Introduction to Programming (in C++)

Introduction

Jordi Cortadella, Ricard Gavaldà, Fernando Orejas
Dept. of Computer Science, UPC

Outline

- Programming examples
- Algorithms, programming languages and computer programs
- Steps in the design of a program

First program in C++

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

// This program reads two numbers and
// writes their sum

int main() {
    int x, y;
    cin >> x >> y;
    int s = x + y;
    cout << s << endl;
}
```

```
> sum
8 13
21
> sum
-15 9
-6
>
```

cout

Calculate x^y

- Algorithm: repeated multiplication

$$\underbrace{x \cdot x \cdot x \cdot \cdots \cdot x}_{\text{y times}}$$

y	x	i	$p=x^i$
4	3	0	1
4	3	1	3
4	3	2	9
4	3	3	27
4	3	4	81

Calculate x^y

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

// Input:  read two integer numbers, x and y,
//         such that y >= 0
// Output: write x^y

int main() {
    int x, y;
    cin >> x >> y;

    int i = 0;
    int p = 1;
    while (i < y) { // Repeat several times (y)
        i = i + 1;
        p = p*x;    // p = x^i
    }
    cout << p << endl;
}
```

Prime factors

- Decompose a number in prime factors
  - Example: input 350 output 2 5 5 7

- Intuitive algorithm:
  - Try all potential divisors d, starting from 2
    - If divisible by d, divide and try again the same divisor
    - If not divisible, go to the next divisor
  - Keep dividing until the number becomes 1

Prime factors

n	d	divisible	write
350	2	yes	2
175	2	no	
175	3	no	
175	4	no	
175	5	yes	5
35	5	yes	5
7	5	no	
7	6	no	
7	7	yes	7
1	**finish**		

The algorithm will never write a non-prime factor. Why ?

Prime factors

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

// Input:  read a natural number n > 0
// Output: write the decomposition in prime factors

int main() {
    int n;
    cin >> n;
    int d = 2;  // Variable to store divisors

