para construir las instrucciones básicas. Aunque no se usan de forma directa, por estar prohibida su utilización, están incluidas implícitamente en las instrucciones de selección e iteración.
8. Un programa escrito de acuerdo a estos principios no solamente tendrá una mejor estructura sino también una excelente presentación.
9. La programación estructurada ofrece estos beneficios, pero no se la debe considerar como una panacea ya que el desarrollo de programas es, principalmente, una tarea de dedicación, esfuerzo y creatividad.

Programa «Spaguetti»

- Hay un famoso artículo de Edsger Dijkstra (considerado como uno de los padres de la programación estructurada) cuyo título es: "La sentencia GOTO considerada dañina"
- Ejemplo de lo que se llamó programa «spaguetti», en Basic.

```
10 GOTO 40
20 PRINT "line number 20"
30 GOTO 60
40 PRINT "line number 40"
50 GOTO 20
60 END
```

Interface Hardware-Software

- Cualquier sentencia estructurada se implementa con un GOTO!

```
for (Bloque A; expresión booleana; Bloque B)  
{ Código interno }
```

1. Primero se evalúa el Bloque A por única vez;
2. Luego se evalúa la expresión booleana
3. Si es verdadero, se ejecuta el Código interno UNA vez, sino ir a 6.
4. Luego se evalúa el Bloque B
5. Ir a 2.
6. Siguiente instrucción

Interface Hardware-Software

```
for (Bloque A; expresión booleana; Bloque B)  
{ Código interno }
```



```
...      # Traducción del bloque A  
...  
eval: ...      # Evaluar la expresión booleana  
...      # Resultado en %eax  
cmp $0, %eax  
je finfor  
...      # Traducción del código interno  
...  
...      # Traducción del bloque B  
...  
jmp eval  
finfor:  
...      # Resto del programa
```

Tipo Booleano en C

- C define al valor entero 0 como false y cualquier valor distinto de 0 como true.

```
int main()
{
    int i,j;

    i=0;
    if (i) printf("rama then: i=%d\n",i);
        else printf("rama else: i=%d\n",i);
    i=8;
    if (i) printf("rama then: i=%d\n",i);
    j=8;
    if (i==j) printf("rama then: i=%d\n",i==j);
    if (i!=j)printf ("rama then: i=%d\n",i==j);
    else printf("rama else: i=%d\n",i!=j);
}
```

rama else: i=0
rama then: i=8
rama then: i=1
rama else: i=0

Tipo Booleano en C

- Es decir, podemos utilizar cualquier valor entero como condición.
- Cuando C evalúa una expresión de tipo condición, el resultado es 0(false) o 1(true)
- Podemos incluir el archivo : `<stdbool.h>` donde esta definido el tipo `bool` y las dos constantes `true` y `false`.

El tipo bool

```
#include <stdio.h>
#include <stdbool.h>

int main()
{
    bool i,j;

    i=true;
    if (i) printf("rama then: i=%d\n",i);
    j=false;
    if (j)printf ("rama then: j=%d\n",j);
    else printf("rama else: j=%d\n",j);
}
```

- El tipo bool es un subconjunto de los enteros y sus valores se almacenan como los enteros (32 bits)

El tipo bool en Python

- Existe el tipo bool, sus valores son True y False
- Igual que en C, el tipo bool es un subconjunto de los enteros

```
>>>c=True  
>>>type(c)  
<type 'bool'>
```

```
>>> a=True  
>>> b=False  
>>> a+b  
1  
>>> print a  
True  
>>> print b  
False  
>>> print a+b+a+b  
2  
>>> print a+a  
2  
>>> print (a+a+a+a+a+a+a)*2  
14
```

Estructuras de control en los lenguajes

- Sentencias Condicionales:
 - if-then-else
 - Select-case
- Sentencias Repetitivas:
 - While
 - Repeat
 - For

Condicionales en C

```
if (condición) {  
    sentencias  
}
```

Si dentro del if rama true o false hay una sola sentencia, las llaves se pueden obviar

```
if (condición) {  
    sentencias_si  
}  
else {  
    sentencias_no  
}
```

Y se pueden anidar:

```
if (condición) {  
    if (otra_condición) {  
        sentencias_si  
    }  
}  
else {  
    sentencias_no  
}
```

```
if (condición) {  
    sentencias_si  
}  
else if (condición2) {  
    sentencias_si2  
}  
else if (condición3) {  
    sentencias_si3  
}  
else {  
    sentencias_no  
}
```

Condicionales en C

Si hubiera ambigüedad C aplica la siguiente regla: el else pertenece al if mas cercano



```
if (condición)
|
|  if (otra_condición) {
|  |  sentencias_si
|  |
|  }
|  else { // ¿¿¿???
|  |  sentencias_no
|  |
|  }
```

```
if (condición)
|
|  if (otra_condición) {
|  |  sentencias_si
|  |
|  }
|  else { // ¿¿¿???
|  |  sentencias_no
|  |
|  }
```

Sentencia Switch-Case en C

```
1 switch (expresión) {  
2     case valor1:  
3         sentencias  
4         break;  
5     case valor2:  
6         sentencias  
7         break;  
8     ...  
9     default:  
10        sentencias  
11        break;  
12 }
```

```
1 #include <stdio.h>  
2  
3 int main(void)  
4 {  
5     int opcion;  
6  
7     printf("1)_Saluda\n");  
8     printf("2)_Despídete\n");  
9     scanf("%d", &opcion);  
10    switch (opcion) {  
11        case 1:  
12            printf("Hola\n");  
13            break;  
14        case 2:  
15            printf("Adiós\n");  
16            break;  
17        default:  
18            printf("Opción_no_válida\n");  
19            break;  
20    }  
21    return 0;  
22 }
```

Sentencia Switch-Case en C

- Comportamiento de la sentencia `break`: si no lo ponemos, no sale del `case` y sigue ejecutando las instrucciones de los otros casos....

