

Introducción a la Computación (para Matemáticas)

Primer Cuatrimestre de 2018



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Programa

Un **programa** es una secuencia finita de **instrucciones**.

- Declaración de variables

TIPO NOMBRE;

- Asignación

VARIABLE = EXPRESIÓN;

- Condicional

if (CONDICIÓN) { PROG1 } else { PROG2 }

- Ciclo

while (CONDICIÓN) { PROG1 }

```
int fil = 1;
while (fil <= 5) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil + 1;
}
```

Salida:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

¿Qué podríamos cambiar
para lograr esta salida?

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

?

```

int fil = 5;
while (fil >= 1) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil - 1;
}

```

Salida:

```

1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

¿Qué podríamos cambiar para lograr esta salida?

```

1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1

```

?

```

int fil = 5;
while (fil >= 1) {
    if (fil % 2 != 0) {
        int col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
        col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
    }
    fil = fil - 1;
}

```

Salida:

```

1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1

```

¿Y para esta salida?

```

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1 2 3
1
1
1

```

?

```

int fil = 1;
while (fil <= 5) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil + 1;
}

```

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```

```

int fil = 5;
while (fil >= 1) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil - 1;
}

```

```

1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

```

int fil = 5;
while (fil >= 1) {
    if (fil % 2 != 0) {
        int col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
        col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
    }
    fil = fil - 1;
}

```

```

1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1

```

¿Cómo puedo hacer para reusar este código sin tener que copiarlo una y otra vez?

```
void imprimir_fila(int fil) {  
    int col = 1;  
    while (col <= fil) {  
        cout << col << " ";  
        col = col + 1;  
    }  
    cout << endl;  
}
```

Ejemplos:

imprimir_fila(2) imprime: **1 2**

imprimir_fila(5) imprime: **1 2 3 4 5**

```
int fil = 1;
while (fil <= 5) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil + 1;
}
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
int fil = 5;
while (fil >= 1) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil - 1;
}
```

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
int fil = 5;
while (fil >= 1) {
    if (fil % 2 != 0) {
        int col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
        col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
    }
    fil = fil - 1;
}
```

```
1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1
```



```
int fil = 1;
while (fil <= 5) {
    imprimir_fila(fil);
    fil = fil + 1;
}
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
int fil = 5;
while (fil >= 1) {
    if (fil % 2 != 0) {
        imprimir_fila(fil);
        imprimir_fila(fil);
        fil = fil - 1;
    }
}
```

```
1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1
```

```
int fil = 5;
while (fil >= 1) {
    imprimir_fila(fil);
};
}
```

fil = fil -

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
void imprimir_fila(int n) {
    int col = 1;
    while (col <= n) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
}
```

Resultado: Código modular.

- Más claro para los humanos.
- Más fácil de actualizar.

(Ej: ¿Qué pasa si ahora quiero separar los números con “,” en lugar de “ ”?)

Funciones

Una **función** es una unidad de código que **aísla** una parte de un cómputo. Es un programa dentro de un programa.

- Permite dividir un problema en **problemas más simples**.
- Permite **ordenar** conceptualmente el código para que sea más fácil de entender. Lo ideal es que realice una tarea específica.
- Permite **reutilizar** soluciones a problemas pequeños en la solución de problemas mayores.
- Una **función** se define una única vez y puede invocarse múltiples veces e incluso llamarse a sí misma (función recurrente).

Declaración de funciones

- Para declarar nuestras propias funciones debemos especificar:
 1. Tipo de retorno (eventualmente puede ser vacío, **avoid**).
 2. Nombre (obligatorio).
 3. Argumentos o parámetros (pueden ser vacíos)
- Los argumentos se especifican separados por comas, y debe tener un tipo de datos asociado.
- La declaración debe aparecer antes de ser utilizada.

Invocación de funciones

- Cuando se invoca a una función, el código \llamador debe respetar el orden y tipo de los argumentos.
- También hay que cuidar que si la función es utilizada como parte de una expresión las operaciones que se le apliquen al resultado de la función sean compatibles.
- Lo mismo cuando asignamos el valor que devuelva la función a una variable, los tipos de ambos deben ser compatibles.

Ventajas de utilizar funciones

- Solamente se escribe una vez. Esto evita errores involuntarios (pero siempre presentes) de transcripción.
- Si una función es probada y funciona bien, funcionará bien cada vez que se use (siempre y cuando el uso sea el correcto). En general, facilita la detección de errores.
- Son portables. Una misma función puede ser útil para distintos casos, distintos programas y distintos programadores
- Código más limpio. Al usar funciones reducimos las líneas de código de nuestro programa y por lo tanto se hacen mucho más fáciles de leer y validar su correctitud.
- Al dividir un programa en varios subprogramas podemos conceptualizar mejor el problema.

Este es el **tipo** del valor que devuelve la función.

Función

Estos son los **argumentos** de la función.

```
int raiz_cuadrada(int n) {  
    int i = 1;  
    while (i * i <= n) {  
        i = i + 1;  
    }  
    return (i - 1);  
}
```

Cuerpo de la función

Al llegar acá, se evalúa la expresión, se **devuelve** el valor resultante y la función **termina**.

Ahora que tengo definida la función **raiz_cuadrada**, puedo usarla en otra parte de mi código para construir nuevas expresiones.

