C++ Programming for Engineers

Petar Jervic
Francisco Luro
Andrew Moss (course-resp)

May 2, 2012

Overview

Today is a gentle introduction to the course

- Meet the teachers
- What the course is about
- Why the content is important
- What you should expect from us
- What we expect from you
- Start with the course material...

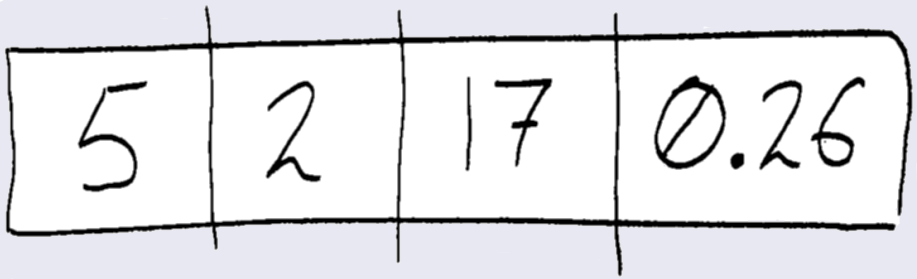
What is programming?

- Being confused is a normal state of affairs
- It is a beginning, not an ending
- Learning to translate confusion into pointed questions
- Describing a solution to a problem

Relation to problem solving

Sample Problem

Which is the largest number?

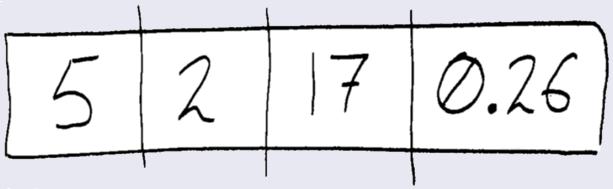


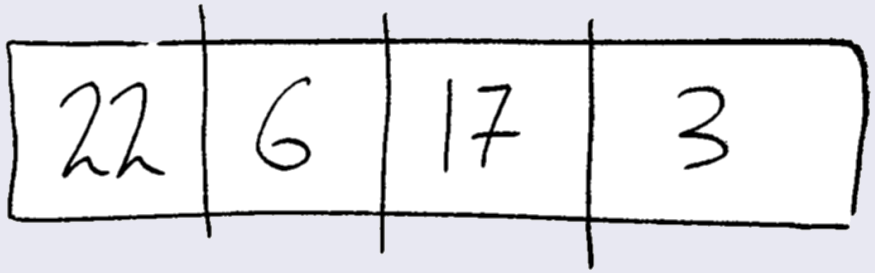
- Hopefully everyone can solve it...
- Expressing *how* to solve it can be a bit trickier
- Need to make some basic assumptions
- Need to make some decisions
- These form the basic steps that we are allowed to make
- Once you know the answer is it still a problem?

Relation to problem solving II

Second Sample Problem

Which is the largest number?



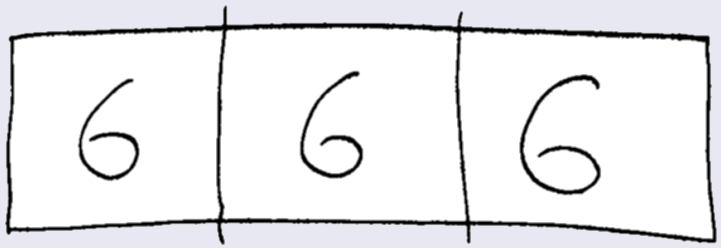


- How is solving this problem similar to the first?
- How is it different from solving the first problem?
- What kind of answers can we express?
  - 1st vs 22

Relation to problem solving III

Last Sample

Which is the largest number?



- Is 6 a valid answer?
- What about 1st?
- How does this relate to our assumptions about the problem?
- How do the solutions to the problems relate to one another?

