Table of Contents

1. Introducción al Lenguaje
2. Estructura de un programa
3. Variables, constantes y tipos de datos
    - Definiendo variables
    - Nombres de variables
    - Constantes
    - Scope
4. Operadores
5. Basic input and output

Introducción a C++

Historia

- Creado en 1979 como una extensión de C por:

Introducción a C++

Historia

- Creado en 1979 como una extensión de C por:

Figure: Bjarne Stroustrup, creador de C++.

Introducción a C++

Historia

- Creado en 1979 como una extensión de C por:

Figure: Bjarne Stroustrup, creador de C++.

- Hecho para ser rápido *y* escalable, ofrenciendo constructores de alto y bajo nivel.

Introducción a C++

Historia

- Creado en 1979 como una extensión de C por:

Figure: Bjarne Stroustrup, creador de C++.

- Hecho para ser rápido *y* escalable, ofrenciendo constructores de alto y bajo nivel.
- Mejora contínua a lo largo de los años, las últimas fueron en Sep. 2011 (conocido como C++11).

Hola Mundo!

El comienzo

```
// My first C++ program

#include <iostream>

int main()
{
  std::cout << "Hello world!";
  return 0;
}
```

Output: `Hello World!`

Hola Mundo

```
// My first C++ program
```

Hola Mundo

```
// My first C++ program

#include <iostrea
m>
```

Hola Mundo

```
// My first C++ program

#include <iostrea
m>

int main()
```

Hola Mundo

```
// My first C++ program

#include <iostrea
m>

int main()
{



}
```

Hola Mundo

```
// My first C++ program

#include <iostrea
m>

int main()
{
       cout << "Hello World!";

}
```

Hola Mundo

```
// My first C++ program

#include <iostrea
m>

int main()
{
  std::cout << "Hello World!";

}
```

Hola Mundo

```
// My first C++ program

#include <iostrea
m>

int main()
{
  std::cout << "Hello World!";
  return 0;
}
```

Generando un ejecutable

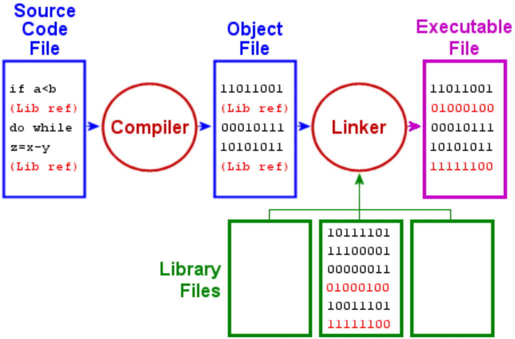


Figure: Compilando C++ [1].

[1] https://stackoverflow.com/questions/23615282/object-files-vs-library-files-and-why

Espacios en Blanco

Definition

whitespace spaces, tabs, and (sometimes) new lines.

Espacios en Blanco

Definition

whitespace spaces, tabs, and (sometimes) new lines.

Completely equivalent as far as compiler is concerned:

```
#include <iostream>
int main(){std::cout<<"Hello world!";return 0;}
```

```
#include <iostream>
int main()
{
std::cout
<<
"Hello world!"
;
return
0
;
}
```

Mayúsculas

C++ es sensible a las mayúsculas! Esto quiere decir que:

```
int main()
```

is different from

```
INT MAIN ()
```

which is different from

```
int Main()
```

and only the first version is correct.

Comentarios

Hay dos tipos de comentarios:

```
// This is a line comment.
// It ends at the end of the line.
```

```
/* This is a C-style comment.
   It ends when the closing star-slash is reached. */
```

Use comments liberally - they are enormously useful!

Do

Avoid stating the obvious. A good comment will not say *what* is happening but rather *why*.

Definiendo una variable

Definition

variable Una entidad que es posible almacenar y efectuar operaciones sobre ellas.

Ejemplos:

```
int anInteger;
double aDouble;
unsigned short i;
float x, y, z;
```

Format: `variable_type variable_name, variable_name2;`

Nombres de variables

A variable name is an example of an identifier.

