Fundamentals of programming

Part 2

Introduction to a Programming Language

ETSI Informática, Málaga

Prof.: Juan Falgueras
Room: 3.2.32

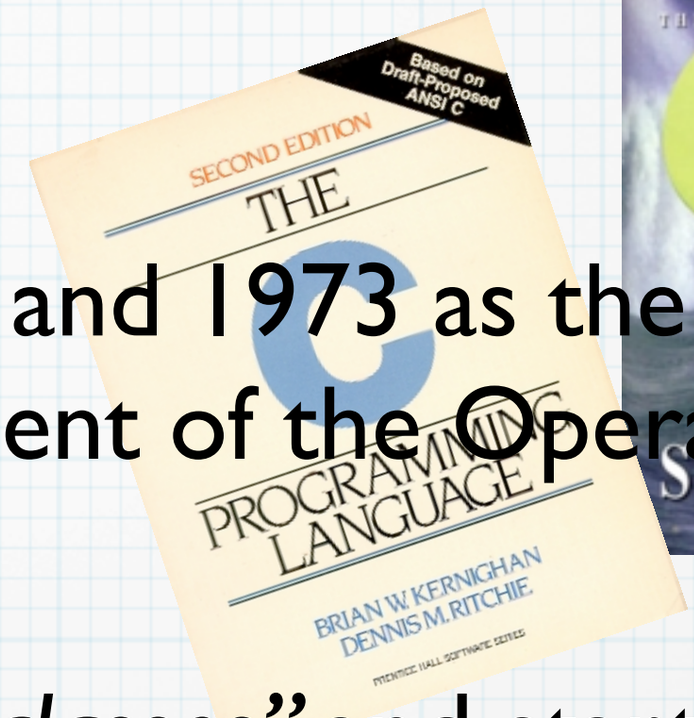
Contents

Contents

2. Introduction to a programming language

Contents

1. Introduction to C++. History



- C was born between 1969 and 1973 as the language of choice for the development of the Operating System UNIX
- C++ was originally "*C with classes*" and started to be developed in 1979
- It is backward compatible with C (a C++ compiler can compile C source as well)
- There are others *Cs*: ObjectiveC, C#, etc.

1) An example of a program in C++

- In C++ programs execution starts in the first instruction in the special function **main()**

```
int main()
{
    // actions
    return 0;
}
```

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

int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

```cpp
// sumupto100.cpp
// juanfc 2012-10-03
// add natural numbers up to 100

#include <iostream>
using namespace std;

int main()
{
    int s, i;
    s = 0;
    i = 1;

    while ( i <= 100 ) {
        s = s + i;
        i = i + 1;
    }
    cout << "Sum: " << s << endl;
    return 0;
}
```

Fixed header

Contents

Contents

2) Simple data types

2) Simple data types

A **data type** is
a set of **values** and a
set of **operations** suitable for being
executed on them

Predefined simple types

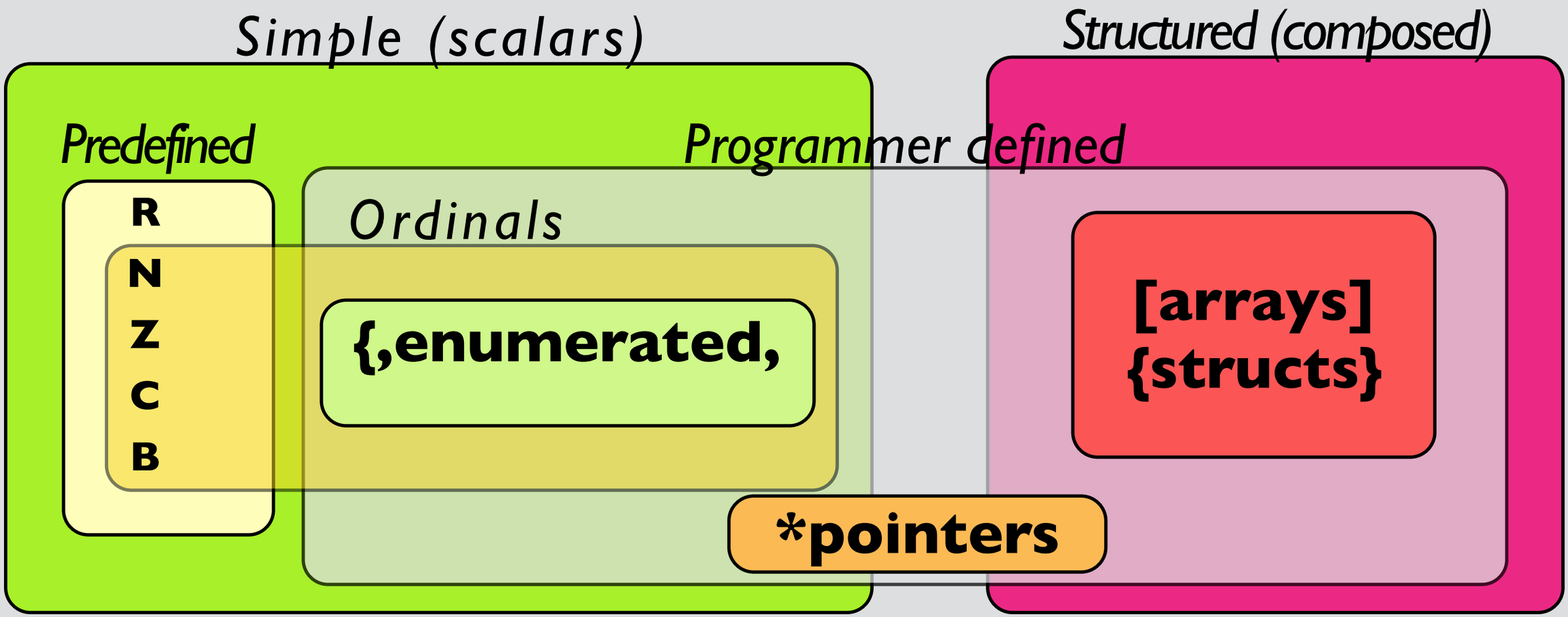
Booleans	*true, false*	**bool**
Chars	'a', 'x', '9', '.'	**char**
Naturals	0,1,2,3,…	**unsigned [long] [int]**
Integers	–3, 0, 1000	**[long] int**
Reals	3.14E–3	**[long long] float/double**

```
        char:  1 bytes
  short int:  2 bytes
         int:  4 bytes
    long int:  8 bytes
       float:  4 bytes
      double:  8 bytes
 long double: 16 bytes
```

sizes and ranges

Tipo	Bytes	min	max
bool	1	false	true
char	1	-128	127
unsigned char	1	0	255
short	2	-32,768	32,767
unsigned short	2	0	65,535
int	4	-2,147,483,648	2,147,483,647
unsigned	4	0	4,294,967,295
long	4	-2,147,483,648	2,147,483,647
long long	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long long	8	0	18,446,744,073,709,551,615
float	4	-3.40282e+38	3.40282e+38
double	8	-1.79769e+308	1.79769e+308
long double	16	-1.18973e+4932	1.18973e+4932

All data types



2) Some predefined simple data types

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

int main()
{
    char               c =           'A';
    short int          s =           125;
    int                i =           -13;
    unsigned int       u =     429496729;
    long int           l =    -229496729;
    unsigned long int ul =    4147483647;
    float              f =        3.1416;
    double             d =    3.1416E100;
    long double        ld = 3.1416E-2000;

    cout << c << ", " << s << ", " << i << ", "
         << u << ", " << l << ", " << ul << ", "
         << f << ", " << d << ", " << l << ", " << d;
    return 0;
}
```

literal constants

- `3` *int*
- `'3'` *char*
- `"3"` *string with only one letter*
- `'x'` *char*
- `"hola"` *string of letters*
- `'\n'` *control char: equivalent to* **endl**
- `"\tInput: 'main value'"`
- `“Greeted \"Hi\" when entering"`
- `'\''`
- `'"'`

2) Variables

- Variables are to store mutable or modifiable data
- Variables have type (int, float…)
- Some variables are called **const**ants and are to not changeable during the execution of the program
- To create a variable:

