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++

```cpp
#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

$$x \cdot x \cdot x \cdot \cdots \cdot x$$

$y$ times

<table>
<thead>
<tr>
<th>$y$</th>
<th>$x$</th>
<th>$i$</th>
<th>$p=x^i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>81</td>
</tr>
</tbody>
</table>
Calculate $x^y$

```cpp
#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;
}
```

Introduction to Programming © Dept. CS, UPC
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

<table>
<thead>
<tr>
<th>n</th>
<th>d</th>
<th>divisible</th>
<th>write</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>2</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>175</td>
<td>2</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>3</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>4</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>5</td>
<td>yes</td>
<td>5</td>
</tr>
<tr>
<td>35</td>
<td>5</td>
<td>yes</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>yes</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>finish</td>
<td></td>
</tr>
</tbody>
</table>

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

Program (high-level language) → Compiler → Program (machine language) → Loader → Instruction Memory — Data Memory → CPU → Output devices (display, printer, speakers, etc.)

Input devices (keyboard, mouse, microphone, etc.)

This course:
• Design of programs
• Language: C++

This course:
• Design of programs
• Language: C++
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

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Machine Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048</td>
<td><code>ld length,%</code></td>
<td>00000010 10000000 00000000 00000110</td>
</tr>
<tr>
<td>2064</td>
<td><code>be done</code></td>
<td>01000000 01000000 01000000 01000000</td>
</tr>
<tr>
<td>2068</td>
<td><code>addcc %r1,-4,%r1</code></td>
<td>10000010 10000000 01111111 11111100</td>
</tr>
<tr>
<td>2072</td>
<td><code>addcc %r1,%r2,%r4</code></td>
<td>10001000 10000000 01000000 00000010</td>
</tr>
<tr>
<td>2076</td>
<td><code>ld %r4,%r5</code></td>
<td>11001010 00000001 00000000 00000000</td>
</tr>
<tr>
<td>2080</td>
<td><code>ba loop</code></td>
<td>00010000 10111111 11111111 11111011</td>
</tr>
<tr>
<td>2084</td>
<td><code>addcc %r3,%r5,%r3</code></td>
<td>10000110 10000000 11000000 00000101</td>
</tr>
<tr>
<td>2088</td>
<td><code>done:</code></td>
<td>10000001 11000011 11100000 00000100</td>
</tr>
<tr>
<td>2092</td>
<td><code>length:</code></td>
<td>00000000 00000000 00000000 00010100</td>
</tr>
<tr>
<td>2096</td>
<td><code>address:</code></td>
<td>00000000 00000000 00001011 10111000</td>
</tr>
<tr>
<td>3000</td>
<td><code>a:</code></td>
<td></td>
</tr>
</tbody>
</table>

(From [http://en.wikipedia.org/wiki/Assembly_language](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.
• 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: \( 3600 \times h + 60 \times m + s = N \)
• 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: $3600 \times h + 60 \times m + s = N$, 
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: $3600 \times h + 60 \times m + s = N,$
  $0 \leq s < 60, \ 0 \leq m < 60$

• 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:
  – $h = \frac{N}{3600}$ (integer division)
  – $m = \frac{N \mod 3600}{60}$ ($\mod$: remainder)
  – $s = N \mod 60$

• Another algorithm:
  – $s = N \mod 60$
  – $x = \frac{N}{60}$
  – $m = x \mod 60$
  – $h = x / 60$

• 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;
}

Introduction to Programming © Dept. CS, UPC
Execution

> decompose_time
3815
1 hours, 3 minutes and 35 seconds

> decompose_time
60
0 hours, 1 minutes and 0 seconds