Definition

identifier Un identificador es una secuencia de caracteres que sirve para nombrar variables

[2] Ver `http://en.cppreference.com/w/cpp/keyword`.

Nombres de variables

A variable name is an example of an identifier.

Definition

identifier Un identificador es una secuencia de caracteres que sirve para nombrar variables

Los identificadores pueden ser cualquier sequencia de caracteres, dígitos o guión bajo pero no deben

- comenzar con un dígito,
- ser una palabra reservada [2].

[2] Ver `http://en.cppreference.com/w/cpp/keyword`.

Nombres de variables

A variable name is an example of an identifier.

Definition

identifier Un identificador es una secuencia de caracteres que sirve para nombrar variables

Los identificadores pueden ser cualquier sequencia de caracteres, dígitos o guión bajo pero no deben

- comenzar con un dígito,
- ser una palabra reservada [2].

Definition

palabra reservada una palabra que tiene un significado especial en C++.

[2] Ver `http://en.cppreference.com/w/cpp/keyword`.

Nombres de variables

Do

- Nombrar variables con significado, aunque signifique aumentar el código!
  Malo: `data, dRange, a, ccn, value, one`
  Bueno: `daysOfWeek, sumSq, isEnabled, unitCell`

Nombres de variables

Do

- Nombrar variables con significado, aunque signifique aumentar el código!
  Malo: `data, dRange, a, ccn, value, one`
  Bueno: `daysOfWeek, sumSq, isEnabled, unitCell`

- Por ejemplo:

```
double rootMeanSquare; // .. go on to
    calculate rms
```

  El nombre de la variable hace obvio que se usa para almacenar el valor cuadrático medio

Nombres de variables

Do

- Nombrar variables con significado, aunque signifique aumentar el código!
  Malo: `data, dRange, a, ccn, value, one`
  Bueno: `daysOfWeek, sumSq, isEnabled, unitCell`

- Por ejemplo:

```
double rootMeanSquare; // .. go on to
    calculate rms
```

  El nombre de la variable hace obvio que se usa para almacenar el valor cuadrático medio

Don't

Evitar usar abreviaciones, a menos que sean muy comunes.

Tipos de Variables

Fundamental data types:

Type	Size	Values
`bool`	1 byte	true or false
`char`	1 byte	256 character values
`unsigned short int`	2 bytes	0 to 65,353
`short int`	2 bytes	-32,768 to 32,767
`unsigned int`	4 bytes	0 to 4,294,967,295
`int`	4 bytes	-2,147,483,648 to 2,147,483,647
`unsigned long int`	8 bytes	0 to 18,446,744,073,709,551,615
`long int`	8 bytes	-9,223,372,036,854,775,807 to 9,223,372,036,854,775,807
`float`	4 bytes	1.2e-38 to 3.4e38
`double`	8 bytes	2.2e-308 to 1.8e308

Usando variables

```cpp
#include <iostream>

int main()
{
  double G = 6.6738e-11;
  double massOfEarth = 5.9722e24;
  double massOfMoon = 7.3477e22;
  double r = 384400e3;
  double force;

  force = G * massOfEarth * massOfMoon / (r * r);

  std::cout << "Force between Earth and Moon is: "
            << force << "\n";

  return 0;
}
```

Constantes

```cpp
#include <iostream>

int main()
{
  const double G = 6.6738e-11;
  const double massOfEarth = 5.9722e24;
  const double massOfMoon = 7.3477e22;
  const double r = 384400e3;
  double force;

  force = G * massOfEarth * massOfMoon / (r * r);

  std::cout << "Force between Earth and Moon is: " << force << "\n";

  G = 7e-11; // WON’T COMPILE. WHAT KIND OF A UNIVERSE WOULD WE BE LIVING IN
             // IF G COULD VARY??

  return 0;
}
```

Constantes

```
#include <iostream>

int main()
{
  const double G = 6.6738e-11;
  const double massOfEarth = 5.9722e24;
  const double massOfMoon = 7.3477e22;
  const double r = 384400e3;
  double force;

  force = G * massOfEarth * massOfMoon / (r * r);

  std::cout << "Force between Earth and Moon is: " << force << "\n";

  G = 7e-11; // WON'T COMPILE. WHAT KIND OF A UNIVERSE WOULD WE BE LIVING IN
             // IF G COULD VARY??

  return 0;
}
```