```
int nameOfTheVariable;
float h, r, S;
int i = 0;
```

2) Predefined simple data types

- values

int i = 0;

- from −2,147,483,648 to +2,147,483,647

char c = 'M';

'a', 'x', '9', ':'.....

float a, b, c;

- until 3.40282e+38; long double: 1.18973e+4932

bool isFound = false;

- false, true

2) Sizes of... memory that each type occupies

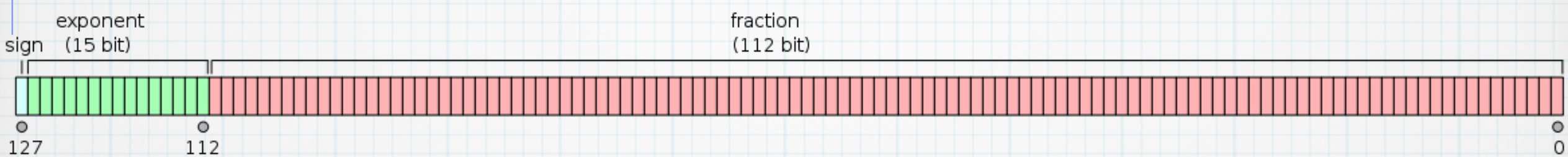
```
// sizesof.cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Sizes in bytes."                                    << endl;
    cout << "The size of a char is:        " << sizeof(char)   << endl;
    cout << "The size of a short int is:   " << sizeof(short)  << endl;
    cout << "The size of an int is:        " << sizeof(int)    << endl;
    cout << "The size of a long int is:    " << sizeof(long)   << endl;
    cout << "The size of a float is:       " << sizeof(float)  << endl;
    cout << "The size of a double is:      " << sizeof(double) << endl;
    cout << "The size of a long double is: " << sizeof(long double)
                                                              << endl;
    return 0;
}
     // The size of a char is:        1 bytes.
     // The size of a short int is:   2 bytes.
     // The size of an int is:        4 bytes.
     // The size of a long int is:    8 bytes.
     // The size of a float is:       4 bytes.
     // The size of a double is:      8 bytes.
     // The size of a long double is: 16 bytes.
```

2) Sizes of... and ranges..., existing constants

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

int main()
{
    cout << USHRT_MAX << endl;  //  65535
    cout << SHRT_MIN << endl;   // -32768
    cout << SHRT_MIN << endl;   //  32767
    cout << INT_MIN << endl;    // -2147483648
    cout << INT_MAX << endl;    //  2147483647
    cout << ULONG_MAX << endl;  // 18446744073709551615
    cout << FLT_MAX << endl;    // 3.40282e+38
    cout << DBL_MAX << endl;    // 1.79769e+308
    cout << LDBL_MAX << endl;   // 1.18973e+4932
    return 0;
}
```

2) Reals… the most complex ones (IEEE 754 Quadruple Floating Point Format)

16 bytes




```
3fff 0000 0000 0000 0000 0000 0000 0000
                                    = 1
```

```
7ffe ffff ffff ffff ffff ffff ffff ffff
 ≈  1.189731495357231765085759326628007 × 10^4932
                        (max quadruple precision)
```

2) Constants

- A constant is a kind of "variable" that does not change
- At the same time they are declared, they must receive their value:

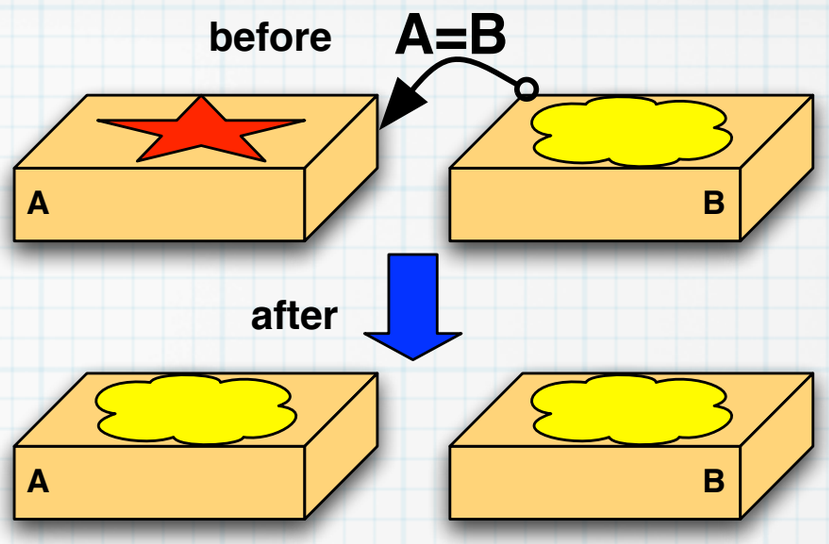
```
const float PI = 3.14;
const char END = 'Q';
const int MAXSIZE = 101;
```

don't forget the type

2) Assignment

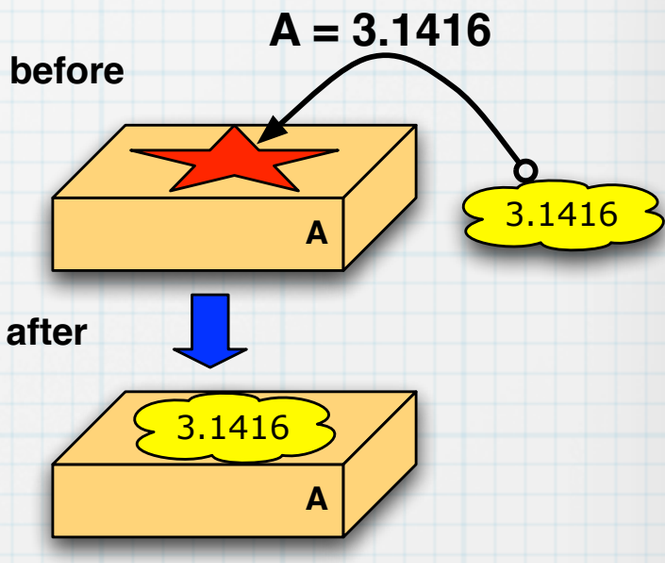
What's assigning

- **LHS ← RHS**



variable = value;

y = 2 + 3 * x;



- Is the most important operation in programming
- The Right-Hand-Side is **evaluated** and the result is ***copied*** onto the variable (simple variable) at the Left-Hand-Side

CHARs

2) Chars and char operators

Tabla ASCII

Dec	Octal	Hex	Binary	Value
000	000	000	00000000	NUL (Null char.)
001	001	001	00000001	SOH (Start of Header)
002	002	002	00000010	STX (Start of Text)
003	003	003	00000011	ETX (End of Text)
004	004	004	00000100	EOT (End of Transmission)
005	005	005	00000101	ENQ (Enquiry)
006	006	006	00000110	ACK (Acknowledgment)
007	007	007	00000111	BEL (Bell)
008	010	008	00001000	BS (Backspace)
009	011	009	00001001	HT (Horizontal Tab)
010	012	00A	00001010	LF (Line Feed)
011	013	00B	00001011	VT (Vertical Tab)
012	014	00C	00001100	FF (Form Feed)
013	015	00D	00001101	CR (Carriage Return)
014	016	00E	00001110	SO (Shift Out)
015	017	00F	00001111	SI (Shift In)
016	020	010	00010000	DLE (Data Link Escape)
017	021	011	00010001	DC1 (XON) (Device Control 1)
018	022	012	00010010	DC2 (Device Control 2)
019	023	013	00010011	DC3 (XOFF)(Device Control 3)
020	024	014	00010100	DC4 (Device Control 4)
021	025	015	00010101	NAK (Negative Acknowledgement)
022	026	016	00010110	SYN (Synchronous Idle)
023	027	017	00010111	ETB (End of Trans. Block)
024	030	018	00011000	CAN (Cancel)
025	031	019	00011001	EM (End of Medium)
026	032	01A	00011010	SUB (Substitute)
027	033	01B	00011011	ESC (Escape)
028	034	01C	00011100	FS (File Separator)
029	035	01D	00011101	GS (Group Separator)
030	036	01E	00011110	RS (Request to Send)(Record Separator)
031	037	01F	00011111	US (Unit Separator)