    // Divide n by divisors from 2 in ascending order
    while (n != 1) {
        if (n%d == 0) { // Check if divisible
            cout << d << endl;
            n = n/d;
        }
        else d = d + 1;
    }
}
```

ALGORITHMS, PROGRAMMING LANGUAGES AND COMPUTER PROGRAMS

An algorithm

- An algorithm is a method for solving a problem. It is usually described as a sequence of steps.

- Example: How can we find out whether a number is prime?
  - Read the number (N).
  - Divide N by all numbers between 2 and N-1 and calculate the remainder of each division.
  - If all remainders are different from zero, the number is not prime. Otherwise, the number is not prime.

A programming language

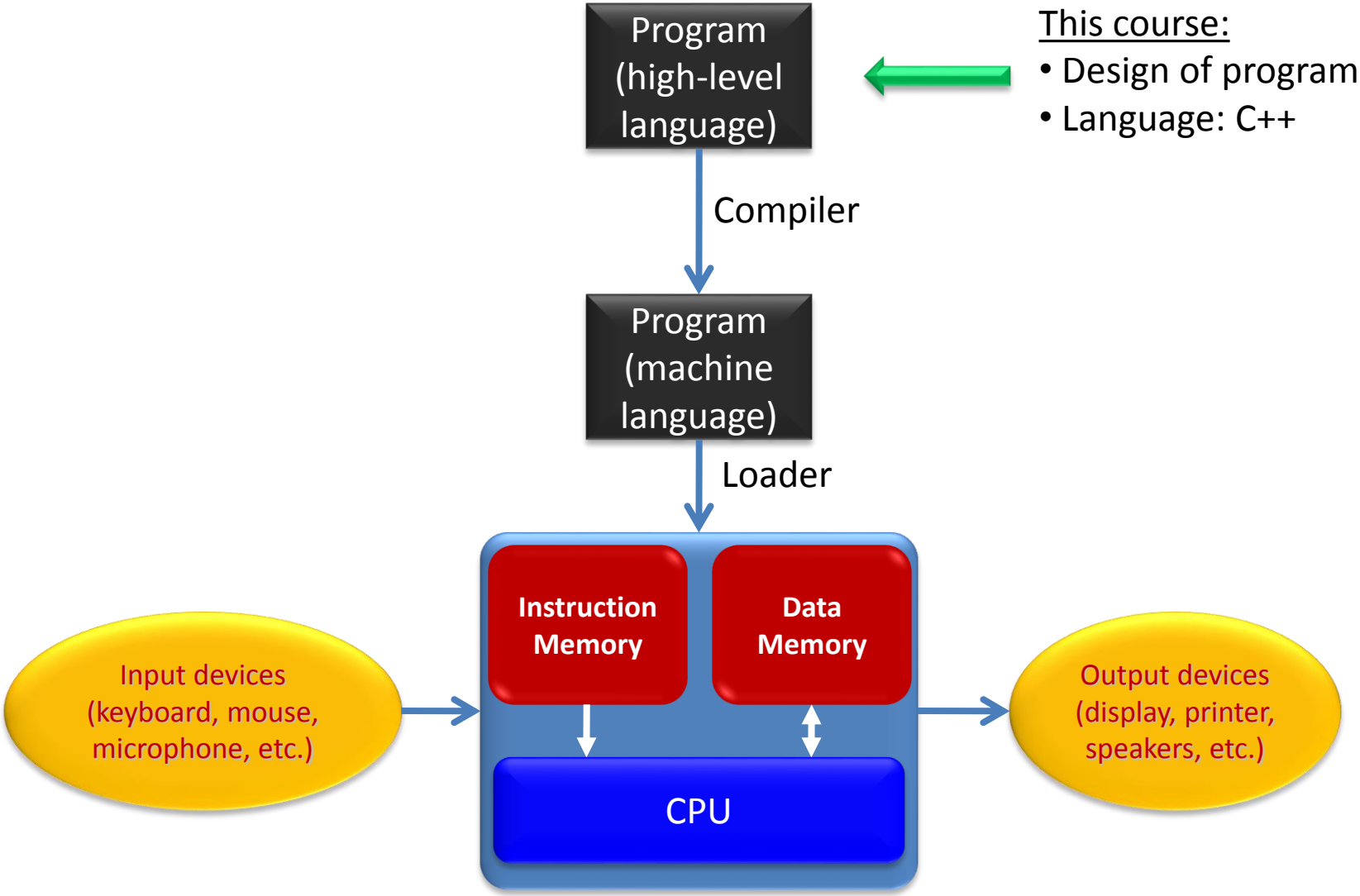
- A programming language is a language used to describe instructions for a computer.
- What's in a programming language?
  - Data (numbers, strings, structures, …)
  - Instructions (arithmetic, sequence, repetition, …)
- A programming language has very strict syntax and semantics, as it must be understood by a computer!

A computer program

- A computer program is an algorithm written in a in a programming language that executes a certain task.

- Examples of tasks a computer program can execute:
  - Calculate the square root of a number
  - Find the number of times the word “equation” appears in a math book
  - Play a music file
  - Find the shortest path between two cities

A computer system



High-level language

- Computers understand very low-level instructions (machine language).
- Software is usually constructed using high-level languages.
  - Higher productivity
  - Better readability
  - Simpler debugging
  - But some time and memory efficiency may be lost
- A compiler can translate a high-level language into machine language automatically.
- There is a huge number of programming languages: C, C++, Java, Pascal, PHP, Modula, Lisp, Python, Excel, Fortran, Cobol, APL, Basic, Tcl, Ruby, Smalltalk, Haskell, Perl, SQL, Prolog, …

Assembly and machine language

		.begin	
		.org 2048	
	a_start	.equ 3000	
2048		ld length,%	
2064		be done	00000010 10000000 00000000 00000110
2068		addcc %r1,-4,%r1	10000010 10000000 01111111 11111100
2072		addcc %r1,%r2,%r4	10001000 10000000 01000000 00000010
2076		ld %r4,%r5	11001010 00000001 00000000 00000000
2080		ba loop	00010000 10111111 11111111 11111011
2084		addcc %r3,%r5,%r3	10000110 10000000 11000000 00000101
2088	done:	jmpl %r15+4,%r0	10000001 11000011 11100000 00000100
2092	length:	20	00000000 00000000 00000000 00010100
2096	address:	a_start	00000000 00000000 00001011 10111000
		.org a_start	
3000	a:		

(From http://en.wikipedia.org/wiki/Assembly_language)

STEPS IN THE DESIGN OF A PROGRAM

Steps in the design of a program

1. Specification
   - The task executed by the program must be described rigorously (without ambiguities).

2. Design of the algorithm
   - The method for executing the task must be selected and designed in such a way that the program is correct according to the specification.

