Cours de C++

Introduction

Cécile Braunstein
cecile.braunstein@lip6.fr

Généralité

Notes

- Interros cours 1/3
- Contrôle TP 1/3
- Mini-projet 1/3
- Bonus (Note de Participation) jusqu'à 2 points

Références

- H. GARRETA, Le langage et la bibliothèque C++, ellipses 2000
- A. KOENIG, B. MOO, Accelerated C++, Addison Wesley 2006
- `http://www.cplusplus.com`
- `http://www.google.com`
- `http://www.cppreference.com`

Why C++ ?

C++

- Middle-level language
- Developed by Bjarne Stroustrup in 1979 at Bell Labs
- First view as an extension of C (*C with classes*)

Philosophy

- C++ is designed to be a statically typed, general-purpose language that is as efficient and portable as C
- C++ is designed to give the programmer choice, even if this makes it possible for the programmer to choose incorrectly
- C++ is object language it implies :
  - Re-use
  - Modularity
  - Maintainability

Hello world !

helloworld.cpp

```
// The first programm

#include <iostream>

int main()
{
  std::cout << "Hello, world !" << std::endl ;

  return 0;
}
```

Call the compiler

```
g++ -Wall -g helloworld.cpp -o hello
```

Compile Options

g++ accepts most options as gcc

- Wall : all warnings
- g : include debug code
- o : specify the outfile name (`a.out` by default)

Program details

#include

Many fundamentals facilities are part of standard library rather than core language

```
#include <iostream>
```

`#include` directive + angle brackets refers to standard header

main function

- Every C++ program must contain a function named `main`. When we run the program, the implementation call this function.
- The result of this function is an integer to tell the implementation if the program ran successfully
  Convention :

$$0 : \textit{success} \quad | \quad \neq 0 : \textit{fail}$$

Using the standard library for output

```
std::cout << "Hello, world !" << std::endl;
```

- **< <** : output operator
- **std : :** : namespace `std`
- **std : :cout** : standard output stream
- **std : :endl** : stream manipulator

Expressions in C++

Type

- A variable is an object that has a name.
- An object is a part of the memory that has a type.
- Every object, expression and function has a type.
- Types specify properties of data and operations on that data.

Primitive types

bool	char	int	unsigned
long	short	float	double

To improve the code re-use it is important to use the right type at the right place !

Expressions in C++
Type conversion

Goal : bring the operands to each operator to a common type.

Some basic rules

1. Preserve information
2. Prefer unsigned conversion
3. Promote values to larger type
4. Preserve precision

Expressions in C++
Type conversion

Exercice

```
#include<iostream>
int main()
{
  float a = 1.2;
  float c;
  int   b;

  b = a; a = c;

  std::cout << c
            << std::endl;
  return 0;
}
```

```
#include<iostream>
int main ()
{
  int          d = -1;
  unsigned int e = d;

  std::cout << e
            << std::endl;
  return 0;
}
```

Variable definition

```
#include <iostream>
#include <string>

int main()
{
  std::cout
  <<"Your first name: ";

  std::string name;
  std::cin >> name;

  std::cout << "Hello, "
  << name << "!"
  << std::endl;
  return 0;
}
```

- Variable can be define anywhere in the program.
- local variable are destroyed when an end of block is reached.
- Variable has a type and an interface
- The variable `name` is implicitly initialize

How to define a variable

type-name name ;
type-name name = value ;
type-name name(args) ;

Expressions in C++

Definition

An expression ask the implementation to compute something.
The computation yields a result and may have side effects

Example

```
4+3
```

- Result :
- Side effect :

```
std::cout << "Hello, world !" << std::endl;
```

- Result :
- Side effect :

Expressions in C++

Definition

An expression ask the implementation to compute something.
The computation yields a result and may have side effects

Example

```
4+3
```

- Result :
- Side effect :

```
std::cout << "Hello, world !" << std::endl;
```

- Result :
- Side effect :