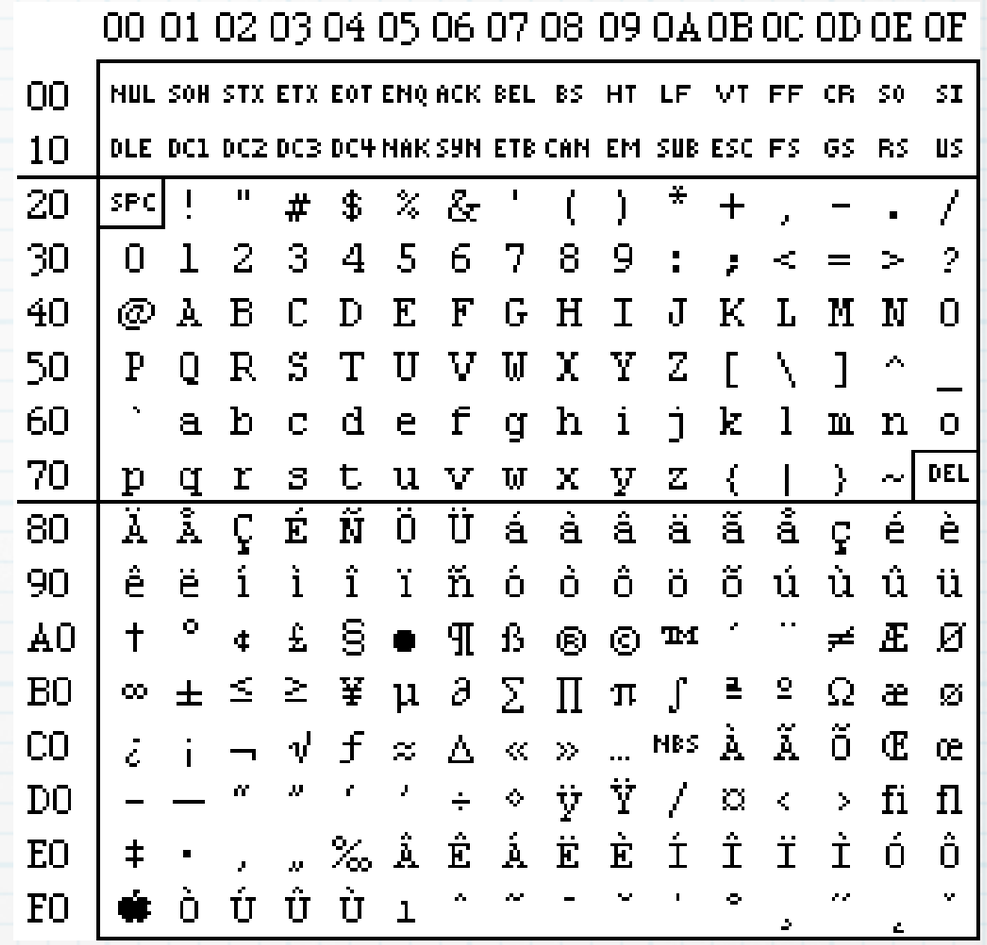
de control
(no visibles)

Ejemplo

"	H	o	l	a	"	\0
	72	111	108	97		0

+32

Dec	Octal	Hex	Binary	Value	Dec	Octal	Hex	Binary	Value	Dec	Octal	Hex	Binary	Value
032	040	020	00100000	SP (Space)	065	101	041	01000001	A	097	141	061	01100001	a
033	041	021	00100001	!	066	102	042	01000010	B	098	142	062	01100010	b
034	042	022	00100010	"	067	103	043	01000011	C	099	143	063	01100011	c
035	043	023	00100011	#	068	104	044	01000100	D	100	144	064	01100100	d
036	044	024	00100100	$	069	105	045	01000101	E	101	145	065	01100101	e
037	045	025	00100101	%	070	106	046	01000110	F	102	146	066	01100110	f
038	046	026	00100110	&	071	107	047	01000111	G	103	147	067	01100111	g
039	047	027	00100111	'	072	110	048	01001000	H	104	150	068	01101000	h
040	050	028	00101000	(	073	111	049	01001001	I	105	151	069	01101001	i
041	051	029	00101001	)	074	112	04A	01001010	J	106	152	06A	01101010	j
042	052	02A	00101010	*	075	113	04B	01001011	K	107	153	06B	01101011	k
043	053	02B	00101011	+	076	114	04C	01001100	L	108	154	06C	01101100	l
044	054	02C	00101100	,	077	115	04D	01001101	M	109	155	06D	01101101	m
045	055	02D	00101101	-	078	116	04E	01001110	N	110	156	06E	01101110	n
046	056	02E	00101110	.	079	117	04F	01001111	O	111	157	06F	01101111	o
047	057	02F	00101111	/	080	120	050	01010000	P	112	160	070	01110000	p
048	060	030	00110000	0	081	121	051	01010001	Q	113	161	071	01110001	q
049	061	031	00110001	1	082	122	052	01010010	R	114	162	072	01110010	r
050	062	032	00110010	2	083	123	053	01010011	S	115	163	073	01110011	s
051	063	033	00110011	3	084	124	054	01010100	T	116	164	074	01110100	t
052	064	034	00110100	4	085	125	055	01010101	U	117	165	075	01110101	u
053	065	035	00110101	5	086	126	056	01010110	V	118	166	076	01110110	v
054	066	036	00110110	6	087	127	057	01010111	W	119	167	077	01110111	w
055	067	037	00110111	7	088	130	058	01011000	X	120	170	078	01111000	x
056	070	038	00111000	8	089	131	059	01011001	Y	121	171	079	01111001	y
057	071	039	00111001	9	090	132	05A	01011010	Z	122	172	07A	01111010	z
058	072	03A	00111010	:	091	133	05B	01011011	[	123	173	07B	01111011	{
059	073	03B	00111011	;	092	134	05C	01011100	\	124	174	07C	01111100	|
060	074	03C	00111100	<	093	135	05D	01011101	]	125	175	07D	01111101	}
061	075	03D	00111101	=	094	136	05E	01011110	^	126	176	07E	01111110	~
062	076	03E	00111110	>	095	137	05F	01011111	_	127	177	07F	01111111	
063	077	03F	00111111	?	096	140	060	01100000	`					
064	100	040	01000000	@										



- They can store letters
- Each char variable can hold a single letter and only those from the ASCII table

2) Inputting chars

```
char c = 'M';
cout << "-> " << c << endl;
cout << "Enter three letters: " << endl;
cin >> c;
cout << "-> " << c << endl;
c = cin.get();
cout << "-> " << c << endl;
cin.get(c);
cout << "-> " << c << endl;
```

2) Char Operators

- There are few operators for chars, assuming `char c; int i;`
  - `int(c)`
  - `char(100)`
  - `char(i+1)`
  - `char(i-1)`

2) Special char constants

- There are some char constants impossible to write without using some special notation: backslash escaping:

```
c = '\n';
c = '\r';
c = '\t';
c = '\0';
c = '\\';
c = '\'';
```

- Can be used inside strings…

```
string s = "Hi, \tyou\nand you";
```

numbers

```cpp
// circle.cpp
#include <iostream>
using namespace std;

const float PI = 3.141592;

int main()
{
    float radio; // or float radio, area;
    float area;

    cout << "Enter the circle radio: ";
    cin >> radio;
    area = PI * radio * radio;

    cout << "The area of the circle is "
         << area << endl;

    return 0;
}
```

2) Operators

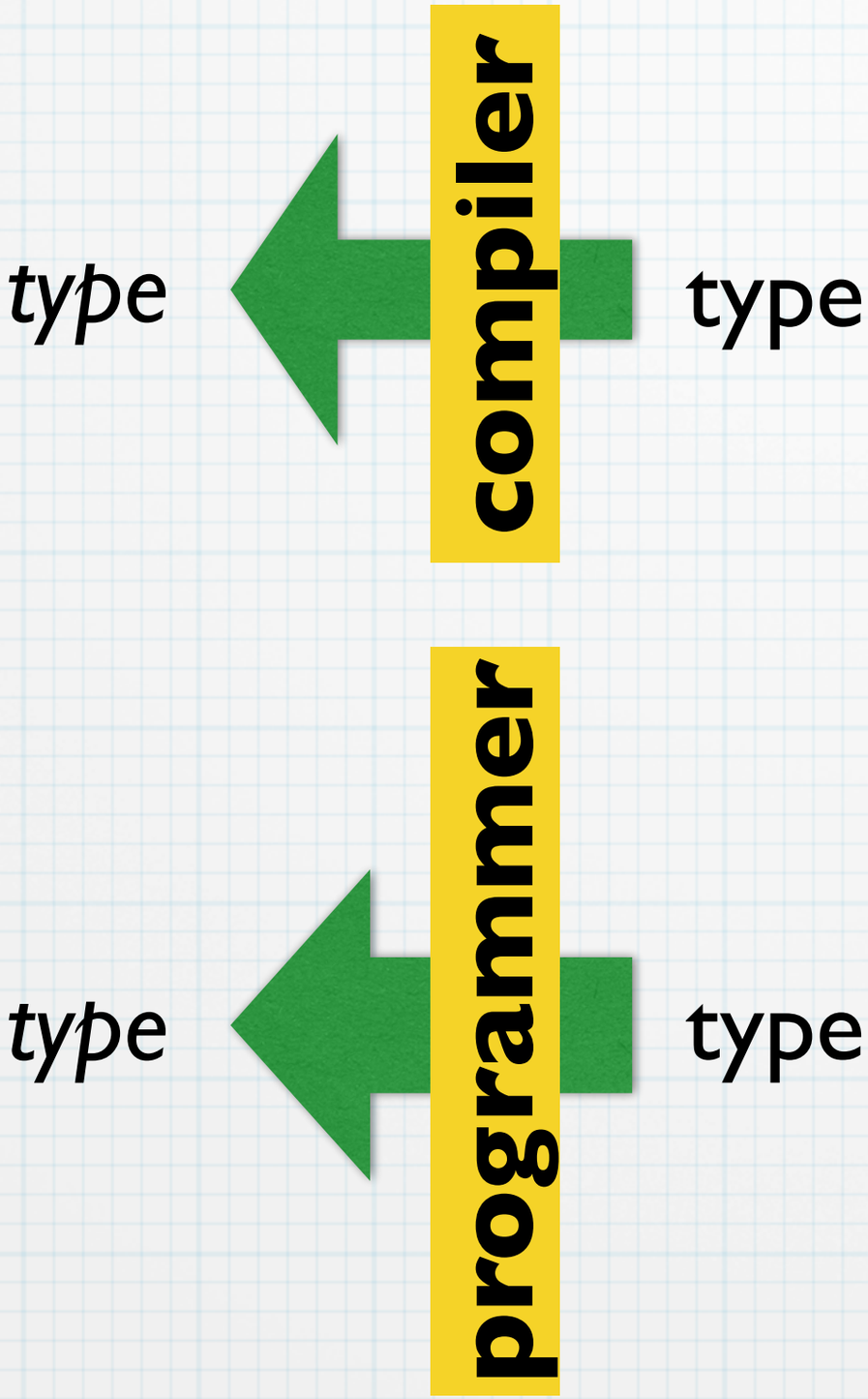
- Each data type admits different operators
- ints and floats admit basic arithmetic operators
  - + – * /