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int opcion;
6
7     printf("1)_Saluda\n");
8     printf("2)_Despídete\n");
9     scanf("%d", &opcion);
10    switch (opcion) {
11        case 1:
12            printf("Hola\n");
13            break;
14        case 2:
15            printf("Adiós\n");
16            break;
17        default:
18            printf("Opción_no_válida\n");
19            break;
20    }
21    return 0;
22 }
```

Estructuras repetitivas o iterativas en C


- Bucle *while*
- Bucle *do-while*
- Bucle *for*

Bucle *while* (C)

```
while (condición) {  
    sentencias  
}
```

`r=r*x`

```
1  #include <stdio.h>  
2  
3  int main(void)  
4  {  
5      int x, n, i, r;  
6  
7      printf("x: "); scanf("%d", &x);  
8      printf("n: "); scanf("%d", &n);  
9      r = 1;  
10     i = 0;  
11     while (i < n) {  
12         r *= x;  
13         i++;  
14     }  
15     printf("%d**%d= %d\n", x, n, r);  
16  
17     return 0;  
18 }
```



Bucle do-while (C)

- No existe equivalente en Python
- La diferencia con el while es que en este caso se ejecuta al menos una vez el cuerpo del bucle porque la condición se evalúa al final.

```
do {  
    sentencias  
} while (condición);
```



```
1 #include <stdio.h>  
2 #include <math.h>  
3  
4 int main(void)  
5 {  
6     int a, b, i;  
7     float s;  
8  
9     /* Pedir límites inferior y superior. */  
10    do {  
11        printf("Límite inferior:"); scanf("%d", &a);  
12        if (a < 0) printf("No puede ser negativo\n");  
13    } while (a < 0);  
14  
15    do {  
16        printf("Límite superior:"); scanf("%d", &b);  
17        if (b < a) printf("No puede ser menor que %d\n", a);  
18    } while (b < a);  
19  
20    /* Calcular el sumatorio de la raíz cuadrada de i para i entre a y b. */  
21    s = 0.0;  
22    for (i = a; i <= b; i++) s += sqrt(i);  
23  
24    /* Mostrar el resultado. */  
25    printf("Sumatorio de raíces de %d a %d: %f\n", a, b, s);  
26  
27    return 0;  
28 }
```

Bucle *for* (C)

```
for (inicialización; condición; incremento) {  
    sentencias  
}
```

- Es equivalente a:

```
inicialización;  
while (condición) {  
    sentencias  
    incremento;  
}
```


Sentencias para alterar el flujo iterativo (C)

- Break:
 - Aborta la ejecución del bucle
- Continue
 - Sale de la iteración actual, pero no aborta la ejecución del bucle

Condicionales en Python

- Algunas formas de los condicionales. Python no tiene switch-case

```
if condición:  
    acción  
    acción  
    ...  
    acción
```

```
if condición:  
    | acciones  
else:  
    | otras acciones
```

```
if condición:  
    ...  
elif otra condición:  
    ...
```

```
var = "par" if (num % 2 == 0) else "impar"
```

Repetitivas: while

```
while condición:
```

```
    acción
```

```
    acción
```

```
    ...
```

```
    acción
```

Repetitivas: For in

- Las listas son secuencias o series de valores y se representan mediante corchetes.
- Los elementos de la lista pueden ser enteros, cadenas, reales, etc

```
for variable in serie de valores:  
    acción  
    acción  
    ...  
    acción
```

```
1 for nombre in ['Pepe', 'Ana', 'Juan']:  
2     print 'Hola, %s.' % nombre
```

For ..in (Python)

- Podemos utilizar la función `range()` que devuelve una lista.

```
>>> range(2, 10) ↵  
[2, 3, 4, 5, 6, 7, 8, 9]  
>>> range(0, 3) ↵  
[0, 1, 2]  
>>> range(-3, 3) ↵  
[-3, -2, -1, 0, 1, 2]
```

```
1 for i in range(1, 6):  
2     print i
```

Función range()

- La función range() tiene 1,2, o 3 argumentos
 - range (4): [0,1,2,3]
 - range(3,7): [3,4,5,6]
 - range(4,8,2):[4,6]
- Mas ejemplos:
 - range(4,1,-1):[4,3,2]

Algunas cosas generales

- A los dos lenguajes...
- Cualquier estructura de control puede anidarse dentro de cualquier otra
- Ojo con la variable de iteración del for (en C y en Python): no se le debe asignar valores dentro del cuerpo del bucle.

Euler 4

- *A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is $9009 = 91 \times 99$.*
- *Find the largest palindrome made from the product of two 3-digit numbers.*