Introduction to C++

Georgia Koutsandria

Internet of Things A.Y. 18-19
Prof. Chiara Petrioli
Dept. of Computer Science
Sapienza University of Rome

PART I

What is C++?

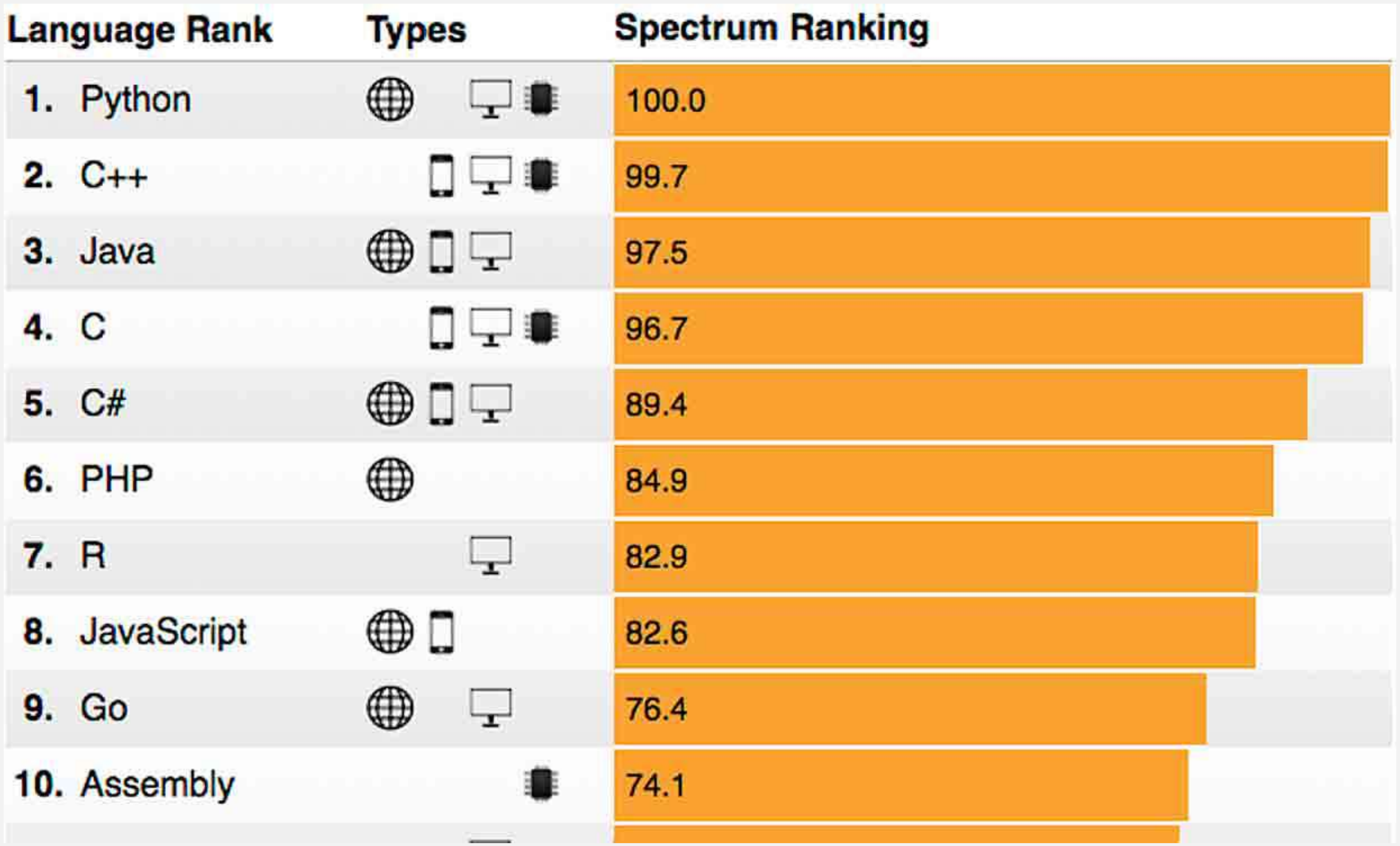
- Developed by Bjarne Stroustrup in 1979 at Bell Labs.
- A superset of C; More processing tasks than C.
- Middle-level programming language
  - Imperative
  - Object-oriented
  - Low-level memory manipulation.

```
EXP_COM[] = {
*RV_TIME[] = {

int summary(void *barg,void
{
    char *str = (char *)arg;
    st_board *board = (st_b
    int ret = 0;

char *ptr_shutter
```

Why C++?



Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥 ▪	100.0
2. C++	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	97.5
4. C	📱 🖥 ▪	96.7
5. C#	🌐 📱 🖥	89.4
6. PHP	🌐	84.9
7. R	🖥	82.9
8. JavaScript	🌐 📱	82.6
9. Go	🌐 🖥	76.4
10. Assembly	▪	74.1

Why C++ in IoT?

- System programming
- Embedded programming
- Resource-constrained devices
- Large systems
- …

```
EXP_COM[] = {
*RV_TIME[] = {

int summary(void *barg,void
{
    char *str = (char *)arg;
    st_board *board = (st_b
    int ret = 0;

char *ptr_shutter
```

Compilers

What is a compiler?

- Computers understand only one language which is called *machine language.*

- This language consists of a set of instructions made of ones and zeros.

- Compilers translate high-level programing languages into machine language.

```
#include <iostream>
using namespace std;
int main ()
{
   count <<"Hello World!";
   return 0;
}
```

```
0101010101010101010
1010101010101010101
1010100010010101010
1010101010101010101
1010101010101010101
```

Oh, now I understand!
Compiler

How to compile a C++ program

- **Windows**: Install an Integrated Development Interface (IDE).
  - Dev-C++ http://www.bloodshed.net/dev/index.html

- **Mac:** Install Xcode with the gcc/clang compilers.

```
g++ -std=c++11 example.cpp -o example_program OR
clang++ -std=c++11 -stdlib=libc++ example.cpp -o example_program
```

- **Linux:** Compile your code directly from the terminal using the following commnad

```
g++ -std=c++0x example.cpp -o example_program
```

Basics of C++

Syntax of C++

- Code is usually written in files with a .cpp extension

- Lines of comments are ignored by the compilers

  // or / …. */ (single line or block of lines)*

- C++ code is case sensitive

  INT is not the same as int is not the same as Int

Structure of a program in C++

```
#include <iostream>

int main(){

    std::cout << "Hello World!" << std::endl;

    return 0;
}
```

Structure of a program in C++

Include files/libraries

A C++ standard library to perform I/O to screen

main(): A funtion of type *int*

```
#include <iostream>

int main(){

        std::cout << "Hello World!" << std::endl;

        return 0;
}
```

return or end the program
A returned ‘0’ indicates success

Structure of a program in C++

```
#include <iostream>

int main(){

        std::cout << "Hello World!" << std::endl;

        return 0;
}
```

std is the 'namespace' for the standard library and contains a wide range of functions

cout is a 'stream' which prints variables and text to screen

defines the text to be printed to screen

special stream object which 'ends the line' and flushes the buffer (more later)

Structure of a program in C++

```
#include <iostream>

int main(){

        std::cout << "Hello World!" << std::endl;

        return 0;
}
```

semicolon to end each statement

Exercise

1. Write a program that prints your name

Exercise 1-Solution

```
#include <iostream>

int main()
{

    std::cout << "Georgia Koutsandria"<<std::endl;
    return 0;
}
```

Variables and Basic Types

What is a variable?

- A portion of memory to store a value that has a name and is of a specific type.
- **Name**: Distinguishes a variable from other variables.
- **Type**: Determines the meaning of the data and operations.

Fundamental Data Types

`int` : integer (7, 1024, …)
`bool` : logical (true, false)
`float` : single precision real number 1.234f, -3.86f
`double` : double precision real number 1.234f, -3.86f
`char` : character variable (‘a’, ‘b’, ‘k’, etc..)

- Let’s declare an integer variable called ‘k’ -> `int k;`
- Let’s assign an initital value to ‘k’-> `int k = 10;`

Fundamental Data Types

Type	Meaning	Min. Size
`bool`	boolean	NA
`char`	character	8 bits
`short`	short integer	16 bits
`int`	integer	16 bits
`long`	long integer	32 bits
`float`	single-precision floating-point	6 significant bits
`double`	double-precision floating-point	10 significant bits

Signed and Unsigned Types

- A `signed` type represents negative or positive numbers (including zero)
- An `unsigned` type represents only values greater than or equal to zero

`int` `long` `short`

signed types

- The corresponding `unsigned` type is obtained by adding unsigned top the type
- E.x.: `unsigned long`

Declaring (initialized) variables

A simple variable definition consists of a type specifier, followed by a list of one or more variable names separated by commas, and ends with a semicolon.

```
int k = 123;
bool flag = true;
float dinstance = 1.238f;
double time = 1.0;
char character = ‘b’;
```

- *Always initialize your variables! Uninitialized variables have a value which is compiler dependent.*
- *Real constants are always declared as double precision. Use ‘f’ suffix to specify single precision.*

Type deduction: auto

- When a new variable is initialized, compilers can automatically figure out the type of a variable by the initializer.

```
int foo = 0;
auto bar = foo; //same as int bar = foo;
```

- The type of `bar` is the type of the value used to initialize it, which is the type of `foo` (int).

