Introducción a la Programación

Usando C, C++, JavaScript, Lilly, ...

Expressions

- Assume X:6 and Y:7
- Compute the following expressions:

  X

  6 + 1

  Y + 1

  X * Y + 1

  (X * Y) + 1

  X * (Y + 1)

C/C++: Operadores binarios infijos para comparar números

- Estos operadores funcionan para "números enteros" (int) y "números reales" (float, double)
- Hasta ahora solo hemos visto "números enteros" en C y mencionado a ciertos "números reales"
  - int: números enteros, con signo, *dentro de un rango limitado*
  - float: aproximación a "números reales", usando precisión simple con punto flotante (típicamente 32 bits)
  - double: aproximación a "números reales", usando doble precisión con punto flotante (típicamente 64 bits)
- Si el tipo de los operandos es mixto, se aplican reglas de conversión implícitas (promoción) para hacer que sean iguales
  - int ➔ float
  - int ➔ double
  - float ➔ double
- Independientemente de lo anterior, muchos compiladores fuerzan el uso de doble precisión *al evaluar expresiones* con puntos flotantes
  - Lógicamente, la representación float es respetada *al almacenar los resultados* de la evaluación en variables de tipo float

Operador	Significado	Ejemplo
`<`	menor que	`x < y`
`<=`	menor o igual a	`x <= y`
`==`	igual a	`x == y`
`!=`	distinto de	`x != y`
`>=`	mayor o igual a	`x >= y`
`>`	mayor que	`x > y`

C Operator Precedence

[C Operator Precedence - cppreference.com](https://en.cppreference.com/w/c/language/operator_precedence)

https://en.cppreference.com/w/c/language/operator_precedence

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	( )	Function call	
	[ ]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*) { *list* }	Compound literal(C99)	
2	++ --	Prefix increment and decrement[note 1]	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of[note 2]	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	? :	Ternary conditional[note 3]	Right-to-left
14[note 4]	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

C++ Operator Precedence

[C++ Operator Precedence - cppreference.com](https://en.cppreference.com/w/cpp/language/operator_precedence)

https://en.cppreference.com/w/cpp/language/operator_precedence

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right →
2	`a++ a--`	Suffix/postfix increment and decrement	
	`type() type{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`. ->`	Member access	
3	`++a --a`	Prefix increment and decrement	Right-to-left ←
	`+a -a`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(type)`	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`co_await`	await-expression (C++20)	
	`new new[]`	Dynamic memory allocation	
	`delete delete[]`	Dynamic memory deallocation	
4	`.* ->*`	Pointer-to-member	Left-to-right →
5	`a*b a/b a%b`	Multiplication, division, and remainder	
6	`a+b a-b`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`<=>`	Three-way comparison operator (since C++20)	
9	`< <= > >=`	For relational operators < and ≤ and > and ≥ respectively	
10	`== !=`	For equality operators = and ≠ respectively	
11	`a&b`	Bitwise AND	
12	`^`	Bitwise XOR (exclusive or)	
13	`|`	Bitwise OR (inclusive or)	
14	`&&`	Logical AND	
15	`||`	Logical OR	
16	`a?b:c`	Ternary conditional[note 2]	Right-to-left ←
	`throw`	throw operator	
	`co_yield`	yield-expression (C++20)	
	`=`	Direct assignment (provided by default for C++ classes)	
	`+= -=`	Compound assignment by sum and difference	
	`*= /= %=`	Compound assignment by product, quotient, and remainder	
	`<<= >>=`	Compound assignment by bitwise left shift and right shift	
	`&= ^= |=`	Compound assignment by bitwise AND, XOR, and OR	
17	`,`	Comma	Left-to-right →

C/C++: Especificadores de formatos: para uso con printf, fprintf, sprintf, etc.

Tipo	*Tipo*	Formateador de Salida
Booleano	bool	n.a. (definido por el programador)
Entero	int	%d
Punto flotante	float y double	%f %lf (con consideraciones adicionales)
String	char *	%s

Assignment

- Syntax in C/C++:

  <lhs> = <rhs>

  where <lhs> and <rhs> must be syntactically valid expressions

- Semantic requirements:

  <lhs> must be assignable (i.e., denote an l-value)

  <lhs> and <rhs> must have the same type or <rhs> must be implicitly convertible to the type of <lhs>

- Operational Semantic:

  <rhs> is evaluated and the result is stored in the variable denoted by <lhs>

  Spanish: se evalua <rhs>, el resultado pasa a ser el contenido de la variable denotada por <lhs>

  After the assignment we have CVALUE(LVALUE(<lhs>)) == RVALUE(<rhs>) (*)

  (*) Warning: in the presence of side effects the above must be refined with temporal indices

Para la Tarea
Estado de la memoria después de ejecutar cada paso

Este es el ejemplo anterior, ya resuelto, para mostrar como pueden reportar el resultado en texto
Simple texto (.txt) en blanco y negro: ¡no desperdicien tiempo poniendo colores!

```
1. int x = 6;          // x:6

2. int y = 7;          // x:6, y:7

3. int z = x * y;      // x:6, y:7, z:42

4. y = y + 1;          // x:6, y:8, z:42

5. x = 2 * z;          // x:84, y:8, z:42
```