Types of solutions

There are two kinds of solution to the problem

- A solution to a single instance of the problem
- A general solution to all instances of the problem

In a specific instance of the problem

- There is a particular list of numbers as input
- The solution is a single number
  - The solutions is defined by the particular input
- We can refer to it by value, or by which box it is in
  - Both kinds of solution communicate the same information
- If we represent the solution as a position, then we need the input to decide which number it is
- If we represent the solution as a number then it is sufficient alone

General solutions

Across *all* instances of the problem

- A solution is a method of deciding the answer
- We can describe as a series of steps
- Some kinds of steps are bigger than others, compare
  - Is the number in the first box bigger than the number in the second box?
  - Which box contains the biggest number?
- The bigger the step the more work we do to perform it
  - The more understanding is required from the reader
- Smaller steps are simpler
- Machines can execute very simple steps

A program is series of very small steps of a particular kind

Programming

Programming is an activity that we can see from two viewpoints

- What happens if we combine steps together (bottom-up)
- How can we split a step into simpler pieces (top-down)

To combine existing steps together we need a starting point

- What kinds of steps are valid / understood by a machine?
- One outcome of this course is learning these starting points
  - Expressions, Statements, Variables, Functions . . .
- Start with examples to see what these things are, what they do
- Seeing how to use them comes from practice
  - Impossible to learn programming by reading
  - To understand you must try things

Programming II (top-down)

To learn how to cut problems into simpler pieces requires experience

- Recognise similarities to problems that we have solved before
- Work out how to adapt what we've seen previously to a new setting
- It is hard to learn something that depends on experience

The exercises start from nothing

- Each one adds a small piece of understanding allowing you to build experience
- As they become more complex they rely on these pieces of experience
- It is vital that you do not skip exercises

Relation to EE

Your background knowledge from previous courses will help you learn bottom-up programming

- Boolean logic is essential to programming
- The most complex circuits are the basic steps we combine in programs

The knowledge you learn on this course will complement your EE skills

- Software / Firmware / Co-design are ubiquitous in modern devices
- Programming is a basic technique for decomposing the real world into steps suitable for a machine

Structure of the course

Four activities on the course

- Lecture — Two hours every Monday morning (10:00-12:00)
- Labs — 6-hr of supervised labs, exercises (08:00-10:00)
- Coursework — Twelve hours a week (outside of class)
- Exams — One per LP

This maximises learning activities during contact time

- Class is too big for one lab; three separate groups / teachers
- Lab sessions are mandatory; no credit without attendence

Coursework is to evaluate how much you have learnt.

- Activities in class prepare you for it.
- It must be completed individually outside of class.

3-week cw = 36 hours, 5-weeks is 60 hours, 8-week project is 96 hours.

Structure (cont)

Lecture attendance is optional

- Each week the lecture will cover the same content as the exercises
- The lectures are designed to tell you what you need to know to solve the exercises
- If you choose not to attend then you are responsible for finding what you need to keep up

Notes will be supplied

- These are a reminder of what is said in the lectures, not a replacement
- You are expected to read these notes before attempting the exercises
- The textbook does not follow exactly the same stucture
  - It is a different viewpoint of the same material
  - It is there as a secondary source to support you if you cannot understand something

Required Effort

You must attend the lab sessions

- Attendence is required to pass the course

You must complete the exercises for each week

- We use these to gauge whether or not you are keeping up with the course
- Do not skip ahead

You will need to put substantial effort into the coursework

- This course is designed to fill 50% of your time
- Lab-time is not for you to attempt the coursework
  - It is to get you to the point where you can do the coursework

Learning to Program is an incremental activity

- Each lesson builds on your understanding of the previous lessons
- There are no independent modules within the course