Do

Usar la palabra reservada `const` para asegurar que una variable no cambia. En C++ es distinto a `mutable`

Constantes

```
// http://en.cppreference.com/w/cpp/language/cv

#include <iostream>

int main()
{
    int n1 = 0;           // non-const object
    const int n2 = 0;     // const object
    int const n3 = 0;     // const object (same as n2)
    volatile int n4 = 0;  // volatile object
    const struct
    {
        int n1;
        mutable int n2;
    } x = {0, 0};      // const object with mutable member

    n1 = 1; // ok, modifiable object
//  n2 = 2; // error: non-modifiable object
    n4 = 3; // ok, treated as a side-effect
//  x.n1 = 4; // error: member of a const object is const
    x.n2 = 4; // ok, mutable member of a const object isn't const

    const int& r1 = n1; // reference to const bound to non-const object
//  r1 = 2; // error: attempt to modify through reference to const
    const_cast<int&>(r1) = 2; // ok, modifies non-const object n1

    const int& r2 = n2; // reference to const bound to const object
//  r2 = 2; // error: attempt to modify through reference to const
//  const_cast<int&>(r2) = 2; // undefined behavior: attempt to modify const object n2
}
```

Alcance

Definition

alcance define la región del programa donde es posible usar la variable.

El alcance de una variable se define por el bloque, formado por los paréntesis {}, dentro del cual se declara.

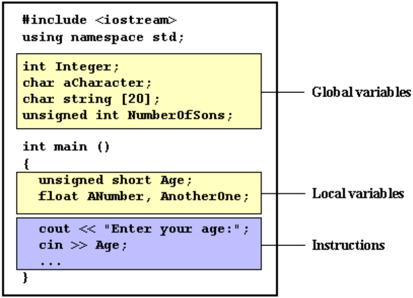
[3] Source: `http://www.cplusplus.com/doc/tutorial/variables/`.

Alcance

Definition

alcance define la región del programa donde es posible usar la variable.

El alcance de una variable se define por el bloque, formado por los paréntesis {}, dentro del cual se declara.



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

int Integer;
char aCharacter;
char string [20];
unsigned int NumberOfSons;

int main ()
{
  unsigned short Age;
  float ANumber, AnotherOne;

  cout << "Enter your age:";
  cin >> Age;
  ...
}
```

Figure: Alcance de variables[3].

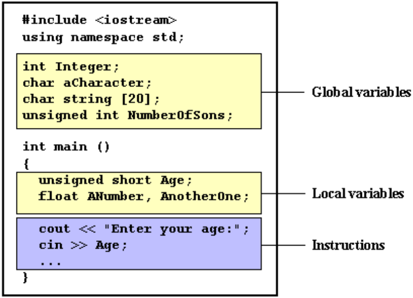
[3] Source: http://www.cplusplus.com/doc/tutorial/variables/.

Alcance

Definition

alcance define la región del programa donde es posible usar la variable.

El alcance de una variable se define por el bloque, formado por los paréntesis {}, dentro del cual se declara.



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

int Integer;
char aCharacter;
char string [20];
unsigned int NumberOfSons;

int main ()
{
  unsigned short Age;
  float ANumber, AnotherOne;

  cout << "Enter your age:";
  cin >> Age;
  ...
}
```

Figure: Alcance de variables[3].

Don't

Evitar declarar variables con alcance global. Las constantes globales están OK.

[3] Source: http://www.cplusplus.com/doc/tutorial/variables/.

Operadores Simples

Asignación: =

```
a = 5; a = b; a = b = c;
```

Operadores Simples

Asignación: =

```
a = 5; a = b; a = b = c;
```

Operadores Aritmeticos : +, −, ∗, /, %

Todos los obvios, excepto el operador *modulo* (%). Entrega el resto de la división de un número por otro.

```
a = 30 % 10;
```

Operadores Simples

Asignación: =

```
a = 5; a = b; a = b = c;
```

Operadores Aritmeticos : +, −, ∗, /, %

Todos los obvios, excepto el operador *modulo* (%). Entrega el resto de la división de un número por otro.

```
a = 30 % 10;
```

Asignación Compuesta: +=, −=, ∗=, /=, %=

Ejecuta la operación sobre el valor actual y después almacena el nuevo valor, por ejemplo:

```
a += 5; a *= b;
```

División de Enteros

Warning!