    7 / 3 yields 2 as is always an integer result
  - ints have another op that is called **remainder** or **modulo**: **%**

    `7 % 3 → 1`

    - *How to know if a number is even?*
    - *What's the last digit of an integer?*

2) Type conversions



- **Implicit**

```
i = 3.2 * 5;
```

- **Explicit** adaptation (*casting*)

```
r = (float)3 / 5;
```

2) Type conversions

- Some conversions do not pose any problem but if
  - The destination is smaller than the origin or
  - The destination is simpler than the origin

Truncation or degeneration may occur

2) Type conversions

- Examples:

```
cout << "Res: " << (unsigned) -3 << endl
```

yields

```
Res: 4294967293
```

```
cout << "Res: " << (int) 3.99 << endl
```

yields

```
Res: 3
```

2) Implicit conversions. Occur before expression evaluation

long double

↑

double

↑

float

unsigned long int

↑

long int

↑

unsigned int

↑*

int

2) Type conversions

```
1  char c;
2  short int s;
3  int i;
4  unsigned int u;
5  long int l;
6  unsigned long int ul;
7  float f;
8  double d;
9  long double ld;
10
11  i = i  +  c;  // c -> int
12  i = i  +  s;  // s -> int
13  u = u  +  i;  // i -> unsigned int
14  l = l  +  u;  // u -> long int
15 ul = ul +  l;  // l -> unsigned long int
16  f = f  + ul;  // ul -> float
17  d = d  +  f;  // f -> double
18 ld = ld +  d;  // d -> long double
```

bools

Introduction to `bool`

Boolean **expressions** allow to express conditions and complex logical situations

Boolean **variables** help to name, remember and handle these situations

Arithmetic expressions

$$2 + 2 \rightarrow 4$$

Booleans expressions

$$6 > 5 \rightarrow \textbf{true}$$

Is an expression that results in a boolean value

Booleans expressions

Any of

- A boolean constant (**true**, or **false**)
- A boolean variable
- A expression involving relationships >, <, etc.
- **and, or**, between boolean expressions
- Negation (**not**) of a boolean expression

Supposing

```
int x;
char c;
```

Build an expression that is true when

Supposing

```
int x;
char c;
```

Build an expression that is true when

1. x is greater than 5

Supposing

```
int x;
char c;
```

Build an expression that is true when

1. x is greater than 5
2. x has two digits

Supposing

```
int x;
char c;
```

Build an expression that is true when

1. x is greater than 5
2. x has two digits
3. c is an uppercase letter

Supposing

```
int x;
char c;
```

Build an expression that is true when

1. x is greater than 5
2. x has two digits
3. c is an uppercase letter
4. c is a letter

Supposing

```
int x;
char c;
```

Build an expression that is true when

1. x is greater than 5
2. x has two digits
3. c is an uppercase letter
4. c is a letter
5. c is a vowel

Supposing

```
int x;
char c;
```

Build an expression that is true when

1. x is greater than 5
2. x has two digits
3. c is an uppercase letter
4. c is a letter
5. c is a vowel
6. x is odd

Supposing

```
int x;
char c;
```

Build an expression that is true when

1. x is greater than 5
2. x has two digits
3. c is an uppercase letter
4. c is a letter
5. c is a vowel
6. x is odd
7. x ends in 0

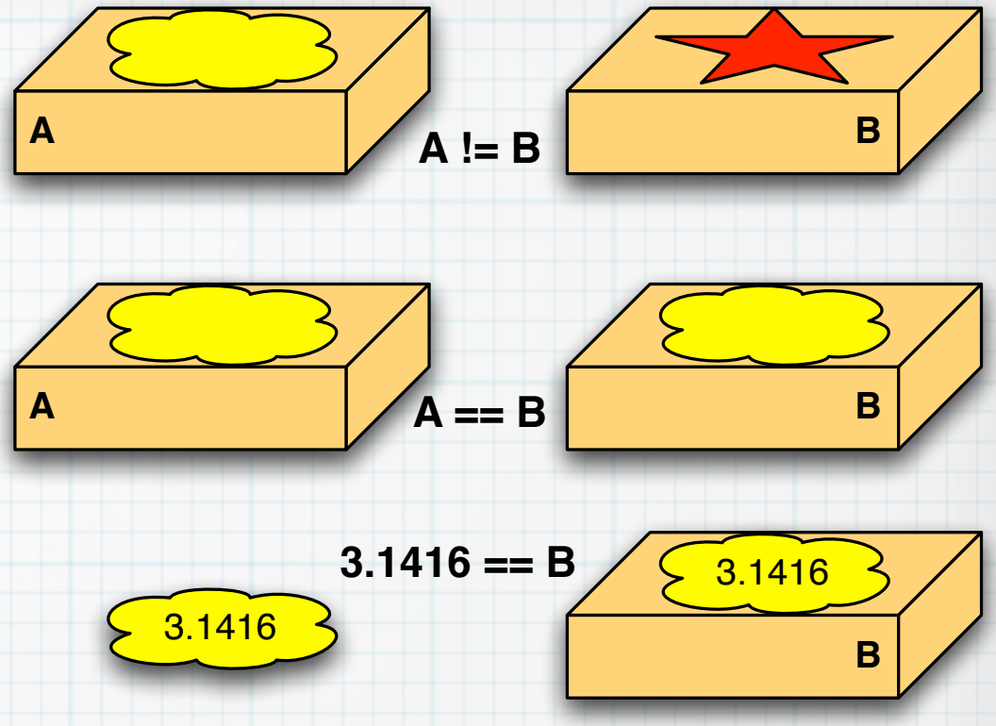
Supposing

```
int x;
char c;
```

Build an expression that is true when

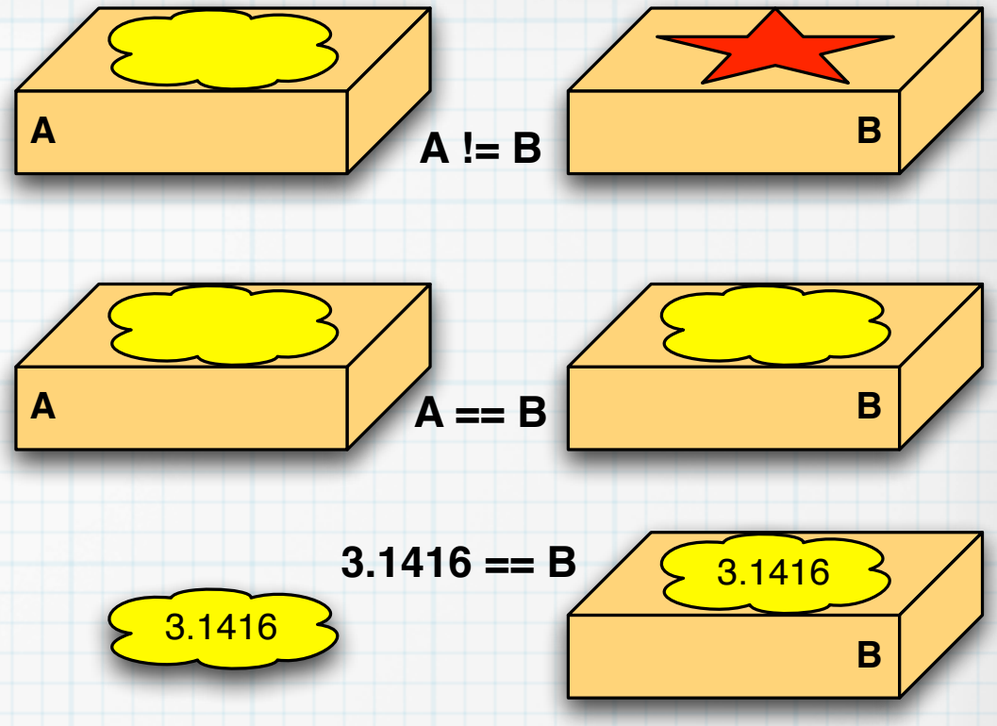
1. x is greater than 5
2. x has two digits
3. c is an uppercase letter
4. c is a letter
5. c is a vowel
6. x is odd
7. x ends in 0
8. x ends in 00

2) Logical Operators



exercise, have a try!