Required Technique

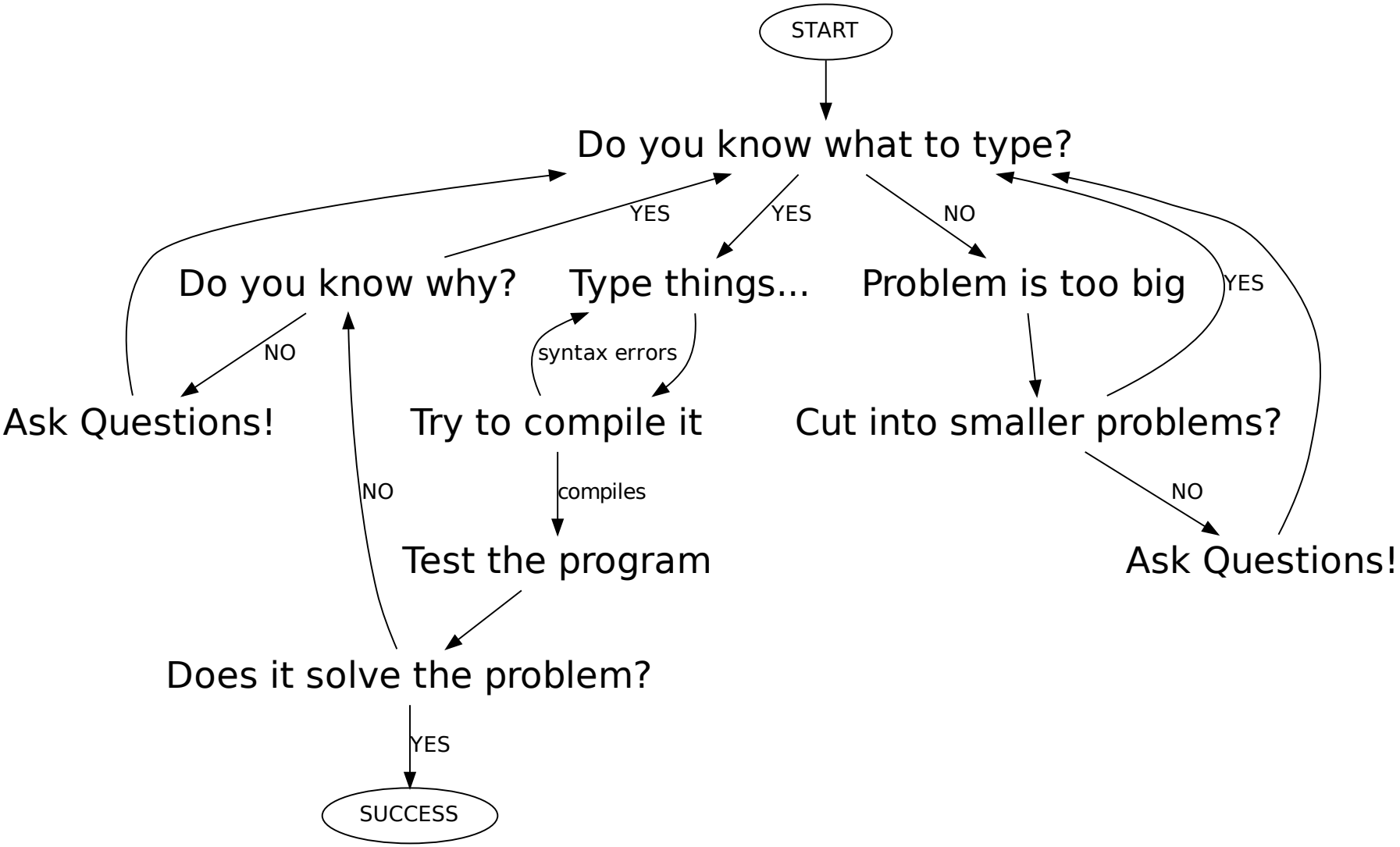
Learning to program is an experimental activity

- You must be willing to make mistakes in order to proceed
- You cannot hold everything necessary in your head and then type in a finished program
- To find out what is necessary you must try the parts that you know

Know a set of possible steps

- Try them
- Validate whether they help
- Repeat

The programming process



Required Technique

Failure is an important part of the process

- Seeing how and where something breaks gives you clues about how to fix it
- Comfort and understanding come from an ability to recover from mistakes
- Learning to play with code is an essential part of learning to program
- You must see the process as game
  - The computer scores points for finding mistakes in your programs
  - You score points for writing programs the computer cannot find faults in
- Learn to beat the machine

Problem I : Deer in headlights

Buridan's camel

A hypothetical camel positioned exactly in the middle of two piles of hay. It dies of hunger because it cannot decide which way to go.

