Introduction to Programming (in C++)

Conclusions

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

Why is programming hard?

- Part of the difficulty is that we have many requirements.
- Our programs must be:
  - Useful
  - Correct
  - Efficient
  - Easy to understand, modify and extend
  - Cheap
  - … and more

Useful programs

Programs must solve the user’s problem,
not do what the programmer knows how to do.
The user must know exactly what the program does.

Specification is the key:

- A contract between the user and the programmer
- Must be unambiguous and complete
- In this course: *pre-condition* + *post-condition*

Rule: Don’t decide in the code what must be decided in the specification

Correct programs

A program is *correct* if it behaves according to its specification on *all* inputs that satisfy the precondition!

Note: “correct” does not even mean anything if we do not have a specification!

Rule:

5 minutes of thinking

=

1 hour of debugging

Correct programs

Good programming methodology helps to create correct programs:

- Start from the <u>specification</u>, not from your idea.
- <u>Divide</u> a complex problem into smaller pieces (procedures and functions). Carefully specify each one.
- Use <u>induction</u> to guide your design of loops and your use of recursion. How do I solve problems of size n, if I could solve problems of size m<n?
- <u>Invariants</u> are a way to express the inductive hypothesis behind a loop.

Correct programs

Rule: work hard to explain why your program is as specified, not what it does line-by-line.

Example: write a program to compute $\lfloor \log_2(n) \rfloor$

```
int n;
cin >> n;
int m = 0;
while (n > 1) { n = n/2; ++m; }
```

"The program reads a number n and then divides it by 2 and increments m until n<=1."

Yes, we can see that. Where's $\log_2$?

Correct programs

Rule: work hard to explain why your program is as specified, not what it does line-by-line

Example: write a program to compute $\lfloor \log_2(n) \rfloor$

```
int n;
cin >> n;
int m = 0;
while (n > 1) { n = n/2; ++m; }
```

"If N is the value read for n, at all times we have $n = N / 2^m$. Therefore, when n=1, $N / 2^m = 1$, so $m = \lfloor \log_2(N) \rfloor$." Ah!

Efficient programs

- A problem may have many correct solutions,
- but some are more efficient than others (in time, memory and communication, for example).
- Choosing the right <u>algorithms and data structures</u> is the key to efficiency.
- To discuss efficiency: consider the time or memory used as a function of the input “size”. See how fast that function grows

Programs are mathematical objects

- Like mathematical formulas.

- One can rigorously prove that they satisfy certain properties:
  - they satisfy a (mathematical) specification
  - they use so much time or memory.

- Incorrect software in critical tasks may cause disasters and loss of human lives.

Easy to understand, modify and extend

Many programs need to be modified because:

- They were incomplete or incorrect: **maintain**
- They are used as a starting point for another program: **reuse**
- We need to add functionalities to them: **extend**

This may be easy or hard, depending on how we wrote the program

Easy to understand, modify and extend

Documentation:

- Comments, pre-/post-conditions, invariants
- Manual, technical specs (for large programs)
- Coding conventions, good naming

Structure:

- Procedures/functions with clear meanings
- No fancy language-dependent constructions

Programming has limits

Not all problems that can be specified can be solved

- Some specifications make no sense

```
// Pre:  none
// Post: write an integer i such that i*i < 0
```

- But there are unsolvable problems for which
  - a well-defined answer always exists
  - yet no algorithm can find the answer every time

- Some problems are intractable, they admit algorithms but only very inefficient ones

Programming has limits

- The string

  `“int main() { cout << “Hello world!” << endl; }”`

  is a C++ program that halts, and the string

  `“int main() { while (true) { } }”`

  is a C++ program that does not halt.
- Consider the specification:

```
// Pre:  string s is a legal (compiling) C++ program
// Post: returns true if the program s halts on the
//       empty input, and false otherwise

bool halts(string s);
```

- Such an algorithm would be very useful
- However, it is a deep result that

THERE IS NO PROGRAM `halts()` to satisfy this specification

Quotes

- “Computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better.” (Donald Knuth)

- “There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.” (C.A.R. Hoare)

- “Controlling complexity is the essence of computer programming.” (Brian Kernigan)

- “The trouble with programmers is that you can never tell what a programmer is doing until it’s too late.” (Seymour Cray)

Quotes

- “The question of whether computers can think is like the question of whether submarines can swim.”
(Edsger W. Dijkstra)

- “Programmers are in a race with the Universe to create bigger and better idiot-proof programs, while the Universe is trying to create bigger and better idiots. So far the Universe is winning.” (Rich Cook)

- “A great lathe operator commands several times the wage of an average lathe operator, but a great writer of software code is worth 10,000 times the price of an average software writer.”
(Bill Gates)