In C++ integer arithmetic truncates (effectively rounds down):

```
int dividend = 20, divisor = 7;
int someInteger = dividend / divisor; // = 2
```

Storing the result in a double doesn't help as the arithmetic has already been done:

```
double someDouble = dividend / divisor; // = 2
```

What gives?

División de Enteros

Warning!

In C++ integer arithmetic truncates (effectively rounds down):

```
int dividend = 20, divisor = 7;
int someInteger = dividend / divisor; // = 2
```

Storing the result in a double doesn't help as the arithmetic has already been done:

```
double someDouble = dividend / divisor; // = 2
```

What gives? Well you have two options:

1.
```
someDouble = static_cast<double>(dividend) /
    static_cast<double>(divisor); // = 2.85714
```

División de Enteros

Warning!

In C++ integer arithmetic truncates (effectively rounds down):

```
int dividend = 20, divisor = 7;
int someInteger = dividend / divisor; // = 2
```

Storing the result in a double doesn't help as the arithmetic has already been done:

```
double someDouble = dividend / divisor; // = 2
```

What gives? Well you have two options:

1.
   ```
   someDouble = static_cast<double>(dividend) /
     static_cast<double>(divisor); // = 2.85714
   ```

2.
   ```
   someInteger = (dividend + (dividend % divisor)) /
     divisor; // = 3
   ```

Operadores Unarios: `++`, `--`

```
a++;
```

Es equivalente a:

```
a += 1;
```

Operadores Unarios: `++, --`

```
a++;
```

Es equivalente a:

```
a += 1;
```

Operadores Relacionales: `==, !=, >, <, >=, <=`

```
bool areNotEqual = (a != b);
```

Operadores Unarios: `++`, `--`

```
a++;
```

Es equivalente a:

```
a += 1;
```

Operadores Relacionales: `==`, `!=`, `>`, `<`, `>=`, `<=`

```
bool areNotEqual = (a != b);
```

Don't

Mix up = and == this will cause endless headaches! Consider:

```
a = 5; b = 6; areEqual = (a = b);
```

This is a problem because C++ considers any number other than 0 be true!

Operadores Lógicos: !, &&, ||

NOT

a	!a
true	false
false	true

AND

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

OR

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Operadores Lógicos: !, &&, ||

NOT

a	!a
true	false
false	true

AND

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

OR

<table>
<tr><th>a</th><th>b</th><th>a || b</th></tr>
<tr><td>true</td><td>true</td><td>true</td></tr>
<tr><td>true</td><td>false</td><td>true</td></tr>
<tr><td>false</td><td>true</td><td>true</td></tr>
<tr><td>false</td><td>false</td><td>false</td></tr>
</table>

Do

Keep it simple: don't try and do too much in a single line. While this:

```
    result = (i < 10) && (++i < n);
```

is a valid expression, deciphering what it does is a lot of work. Instead use:

```
    result = i < 10;
    ++i;
    result = result && i < n;
```

Precedencia de Operadores

Precedence	Op.	Associativity
1	`++ -- !`	Right
2	`* / %`	Left
3	`+ -`	
4	`< <= > >=`	
5	`== !=`	
6	`&&`	
7	`||`	
8	`=`	Right

Operator precedence tells you the order that an expression will be evaluated in. Some are obvious but consider:

```
a = 21 + 7 % 2;
```

which could be interpreted as

```
a = 21 + (7 % 2); // this
a = (21 + 7) % 2; // or this.
```

In fact the first version is correct.