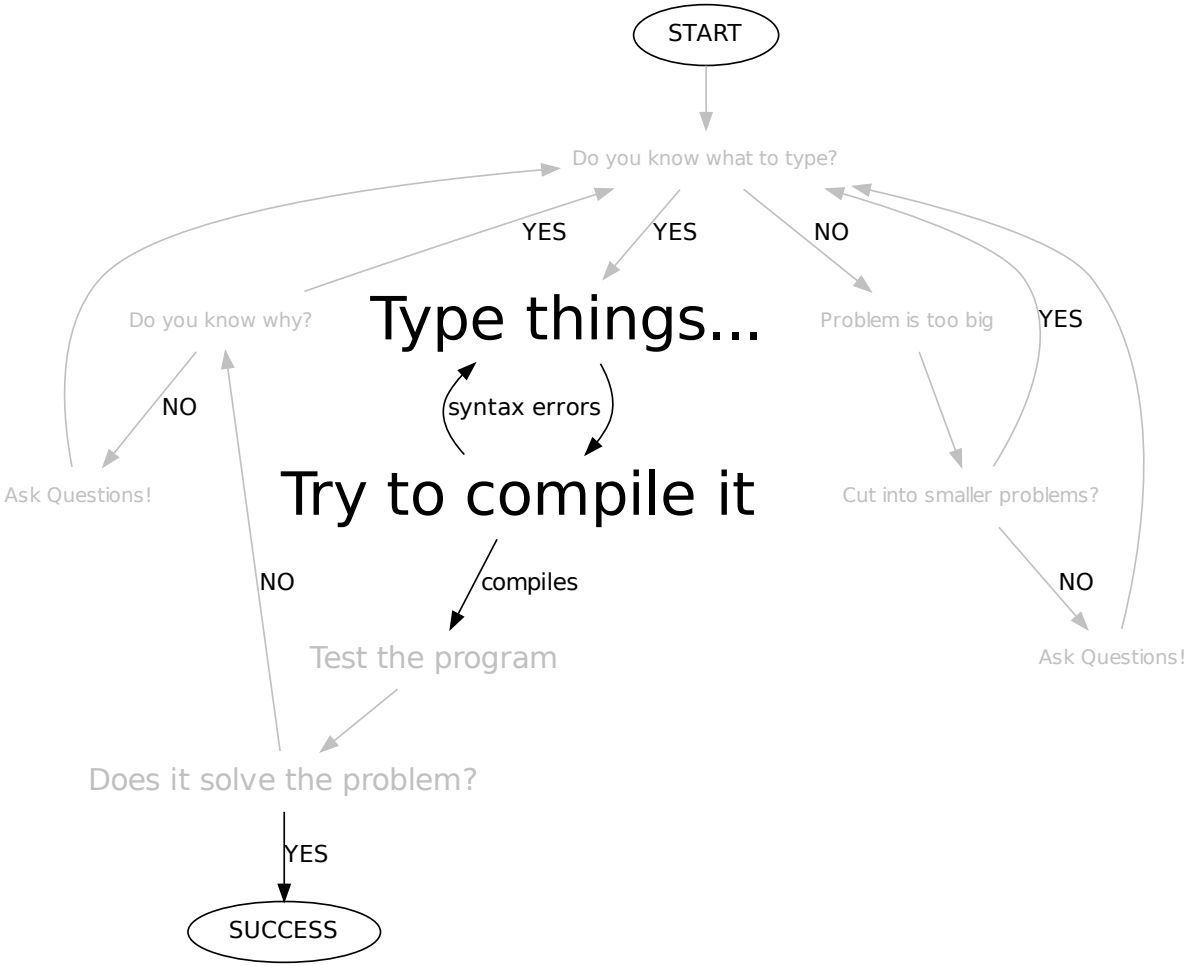
- This is a situation that occurs amongst programming students
- Trying to solve Problem X, they think of solutions A, B, C, ...
- Because they cannot decide which solution is better they freeze
- Panic prevents them from making a decision.