Introduction to strings

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main(){
      string mystring;
      mystring = " This is a string ";
      cout << mystring;
      return 0;
}
```

Stores sequences of characters

includes the header <string>

initializes string

Operators

Operators

- The assignment operator (=)
  - `x = 100;`
- Simple arithmentic operations
  - Addition: +
  - Subtraction: -
  - Multiplication: *
  - Division: /
  - Modulo: %

Compound assignment

- They modify the current value of a variable by performing an operation on it :

  (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

expression	equivalent to..
`y += x;`	y = y + x;
`x -= 5;`	x = x -5;
`x /=y;`	x = x / y;
`x *= y+1;`	x = x * (y+1);

Example

```
// compound assignment operators
#include <iostream>
using namespace std;

int main ()
{
  int a, b=3;
  a = b;
  a+=2;            // equivalent to a=a+2
  cout << a;
}
```

Increment and decrement

- The increase(++) and the decrease(--) opetator, increase or reduce by one the value stored in a variable.
- They can be used both as a prefix and as a suffix (++x or x++).

```
x = 3;
y = ++x; // y contains 4
w = x++; // y contains 3
z = --x; // z contains 2
k = x--; // k contains 3
```

Relational and comparison operators

- They can be used to compare two expressions.
- The result of such operations is either true or false.

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Logical operators

- The operator ! is used for the boolean operation NOT.
- The operator && corresponds to the boolean logical operator AND.
- The operator || corresponds to the boolean logical operator OR.

```
!true // evaluates to false
!(6 <= 4) // evaluates to true
((5 == 5) && ( 3 > 6 )) // evaluates to false
((5 == 5) || ( 3 > 6 )) // evaluates to true
```

Basic Input/Output

Streams

- C++ uses convenient abstraction to perform input and output operations in sequential media, e.g., screen, keyboard or a file.

- **Stream:** Insert or extract characters to/from.

```
#include <iostream>
```

Standard output (cout)

- Default standarad output: screen
- It is used together with the insertion operator (<<)

```
// prints Output sentence on screen
cout << " Output sentence";
// prints number 2 on screen
cout << 2;
// prints the value of x on screen
cout << x;
```

Standard output (cout)

```
// prints Output sentence on screen
cout << " Output sentence";
// prints number 2 on screen
cout << 2;
// prints the value of x on screen
cout << x;
```

When the text is enclosed in double quotes (“), the text is printed literally

The << operator inserts the data that follow it into the stream that precedes it

Standard output (cout)

- Multiple insertion operations (<<) may be chained in a single statement:

```
cout << " This " << " is " << " an " << " example. ";
cout << " I am " << age << " years old. ";
```

- To add line breaks at the end, cout has to be instructed to do so:

```
cout << " First sentence.\n ";
cout << " Second sentence.\nThird sentence.";
                        OR
cout << " First sentence. " << endl;
```

Standard input (cin)

- Default standarad input: keyboard
- It is used together with the extraction operator (>>)

```
int age;
cin >> age;
```

Extracts from cin a value to be stored in the variable age

Declares a variable of type `int` called age

- The characters introduced using the keyboard are only transmitted to the program when the ENTER (or RETURN) key is pressed.

I/O example

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

int main(){
     int i = 0;
     cout << "Please enter an integer value:  ";
     cin >> i;
     cout << "The value you entered is " << i ;
     cout << " and its double is "  << i*2 << ".\n " ;
     return 0;
}
```

Standard input and strings

- cin extraction always considers spaces (whitespaces, tabs, new-line,..) as terminating the value being extracted.
- Extracts a single word, not a phrase or an entire sentence.
- Function *getline* takes the stream(cin) as first argument, and the string variable as second.

Standard input and strings

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

int main(){
      string mystr;
      cout << "What's your name? ";
      getline (cin, mystr);
      cout << "Hello " <<  mystr << "!\n " ;
      return 0;
}
```

Standard input and strings

- The standard header <sstream> defines a type called stringstream.

- Covert strings to numerical values and vice versa.