3. Coding in a programming language
   - The algorithm must be written in a programming language that can be executed by the computer.

4. Execution
   - The program must be executed with a set of examples that reasonably cover all the possible cases of data input. If the program does not work properly, the algorithm will have to be redesigned.

Example

- Design a program that
  - given a natural number representing a certain amount of time in seconds (N),
  - calculates three numbers (h, m, s) that represent the same time decomposed into hours (h), minutes (m) and seconds (s)
  - Example
    - Given N=3815,
    - Calculate h=1, m=3, s=35

Specification

- Precondition:
  - Specification of the data before the program is executed
- Postcondition:
  - Specification of the data after the program is executed
- Example
  - Precondition: $N \geq 0$
  - Postcondition: $3600*h + 60*m + s = N$

Specification

- Alternatively, specifications can describe the input and output data of a program.

  Input: the program reads a natural number representing a number of seconds.

  Output: the program writes the same time decomposed into hours, minutes and seconds.

- Specifications can be described in many ways, e.g. using plain English or formal logic propositions.

- Even when written in English, specifications must be rigorous and unambiguous.

A bad specification

- Precondition: $N \geq 0$
- Postcondition: $3600*h + 60*m + s = N$,

A bad specification

- Does the specification really describe what the program is supposed to calculate?

- Example
  - Assume N = 3815
  - The solution h=1, m=3, s=35 meets the specification (1∗3600 + 3∗60 + 35 = 3815)
  - But the solutions h=0, m=30, s=2015 and h=0, m=0 and s=3815 also meet the specification. What's wrong?

A good specification

- Precondition: $N \geq 0$
- Postcondition: $3600*h + 60*m + s = N$,
  $0 <= s < 60,\ 0 <= m < 60$

- The solution h=1, m=3, s=35 fulfils the specification.
- The solutions h=0, m=30, s=2015 and h=0, m=0, s=3815 do not.

Algorithms

- An algorithm:
  - h = N / 3600 (integer division)
  - m = (N *mod* 3600) / 60 (*mod*: remainder)
  - s = N *mod* 60

- Another algorithm:
  - s = N *mod* 60
  - x = N / 60
  - m = x *mod* 60
  - h = x / 60

- Many algorithms may exist to solve the same problem. Use the most efficient one whenever possible. But, which one is the most efficient? There is no easy answer.

Program in C++

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

// This program reads a natural number that represents an amount
// of time in seconds and writes the decomposition in hours,
// minutes and seconds

int main() {
    int N;
    cin >> N;
    int h = N / 3600;
    int m = (N % 3600) / 60;
    int s = N % 60;
    cout << h << " hours, " << m << " minutes and "
         << s << " seconds" << endl;
}
```

Execution

```
> decompose_time
3815
1 hours, 3 minutes and 35 seconds

> decompose_time
60
0 hours, 1 minutes and 0 seconds
```