2) Logical Operators



- Some expressions yield bools: (with a, b simple types)

```
a>b,  a<=b,  a==b,  a!=b
```

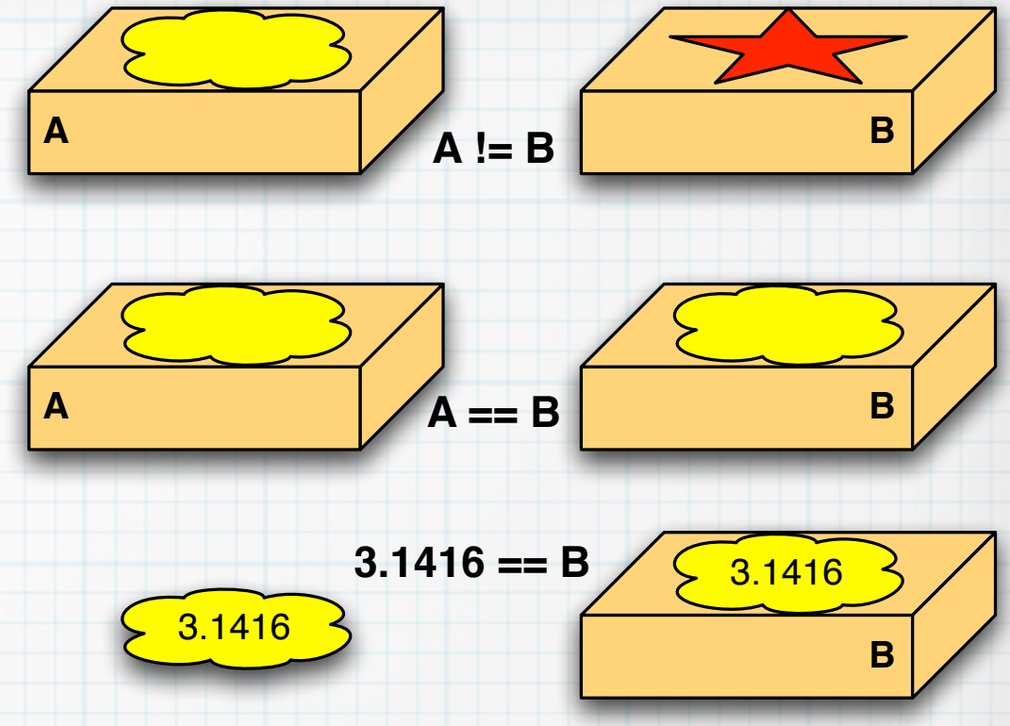
- Logical values can be operated:

```
bool itsRaining, itsHot, itsOk, isLetter;

itsOk = not itsRaining and not itsHot;

isLetter =
```

exercise, have a try!

```
eqNull = a==0 and b==0 and c==0;
```

2) Logical Operators



- Some expressions yield bools: (with a, b simple types)

`a>b, a<=b, a==b, a!=b`

- Logical values can be operated:

```
bool itsRaining, itsHot, itsOk, isLetter;

itsOk = not itsRaining and not itsHot;

isLetter = (c >= 'a') and (c<= 'z') or
           (c >= 'A') and (c<= 'Z');

eqNull = a==0 and b==0 and c==0;
```

2) Logical Operators

The negation of a conjunction is the disjunction of the negations.

The negation of a disjunction is the conjunction of the negations.

2) Logical Operators

The negation of a conjunction is the disjunction of the negations

The negation of a disjunction is the conjunction of the negations.

De Morgan's Laws

- **not** (P **and** Q) ≡ (**not** P) **or** (**not** Q)
- **not** (P **or** Q) ≡ (**not** P) **and** (**not** Q)

Examples:

- `not ((A==B) or (A==C))` ➜ `(A != B) and (A != C)`
- `not ((A==B) and (C>D))` ➜ `(A != B) or (C <= D)`

typedefs

2) Programmer defined simple data types

- New types, based on known ones, can be defined

```
typedef int TCounter;
```

2) Programmer defined simple data types

- They are useful for:
  - Simplifying of giving more specific names to the types
  - Making easier to change many types at the same time
  - To express yet the types are sometimes equal, they contain different things

2) Programmer defined simple data types

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

// typedefs
typedef unsigned short int TUshort;

int main()
{
    TUshort width = 5;
    TUshort length;

    length = 10;
    TUshort area  = width * length;

    cout << "width:  " << width  << endl;
    cout << "length: " << length << endl;
    cout << "area:   " << area   << endl;
}
```

2) Programmer defined simple data types

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

typedef unsigned TBase;
typedef TBase TArea;
typedef TBase TVolume;
typedef TBase TLength;

int main()
{
    TLength a, b, c;
    TArea s1, s2;
    TVolume v;
    // …
    return 0;
}
```

Contents

Contents

5) Control flow

5) Control flow

S;e;q;u;e;n;c;e;

C^{h}oi^{c}e

Loop

- Control structures are based on conditions
- A condition establishes (being **true** or **false**) if something will happen or not
- Handling boolean conditions is first stage in commanding control structures

- Exercices:

1. Write a condition to know if a number is even
   - Answer: `bool isEven = n % 2 == 0;`
2. Write a condition to know if a number is between 10 and 100 (both excluded)
3. Write a condition to know if a char c contains a letter

Fundamentals of Programming. ETSII

1st list of exercises

UMA - ETSIS

II. Variables and logic expressions

1. Define in C language the next constants:
   `MAXCHAR` valued 256; `PI` as 3.1416; `ENDOFLINE` char with a value '`\n`'

2. Given an `unsigned num`, write an boolean expression that reveals if `num` is an even number or not.

3. Given an `unsigned num`, write an boolean expression that reveals if `num` is a number with three digits or not.

4. Given an `unsigned num`, write an boolean expression that reveals if `num` divides 100 or not.

5. Write a boolean expression to show next *belonging* relationship: $x \in \{3, 4, 5, 6, 7\}$.

6. Write a boolean expression to show next *belonging* relationship: $x \in \{1, 2, 3, 7, 8, 9\}$.

7. Write a boolean expression to show next *belonging* relationships: $x \in \{3, 4, 6, 8, 9\}$, and at the same time, $y \in \{6, 7, 8, 3\}$.

8. Given the variables `unsigned x,y`, write an boolean expression to reveal if neither `x` nor `y` are greater than 10 or they are.

9. Given the variable `char c`, write an boolean expression to reveal if `c` is an uppercase letter or not.

10. Given the variable `char c`, write an boolean expression to reveal if `c` is any letter (upper or lowercase).

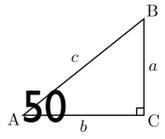
11. Given the variable `char c`, write an boolean expression to reveal if `c` is or not is any kind of vowel, lower or upper vowel.