Precedencia de Operadores

Precedence	Op.	Associativity
1	++ -- !	Right
2	* / %	Left
3	+ -	
4	< <= > >=	
5	== !=	
6	&&	
7	||	
8	=	Right

Operator precedence tells you the order that an expression will be evaluated in. Some are obvious but consider:

```
  a = 21 + 7 % 2;
```

which could be interpreted as

```
  a = 21 + (7 % 2); // this
  a = (21 + 7) % 2; // or this.
```

In fact the first version is correct.

Do

Use parentheses to make an expressions more clear even if they are not necessary.

Precedencia de Operadores

Precedence	Op.	Associativity
1	++ -- !	Right
2	* / %	Left
3	+ -	
4	< <= > >=	
5	== !=	
6	&&	
7	||	
8	=	Right

Operators that have the same precedence are evaluated according to their *associativity* e.g.:

```
a = b = c;    //
    evaluates as
a = (b = c);
```

because = is right associative.

[4] http://en.cppreference.com/w/cpp/language/operator_precedence

Precedencia de Operadores

Precedence	Op.	Associativity
1	++ -- !	Right
2	* / %	Left
3	+ -	
4	< <= > >=	
5	== !=	
6	&&	
7	||	
8	=	Right

Operators that have the same precedence are evaluated according to their *associativity* e.g.:

```
a = b = c;    //
    evaluates as
a = (b = c);
```

because = is right associative.While:

```
a * b / c;    //
    evaluates as
(a * b) / c;
```

because * and / are left associative. See[4] for a full list of operators and their precedence.

[4] http://en.cppreference.com/w/cpp/language/operator_precedence

Salida Estándar (cout)

You've already met `cout`, it uses the *indirection operator* (<<) to print to the screen e.g.:

```
std::cout << "Have some pi: ";
std::cout << 3.1415926;
std::cout << a;
```

Salida Estándar (cout)

You've already met cout, it uses the *indirection operator* (<<) to print to the screen e.g.:

```
std::cout << "Have some pi: ";
std::cout << 3.1415926;
std::cout << a;
```

We can use << more than once in the same statement:

```
double t0 = 1.5, t1 = 2.5;
cout << "t0: " << t0 << ", t1: " << t1
     << ", delta: " << t1 - t0;
```

Output: `t0:  1.5, t1:  2.5, delta:  1`

Salida Estándar (cout)

You've already met cout, it uses the *indirection operator* (<<) to print to the screen e.g.:

```
std::cout << "Have some pi: ";
std::cout << 3.1415926;
std::cout << a;
```

We can use << more than once in the same statement:

```
double t0 = 1.5, t1 = 2.5;
cout << "t0: " << t0 << ", t1: " << t1
     << ", delta: " << t1 - t0;
```

Output: `t0:  1.5, t1:  2.5, delta:  1`

If you want a new line you have to use `\n`:

```
double t0 = 1.5, t1 = 2.5;
cout << "t0: " << t0 << "\nt1: " << t1 << "\n";
```

Output:

```
t0:  1.5
t1:  2.5
```

Standard input (cin)

Extracting information out of the user

To get input from the user, use the extraction operator (>>) of the cin object (pronounced *see-in*) e.g.:

```
  double radius;
  std::cin >> radius;
```

at this point the program will stop and wait for the user to enter a number and push RETURN.

Standard input (`cin`)

Extracting information out of the user

To get input from the user, use the extraction operator (>>) of the `cin` object (pronounced *see-in*) e.g.:

```
double radius;
std::cin >> radius;
```

at this point the program will stop and wait for the user to enter a number and push RETURN. As with output we can use >> more than once in a statement e.g.:

```
std::cin >> width >> height;
```

in this case the program will wait for two sets of numbers to be entered. They can be separated by a space, tab or a newline.

Ejemplo Completo

```
#include <iostream>

int main()
{
  unsigned int width, height;
  std::cout << "Please enter a width and height: ";
  std::cin >> width >> height;
  std::cout << "Area is: " << width * height << "\n";

  const double ratio = static_cast<double>(height) /
    static_cast<double>(width);
  std::cout << "Ratio is: 1:" << ratio
            << " (width:height)" << "\n";

  const bool isSquare = (width == height);
  std::cout << "Is it a square? " << isSquare << "\n";

  return 0;
}
```