```
string mystr ("1204");
int myint;
stringstream(mystr) >> myint;
```

Standard input and strings

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

int main(){
        string mystr;
        float price=0;
        int quantity=0;
        cout << "Enter price: ";
        getline (cin, mystr);
        stringstream(mystr) >> price;
        cout << "Enter quantity: ";
        getline (cin, mystr);
        stringstream(mystr) >> quantity;
        cout << "Total price: " <<  price*quantity << endl;
        return 0;}
```

Exercices

1. Write a program that prompts the user to input two integer numbers, then performs their sum, and prints result.

2. Write a program that promts the user to input the sentence "This is my first sentence.", and prints that sentence.

3. Write a program that promts the user to input a float to be stored as a string, converts it to float and prints it.

Exercise 1-Solution

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

int main(){
	int firstNum = 0, secondNum=0, sum=0;
	cout << "Enter the first number: ";
	cin >> firstNum;
	cout << "Enter the second number: ";
	cin >> secondNum;
	sum = firstNum + secondNum;
	cout << "This is the sum: " << sum << ".\n";
	return 0;
}
```

Exercise 2-Solution

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

int main(){
      string sentence;
      cout << "Enter a sentence: ";
      getline(cin, sentence);
      cout << "You entered: " << sentence << ".\n";
      return 0;
}
```

Exercise 3-Solution

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

int main(){
      string mystr;
      float num = 0;
      cout << "Enter a float number: ";
      getline(cin, mystr);
      stringstream(mystr) >> num;
      cout << "You entered: " << num << ".\n";
      return 0;
}
```

Statements and Flow Control

Statements

- Used for declaration, expression, conditional execution, jump statements, loops etc..
- Most statements end with a semicolon (`;`)
- Common errors

✗
```
int k = 123 //missing semicolon
```
✗
```
int k = 123;; //extraneous semicolon
```
✓
```
int k = 123; //single semicolon
```

Conditional Statements

1. *`if`* statement: Determines the flow of control based on a condition.

```
if (condition)
   statement
```

2. *`switch`* statement: Evaluates an integral expression and chooses one of several execution paths based on the expression's value.

```
switch (condition)
   statement
```

Condition(s)

- The *`Condition`* must be enclosed in parenthesis
- It can be an expression or an initialized variable declaration. It must have a type that is convertible to `bool`.

```
if (condition)
     statement
```

True The statement is executed.

False The statement is skipped.

The *if* Conditional Statement

- Conditionally executes another statement based on whether a specified condition is true.

simple if

```
int number=0;
cout << "Enter an integer: ";
cin >> number;
// checks if the number is positive
if ( number > 0) {
      cout << "You entered a positive integer: " << number << endl;
}
```

Condition

The *if-else* Conditional Statement

```
int number=0;
cout << "Enter an integer: ";
cin >> number;
// checks if the number is positive
if ( number >= 0) {
      cout << "Positive integer: " << number << endl;
}
else{
      cout << "Negative integer: " << number << endl;
}
```

Nested *if* Conditional Statement

```
int number=0;
cout << "Enter an integer: ";
cin >> number;
// checks if the number is positive
if ( number > 0) {
        cout << "Positive integer: " << number << endl;
}
else if ( number < 0) {
        cout << "Negative integer: " << number << endl;
}
else{
        cout << "You entered 0. " << number << endl;
}
```

The *switch* Conditional Statement

- A convenient way of selecting among a (possible large) number of fixed alternatives.

```
switch(x){
    case 1:
       cout << "x is 1";
       break;
    case 2:
       cout << "x is 2";
       break;
    default:
       cout << "value of x is unknown";

}
```

Iterative Statements (loops)

- Repeated execution until a condition is true
- Statements that test the condition before executing the block: `while, for`
- Statement that executes the body and then tests the condition: `do while`

```
while (condition)
      statement
```

```
for (initializer; condition; expression)
      statement
```

```
do
          statement
while  (condition);
```

The for loop

- It repeats *statement* while *condition* is true.

```
#include <iostream>
using namespace std;
int main(){
     for (int n=0; n<5; n++)
          cout << n << " ";
     cout << endl;
}
```

The while loop

- It simply repeats *statement* while *condition* is true.
- The loop ends if, after any execution of *statement*, *expression* is no longer true.

```
#include <iostream>
using namespace std;
int main(){
     int n = 10;
     while (n>0){
       cout << n << ", ";
       --n;
     }
     cout << "liftoff!\n";
}
```

The do-while loop

- It behaves like the *while* loop, except that *condition* is evaluated after the execution of the *statement*.