12. In the next C++ program

```
#include <iostream>
using namespace std;
const float PI = 3.141592;
int main()
{
    float radio;
    float area;
    cout << "Enter the circle radio: ";
    cin >> radio;
    area = PI * radio * radio;
    cout << "The surface of a circle with radio " << radio
         << " is " << area << endl;
    return 0;
}
```

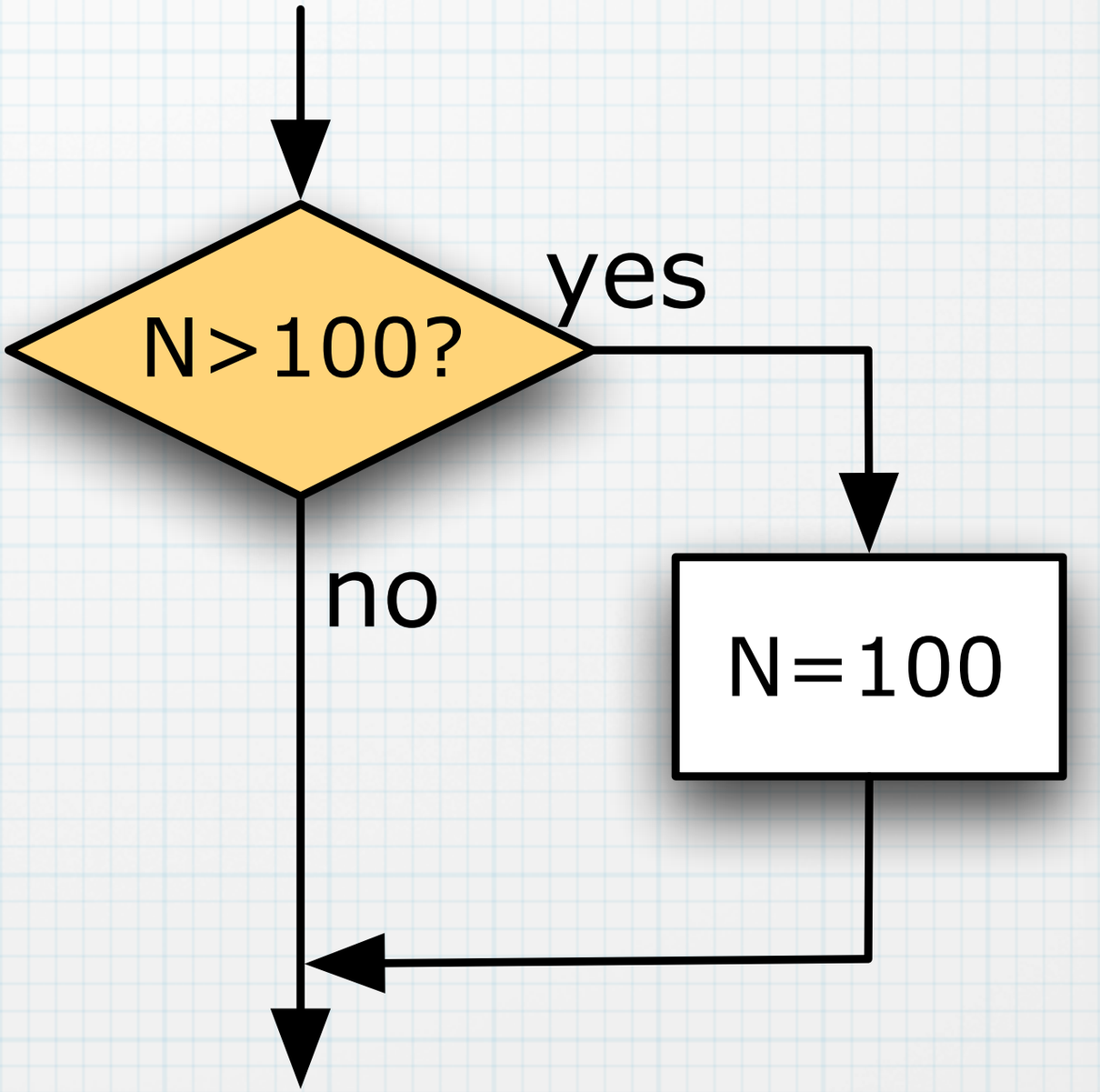
    make the needed modifications so as to the program asks for the height of a cilinder and its radio and then shows its volume ($V = S_b \times h$)

13. Considering a right triangle like the one in next figure. Build a program that asks for the length of the two sides a, b that form the right angle and computes and shows on the screen its hypotenuse c, but rounding its value to the next integer. In order to use the square root function (`sqrt(value)`), the `cmath` file must be also included: `#include <cmath>`



Simple alternative

```
if (N > 100) {
  N = 100;
}
```



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

int main()
{
    int a;
    cin >> a;

    if (a % 2 == 0) {
        cout << "The number is even" << endl;
    }
    return 0;
}
```

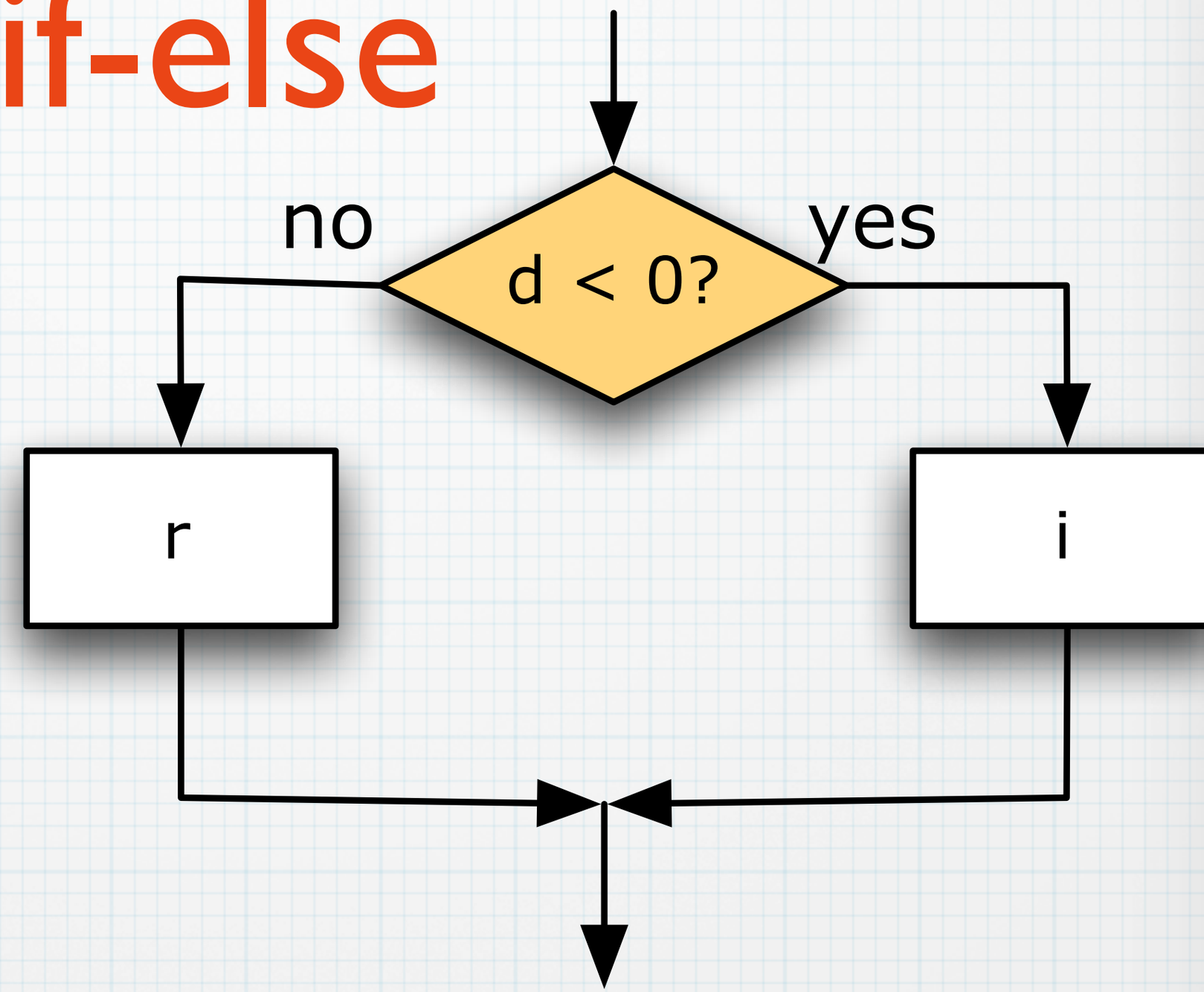
- Exercices:

1. Write a program that asks the user for a number and tells they whether the number is odd

2. Write a program that reads a letter and convert (and print it) in uppercase. To change a letter to uppercase you can do: c + ('A'-'a');

if-else