Tarea 0:
Reportar el estado de la memoria después de ejecutar cada paso

Caso B

```
1. int x = 6;          // x: 6
2. int y = 7;          // x: 6, y: 7
3. int z = x * y;      // x: 6, y: 7, z: 42
4. x = x * x + y;      // x: 43, y: 7, z: 42
5. y = x * (x - z);    // x: 43, y: 43, z: 42
6. z = z + 1;          // x: 43, y: 43, z: 43
```

Tarea 1:
Intercambiar el valor de dos variables

Utilizando solo definiciones de variables y asignaciones, agregue instrucciones a la siguiente secuencia para intercambiar los valores de *x* y *y*. Claro está que su fragmento de código debe ser general, es decir que debe funcionar independientemente de los valores iniciales de *x* y *y*.

```
1. int x = 6;
2. int y = 7;
…. …
```

C/C++: Estructuras para Controlar el Flujo Secuencial

- Condicionales:
  - if
  - switch
- Lazos:
  - while
  - do
  - for

C/C++: Instrucción if – if statement

- Sintaxis 1

```
if (<exp>) {
        ... lista de instrucciones a ejecutar si la expresión <exp> se cumple ...
} else {
        ... lista de instrucciones a ejecutar si la expresión <exp> no se cumple ...
}
```

- Sintaxis 2

```
if (<exp>) {
        ... lista de instrucciones a ejecutar si la expresión <exp> se cumple ...
}
```

C/C++: Instrucción while – while statement

- Sintaxis

```
while (<exp>) {
        <body>
}
```

- Semántica

Mientras <exp> se cumple (evalúa a true) ejecutar el cuerpo <body> del while, es decir …

… si <exp> ➔ true, ejecutar <body> y repetir el while, si no, continuar con la instrucción que sigue

Tarea 2:
Computar con Lazos

Como seguramente apreciaron, los casos A y C de la tarea 0 tienen secuencias repetitivas: por lo tanto se prestan a ser mejorados (generalizados y más cortos a la vez) con el uso de lazos.

Completen los fragmentos de código siguientes (simplemente escríbanlos en un archivo de texto) para lograr tal objetivo. El código debe imprimir la secuencia de valores que nos interesa: z para el caso A y s para el caso C, por lo tanto deben incluir una instrucción fprintf en el cuerpo. Hay varias opciones para la condición de ejecución del lazo: por ahora simplemente asegúrense de que la ejecución termina después de imprimir un numero razonable de valores. Bonus: pueden poner su código dentro de la función main y ejecutar el programa de veras, si completaron su ambiente de desarrollo.

Caso A

```
int x = 1;
int z = 0;
while (<condición>) {
   <cuerpo>
}
```

Caso C

```
int p = 0;
int u = 1;
int s = u + p;
while (<condición>) {
   <cuerpo>
}
```

Contando de 1 a 10: uso de variables y lazos

```c
#include <stdio.h>

int main()
{
  int i = 1;
  while (i <= 10) {
    fprintf(stdout, "%d\n", i);
    i = i + 1;
  }
  return 0;
}
```

Pasando argumentos al programa

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
  fprintf(stdout, "argc: %d\n\n", argc);
  int i = 0;
  while (i < argc) {
    fprintf(stdout, "argv[%d]: %s\n", i, argv[i]);
    i = i + 1;
  }
  return 0;
}
```

Contando de 1 a N, uso de argumentos

```c
// Advertencia: este programa deliberadamente tiene errores y omisiones!

#include <stdio.h>

int main(int argc, const char *argv[])
{
  // obtener y chequear el valor del argumento
  int N = argv[1]; // <<< el compilador debe reportar un error aqui

  // contar de 1 a N
  int i = 1;
  while (i <= N) {
    fprintf(stdout, "%d\n", i);
    i = i + 1;
  }

  return 0;
}
```

Contando de 1 a N, uso de argumentos con chequeo

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, const char *argv[])
{
  // si el número de argumentos es distinto de 2, terminar con un código de error
  // recuerden que el número de argumentos incluye el nombre del ejecutable, almacenado en argv[0]
  if (argc != 2) {
    fprintf(stderr, "uso: %s <int N>\n", argv[0]);
    return 1;
  }

  // obtener y chequear el valor del argumento
  // notar el uso de la función atoi para convertir el string a entero
  int N = atoi(argv[1]);
  // si N es menor o igual a 0, terminar con un código de error
  if (N <= 0) {
    fprintf(stderr, "N debe ser mayor que cero, pero N == %d\n", N);
    return 1;
  }

  // por fin: contar de 1 a N!
  int i = 1;
  while (i <= N) {
    fprintf(stdout, "%d\n", i);
    i = i + 1;
  }
  return 0;
}
```

Ejercicio para la casa: hola con un argumento opcional
Completar este programa

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
  if (argc == 1) {
    // Simplemente imprimir "Hola!"
  } else if (argc == 2) {
    // Imprimir "Hola <nombre>!", donde <nombre> es pasado en la línea de comando
    // Por ejemplo: "Hola Francisco!"
  } else {
    fprintf(stderr, "uso: %s [<nombre>]\n", argv[0]);
    return 1;
  }
  return 0;
}
```

Éxito: leer, investigar, estudiar, practicar (programar), experimentar