Expressions in C++

Definition

An expression ask the implementation to compute something.
The computation yields a result and may have side effects

Example

```
4+3
```

- Result : 7
- Side effect :

```
std::cout << "Hello, world !" << std::endl;
```

- Result :
- Side effect :

Expressions in C++

Definition

An expression ask the implementation to compute something.
The computation yields a result and may have side effects

Example

```
4+3
```

- Result : 7
- Side effect : none

```
std::cout << "Hello, world !" << std::endl;
```

- Result :
- Side effect :

Expressions in C++

Definition

An expression ask the implementation to compute something.
The computation yields a result and may have side effects

Example

```
4+3
```

- Result : 7
- Side effect : none

```
std::cout << "Hello, world !" << std::endl;
```

- Result : std : :cout
- Side effect :

Expressions in C++

Definition

An expression ask the implementation to compute something.
The computation yields a result and may have side effects

Example

```
4+3
```

- Result : 7
- Side effect : none

```
std::cout << "Hello, world !" << std::endl;
```

- Result : std : :cout
- Side effect : writes `Hello, world !` and ends the output line

Overloaded operators

The effect of an operator depends on the type of its operands.

Example

```
#include<iostream>
#include<string>
int main()
{ // Example 1
  int a = 2;
  int b = 3;
  std::cout << a + b << std::endl;

  // Example 2
  std::string s = "Hello,";
  std::cout << s + "World !" << std::endl;

  return 0;
}
```

Operators Precedence

`x.y`
`a[k]`
`++x , -- -x`
	The member `y` of object `x`
	The element in object `a` indexed by `k`
	Incr/decr `x`, returning the original value of `x`

`x++ , x-- `
`&x`
`*x`
	Incr/decr `x`, returning the resulted value of `x`
	address of `x`
	value at the address `x`

`* , /` `%`	Product, quotient Remainder ; `x-((x/y)*y)`
`+ , -`	Sum, Subtract
`== , !=` `&&`	Yields a `bool` indicating whether `x`(is not) equal(s) `y` logic AND
`||`	logic OR
`x = y` `x op= y`	Assign the value `y` to `x`, yielding `x` as its result Compound assignment operator

if/else statement

```
if ( a == 0 )
  std::cout << "a equals zero" << std::endl;
```

use a block :

```
if ( a == 0 ) {
  std::cout << "a equals zero" << std::endl;
a = 1;
}
else{
  std::cout << "a is not equal" << std::endl;
}
```

Other syntax
While loop

```
while ( a > 0 ) {
  a--;
}
```

```
do {
  a--;
while ( a > 0 );
```

Other syntax
For loop

```
for ( a = 5 ; a > 0 ; a++ ) {
  std::cout << "a := " << a << std::endl;
}
```

```
for ( a = 5 ; a > 0 ; a++ ) {
  if ( a == 3 ) continue;
  if ( a == 1 ) break;;
 std:: cout << "a := " << a << std::endl;
}
```

switch statement

```
for ( value=5 ; value>0 ; value-- ) {
  switch ( value ) {
    case 1: std::cout << "Case 1." << std::endl; break;
    case 2: std::cout << "Case 2." << std::endl;
    case 3: std::cout << "Case 3." << std::endl; break;
    default:
      std::cout << "Unknown." << endl;
  }
}
```

The type `vector`

This a container

Contains a collection of value. Each value has the same type. Different vectors can hold value of different types.

Template classes

A container is defined with the template classes.
Separate the what it means to be `vector` from the type of the data inside.

➢ A lot of operation does not depend on the type of the element.

The type `vector`
Operations

Action	Method
Insert an element	`v.push_back()`
Remove an element	`v.pop_back()`
Remove all elements	`v.clear()`
Returns a value that denotes the first element	`v.begin()`
Returns a value that denotes (one past) the last element	`v.end()`
Returns a value that denotes the i^{th} element	`v[i]`
Take the vector size	`v.size()`
Check emptiness	`v.empty()`

Attention : The first element is indexed by 0 and the last by size-1.