```
if (d < 0) {
   i;
} else {
   r;
}
```



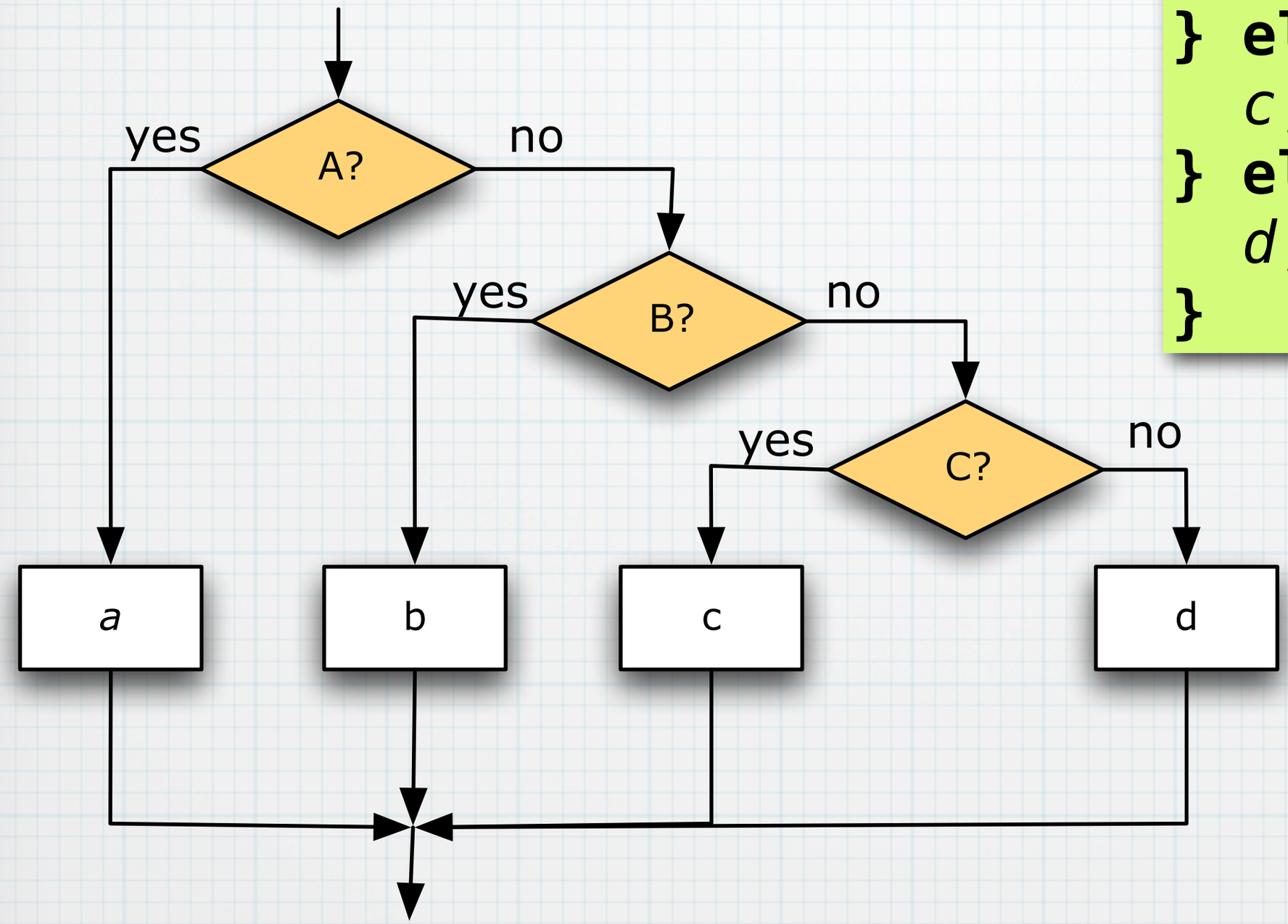
- Exercice:

1. Write a program that asks the user for a number and tells they whether the number is odd or not (use an **if-else** structure)…

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

int main()
{
    int a;
    cin >> a;

    if (a % 2 == 0) {
        cout << "The number is even" << endl;
    } else {
        cout << "The number is odd" << endl;
    }
    return 0;
}
```

Concatenated

```
if (A) {
   a;
} else if (B) {
   b;
} else if (C) {
   c;
} else {
   d;
}
```

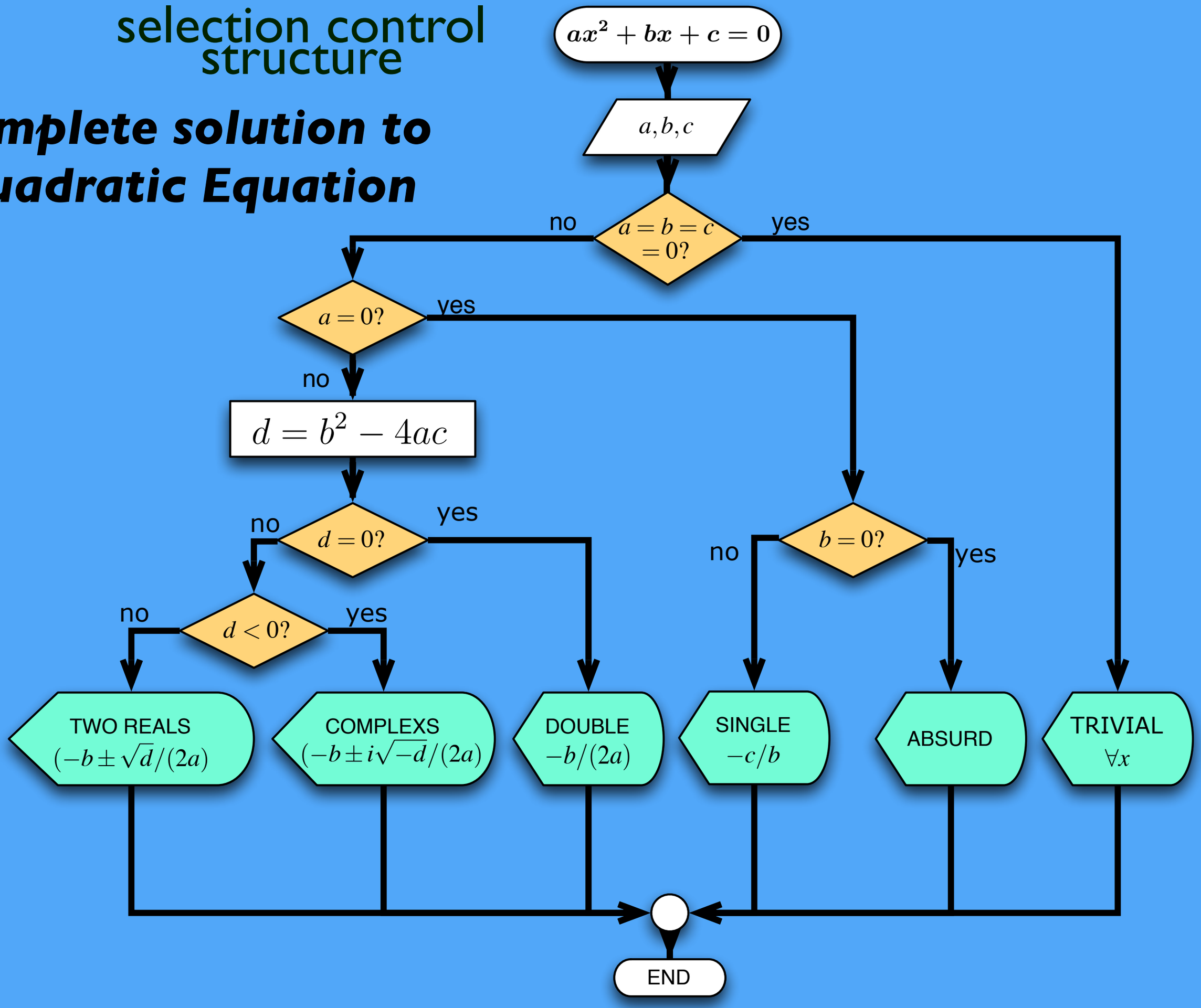


Is like a sieve

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

int main()
{
    int a;
    cin >> a;