```
#include <iostream>
using namespace std;
int main(){
     string str;
     do {
       cout << "Enter text: ";
       getline(cin,str);
       cout << "You entered: " << str << "\n";
     } while(str!="ciao");
}
```

Jump Statements

The break statement

- It leaves a loop, even if the condition for its end is not fullfilled
- It can be used to end an infinite loop, or to force it to end before it natural end
- E.g., Let’s stop the countdown before its natural end

The break statement

```
//break loop example
#include <iostream>
using namespace std;
int main(){
     for (int n=10; n>0; n--)
     {
          cout << n << ", ";
          if (n==3)
          {
               cout << "Countdown aborted!";
               break;
          }
     }
}
```

The continue statement

- It causes the program to skip the rest of the loop in the current iteration, causing it to jump to the start of the following iteration.

- E.g., Let’s skip number 5 in the countdown example

The continue statement

```
//continue loop example
#include <iostream>
using namespace std;
int main(){
     for (int n=10; n>0; n--){
          if (n==5)
               continue;
          cout << n << ", ";
     }
     cout << "liftoff!\n";

}
```

The return statement

- It terminates the function that is currently executing and returns control to the point from which the function was called.

- Two forms of return statemens:

```
return;
return statement;
```

Exercises

1. Write a program that prompts the user to input three integer number and finds the greatest value among them. E.g., if input numbers are 10, 15, and 20, then the greatest number is 20. (use only if statements)

Exercise 1-Solution

```
//find the greatest number among 3
#include <iostream>
using namespace std;

int main(){
    float n1, n2, n3;
    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;
    if(n1 >= n2 && n1 >= n3)
        cout << "Largest number: " << n1 << endl;
    if(n2 >= n1 && n2 >= n3)
        cout << "Largest number: " << n2 << endl;
    if(n3 >= n1 && n3 >= n2)
        cout << "Largest number: " << n3 << endl;

    return 0;
}
```

Exercises

2. Write a program that prompts the user to input three integer values and finds the greatest value among them.E.g., if input numbers are 10, 15, and 20, then the greatest value is number 20. (use if/else if/else statements)

Exercise 2-Solution

```
//find the greatest number among 3
#include <iostream>
using namespace std;

int main(){
    float n1, n2, n3;
    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;
    if(n1 >= n2 && n1 >= n3)
        cout << "Largest number: " << n1 << endl;
    else if(n2 >= n1 && n2 >= n3)
        cout << "Largest number: " << n2 << endl;
    else
        cout << "Largest number: " << n3 << endl;

    return 0;
}
```

Exercises

3. Write a program to print the first 10 integer numbers (excluding zero, starting from 1 to 10).

Exercise 3-Solution

```
//print the first 10 integer numbers
//excluding 0 (from 1 to 10)
#include <iostream>
using namespace std;
int main(){
     cout << "These are the first 10 integers: ";
     for (int i=1; i <= 10; i++)
          cout << i << " ";
     cout << endl;
     return 0;
}
```

Exercises

4. Write a program that prints the squares of the numbers from 0 to 20. E.g., 0 1 4 9 16 25 36 … 400

Exercise 4-Solution

```
//find the squares of numbers from 0 to 20
#include <iostream>
using namespace std;
int main(){
      for (int i=0; i < 21; i++)
            cout << i*i << " ";
      cout << endl;
      return 0;
}
```

Exercises

5. Write a program to find the sum of digits of a given number. E.g., if input number is 1234, then the sum is 10.

Exercise 5-Solution

```
//find the sum of digits of a given number
#include <iostream>
using namespace std;
int main(){
      int num=0, val=0, sum=0;
      cout << "Enter a number: ";
      cin >> val;
      num = val;
      while (num!=0){
            sum += num%10;
            num /= 10;
      }
      cout << "The sum of the digits of " << val << " is ";
      cout << sum << ".\n ";
      return 0;
}
```

Exercises

6. Write a program that prompts the user to enter integer numbers and prints their sum until user enters number 0. Hint: use do..while

Exercise 6-Solution

```
//enter numbers until 0 is given as input
//print the sum of them
#include <iostream>
using namespace std;
int main(){
      int num=0, sum=0;
      do{
            cout << "Enter a number: ";
            cin >> num;
            sum += num;
      }while (num!=0);
      cout << "The sum of the numbers is " << sum;
      cout << ".\n ";
      return 0;
}
```

Additional Resources

- http://www.cplusplus.com/doc/tutorial/
- https://en.cppreference.com/w/
- Programming: Principles and Practice Using C++, Bjarne Stroustrup (Updated for C++11/C++14)
- C++ Primer, Stanley Lippman, Josée Lajoie, and Barbara E. Moo (Updated for C++11)