Ejemplo:

```
int x = raiz_cuadrada(100);  
x = raiz_cuadrada(x + 6) / 2;
```

Valor de retorno

- ▶ Una función retorna un valor mediante la sentencia **return**.
- ▶ Por ejemplo, la siguiente función toma un parámetro entero y devuelve el *siguiente* valor:

```
▶  
1 int siguiente(int a) {  
2     return a+1;  
3 }
```

- ▶ Otra versión (quizás menos intuitiva):

```
▶  
1 int siguiente(int a) {  
2     int b =0;  
3     b = a+1;  
4     return b;  
5 }
```

Uso de funciones

- ▶ Volviendo al ejemplo anterior, podemos hacer un llamado (invocación) a la función **siguiente()** dentro de nuestro programa:

```
1  #include <iostream>
2  using namespace std;
3
4  int siguiente(int a) {
5      return a+1;
6  }
7
8  int main() {
9      int a = 5;
10     int b = 2 * siguiente(a); // llamado a siguiente
11
12     cout << b;
13     return 0;
14 }
```

Procedimiento

Procedimiento == **Función** que no devuelve valor alguno.

```
void imprimir_fila(int n) {  
    int col = 1;  
    while (col <= n) {  
        cout << col << " ";  
        col = col + 1;  
    }  
    cout << "\n";  
}
```

En C++, los procedimientos son de tipo **void** (“nulo”, es español).

Funciones con n-parámetros

- ▶ En caso de que haya más de un parámetro, se separan por comas:

```
1  #include <iostream>
2  using namespace std;
3
4  int suma(int a, int b) {
5      return a+b;
6  }
7
8  int main() {
9      int a = suma(2,3); // llamado con 2 argumentos
10     cout << a;
11     return 0;
12 }
```

Alcance de las variables

```
int raiz_cuadrada(int n) {  
    int i = 1;  
    while (i * i <= n) {  
        i = i + 1;  
    }  
    return i - 1;  
}  
int main() {  
    int x = 1;  
    while (x <= 5) {  
        cout << raiz_cuadrada(x) << "\n";  
        x = x + 1; } }
```

Cada ejecución de una función tiene su **propio espacio de memoria**, como si fuera un programa separado.

n, **i** son **alcanzables** dentro de `raiz_cuadrada`, pero no fuera.

Un detalle técnico

En C++, las funciones deben definirse **antes** de ser usadas.

- Antes == Más arriba en el archivo de código.
- Por eso, la función principal (**main**) suele definirse abajo de todo.

Si necesitamos usar una función antes de su definición, podemos copiar su **signatura** arriba de todo:

```
#include <iostream>
using namespace std;
int raiz_cuadrada(int n);
int main() {
    cout << raiz_cuadrada(8) << "\n";
}

int raiz_cuadrada(int n) {
    int i = 1;
    while (i * i <= n) {
        i = i + 1;
    }
    return i - 1;
}
```

Ejemplo:

```
#include <iostream>
using namespace std;

int sumaprop(int a, int b) {
    int c = a + b;
    return c;
}

int main() {
    int s1=0; int s2=0;
    cout << "Ingrese un número entero" << endl;
    cin >> s1;
    cout << "Ingrese otro número entero" << endl;
    cin >> s2;
    cout << "La suma de sus dos números ingresados es:" <<
        sumaprop(s1,s2) << endl;
    return 0;
}
```

Ejemplo (cont.)

```
#include <iostream>
using namespace std;
int sumaprop(int a, int b) {
    int c = a + b;
    return c;
}
int suma3(int a1, int a2, int a3) {
    int aux = sumaprop(a1, a2);
    int r = sumaprop(aux, a3);
    return r;
}
int main() {
    int s1=0; int s2=0; int s3=0;
    cout << "Ingrese un número entero" << endl;
    cin >> s1;
    cout << "Ingrese otro número entero" << endl;
    cin >> s2;
    cout << "Ingrese otro número entero" << endl;
    cin >> s3;
    cout << "La suma de los números ingresados es:" << suma3(s1, s2, s3) << endl;
    return 0;
}
```

Ejemplo (cont.)

¿Cómo hago para generalizar la función a N valores ingresados?

¿Cómo escribo una función que calcule el promedio de los valores ingresados usando la función anterior?

Tipo vector

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    //vector sin inicializar ni indicar tamaño
    vector<double> arreglo_1;

    //vector con tamaño 5 y componentes inicializadas
    vector<double> arreglo_2(5,3.1415);

    //mostrar las componentes con un ciclo
    for(unsigned int i=0;i<arreglo_2.size();i++)
    { //con el mtodo .size() se obtiene el tamaño del vector
        cout<<arreglo_2[i]<<endl;
    }
    cout<<endl<<endl<<endl;
    return 0;
}
```


Ejercicio

Conjetura de Collatz

Conocida también como **conjetura $3n+1$** . Sea la siguiente operación, aplicable a cualquier número entero positivo:

- Si el número es par, se divide entre 2.
- Si el número es impar, se multiplica por 3 y se suma 1.

La conjetura dice que siempre alcanzaremos el 1.

Ejemplos:

$n = 6$, uno llega a la siguiente sucesión: 6, 3, 10, 5, 16, 8, 4, 2, 1.

$n = 11$, la sucesión tarda un poco más en alcanzar el 1:
11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Repaso de la clase de hoy

- Modularidad del código: funciones y procedimientos.
- Alcance (scope) de variables.

Próximos temas

- Especificación de problemas.
- Correctitud de programas.