The solution

- If you do not know which solution is better, then pick any solution
- Try it
  - If it works then you do not need the others
  - If it doesn't work then pick another and repeat.

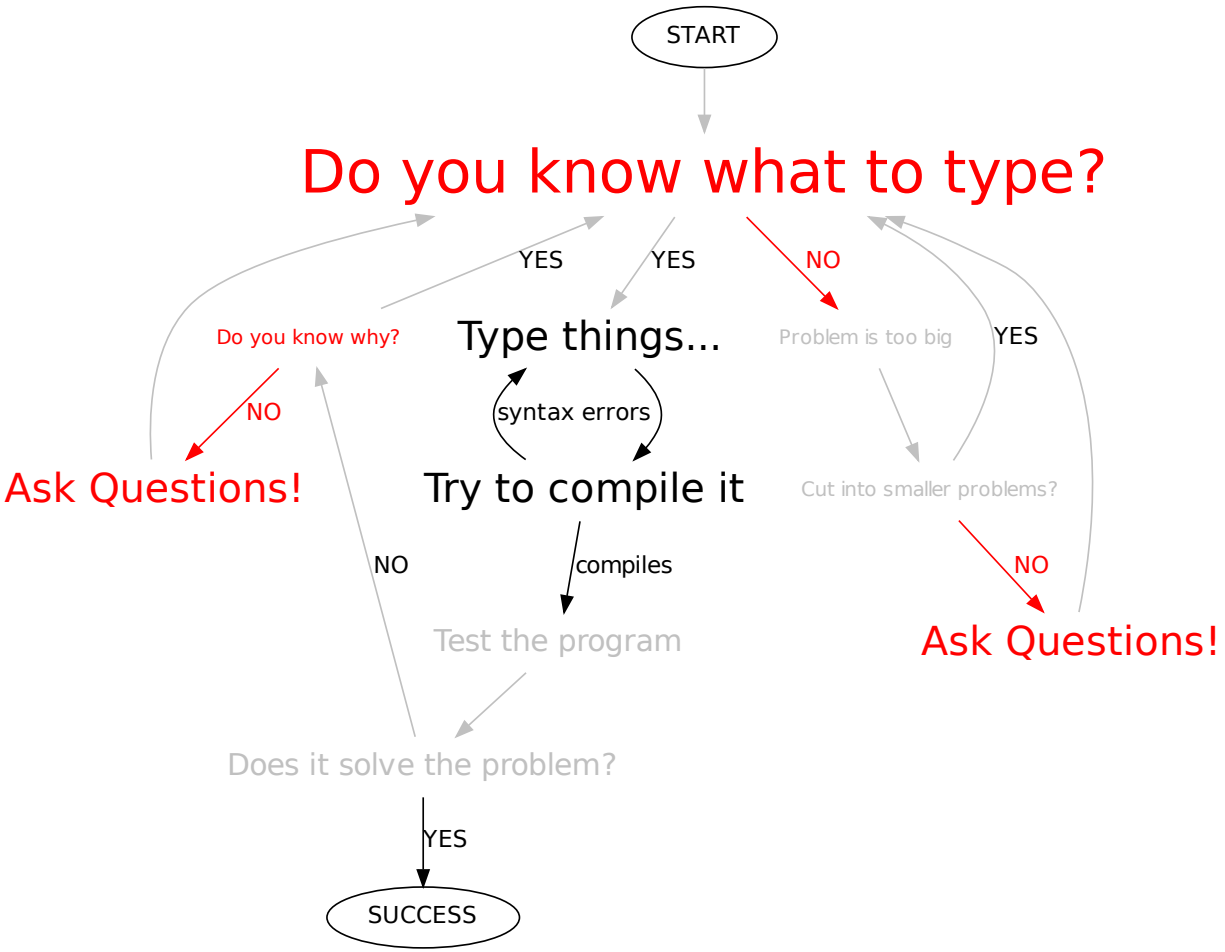
Problem II : No clear solution

- Programming is not just a typing exercise
- Without problem-solving it is impossible to find the answer

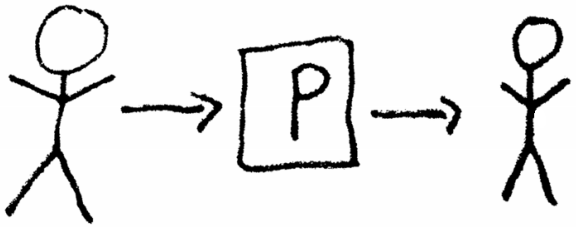


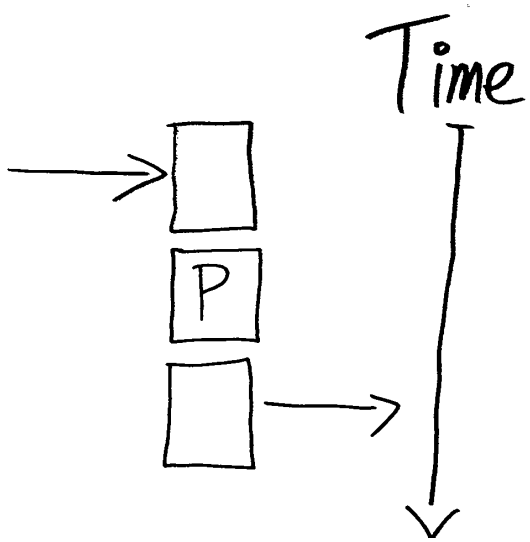
Problem III : Avoiding trial and error

- Getting it wrong feels like failing
  - But when a problem is big enough it takes more than one attempt
- Without experimenting you cannot learn to get it right



Interaction





Classic batch view of a program

- Has some form of input from a user.
- Produces some form of output for the user.
- Some form of processing upon the data inbetween.

This is a sequence of steps

- The input happens before the processing.
- The processing happens before the output

Types of steps

Input

- Takes something from the environment (e.g. keystrokes)
- Produces a value
- Can be stored in a variable

Processing

- Calculates something based on values / variables
- Creates a new value
- Expressions will be covered in week 2.

Output

- Causes something to happen in the environment
- Characters to be displayed on a console

What is a program

Start with a simple definition of a program

- It will become more complex over the course
- Add decisions in week 3, and loops later in the course

A Program

A set of variables for storing data, and a sequence of input / processing / output steps.

- Each of these things is defined in a text file called the source code
- A program called a compiler converts this into an executable (binary)
- Running the binary will then execute the program on the machine

The compiler

First part of the course will use a command-line compiler

- Visual Studio *after* you have learnt the basic syntax and operation
- No intellisense while you are learning syntax
- IDE will be introduced once you understand the basics

Week 1 Labs

The exercises are on the course page

- Read them before you come to the lab.
- The lab sessions are supervised time to work through them.
- Ask questions and get help

All the exercises this week are simple sequential programs

- Need to define a series of steps
- Each of the steps can be input / output / processing
- Produce a program with the behaviour specified in the exercise

Examples

Declaring variables to store data

- `int aname;`
- `char bernard;`
- `string somename;`

Input values into variables

- `cin »somename;`
- `cin » bernard;`
- `cin » aname;`

Output values from variables

- cout « somename;
- cout « bernard;
- cout « aname;