    if (a < 5) {
        cout << "Suspenso (Fail)" << endl;
    } else if (a < 7) {
        cout << "Aprobado (Pass)" << endl;
    } else if (a < 9) {
        cout << "Notable" << endl;
    } else {
        cout << "Sobresaliente" << endl;
    }
    return 0;
}
```

selection control structure

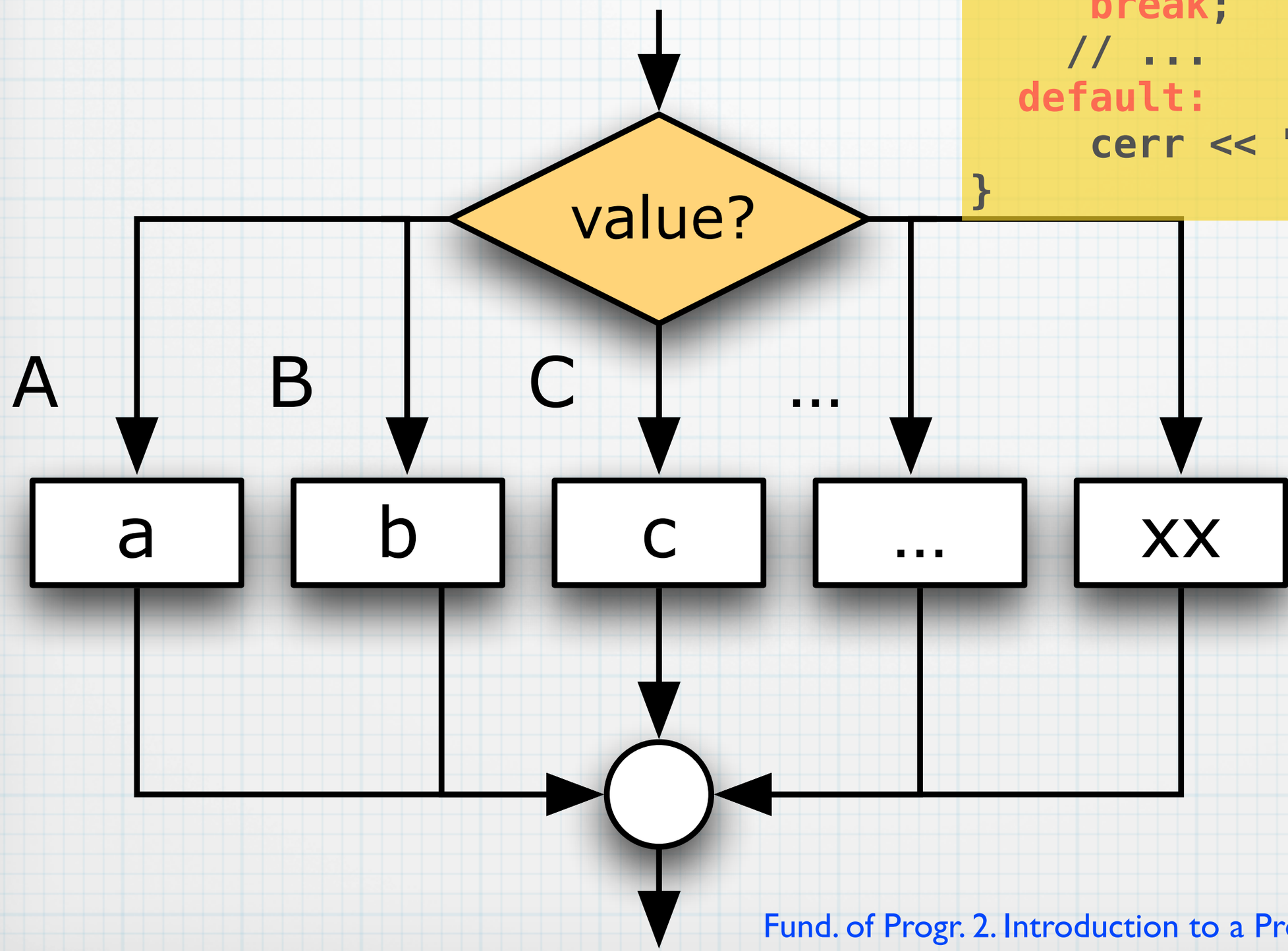
Complete solution to Quadratic Equation



switches

Multiple choice

```
switch (achar) {
  case 's':
     y = sin(x);
     break;
  case 'c':
     y = cos(x);
     break;
    // ...
  default:
     cerr << "Error" << endl;
}
```



Multiple choice

```
switch (achar) {
  case 's':
      y = sin(x);
      break;
  case 'c':
      y = cos(x);
      break;
    // ...
  default:
      cerr << "Error" << endl;
}
```

```
int selection;
cout << "Choose an option: ";
cin >> selection;

switch (selection) {
 case 0:
   return 0; // THE END
 case 1:
   // do whatever things are necessary
 break;
 case 2:
   // do whatever things are necessary
 break;
}
```

Ask for 2 numbers and an operation

```
float a, b, r;
char op;

cout << "Enter 2 numbers: " << endl;
cin >> a >> b;
cout << "Operation: ";
cin >> op;
```

```
switch (op) {
 case '+':
    r = a+b;
    break;
 case '-':
    r = a-b;
    break;
 case '*':
    r = a*b;
    break;
 case '/':
    r = a/b;
    break;
}
cout << "r: " << r << endl;
```

```
char grade;
cout << "Enter your control grade (A, B, C, D, E or F): ";
cin >> grade;

switch (toupper(grade)) {
case 'A':
    cout << "Excellent. "
         << "You need not take the final." << endl;
    break;

case 'B':
    cout << "Very good. ";
    grade = 'A';
    cout << "Your midterm grade is now "
         << grade << endl;
    break;

case 'C':
    cout << "Passing." << endl;
    break;

case 'D':
case 'F':
    cout << "Not good. "
         << "Go study." << endl;
    break;

default:
    cout << "That is not a possible grade." << endl;
}
```

```
char grade;
cout << "Enter your control grade (A, B, C, D, E or F): ";
cin >> grade;

switch (toupper(grade)) {
case 'A':
    cout << "Excellent. "
         << "You need not take the final." << endl;
    break;

case 'B':
    cout << "Very good. ";
    grade = 'A';
    cout << "Your midterm grade is now "
         << grade << endl;
    break;

case 'C':
    cout << "Passing." << endl;
    break;

case 'D':
case 'F':
    cout << "Not good. "
         << "Go study." << endl;
    break;

default:
    cout << "That is not a possible grade." << endl;
}
```

What does this mean?

Loops

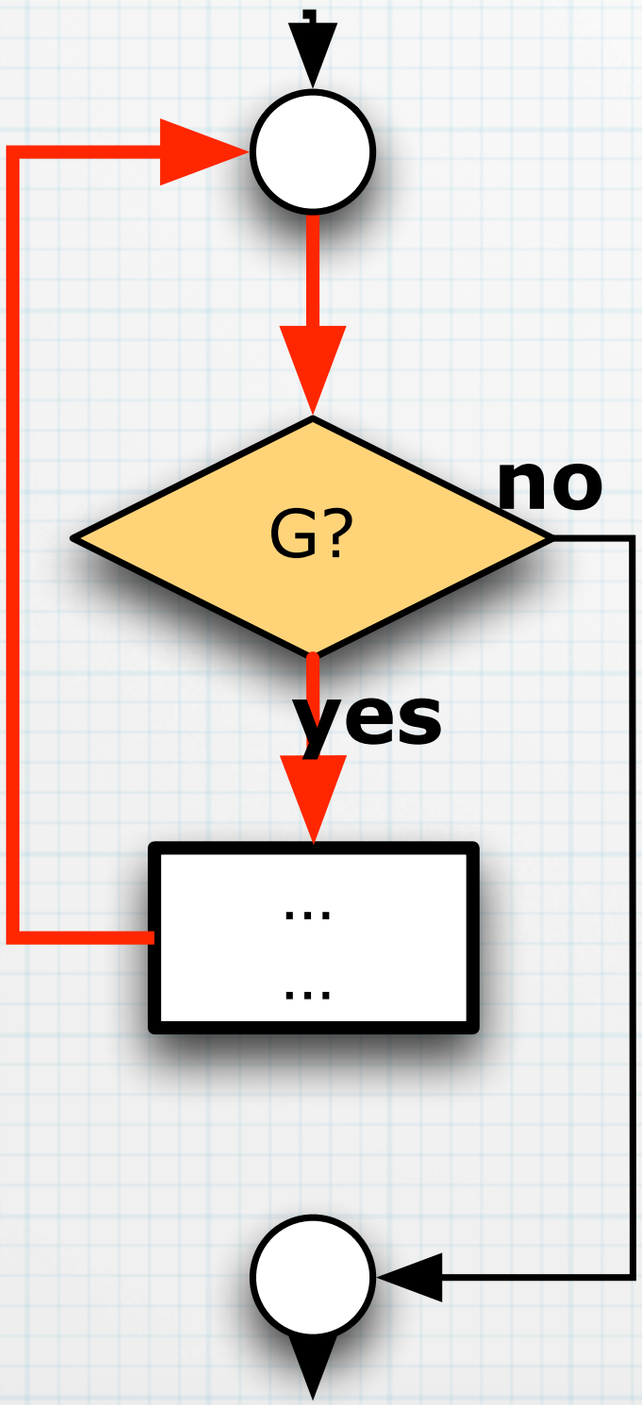
Write one hundred times!
"I'll not throw paper airplanes in class"

Loops

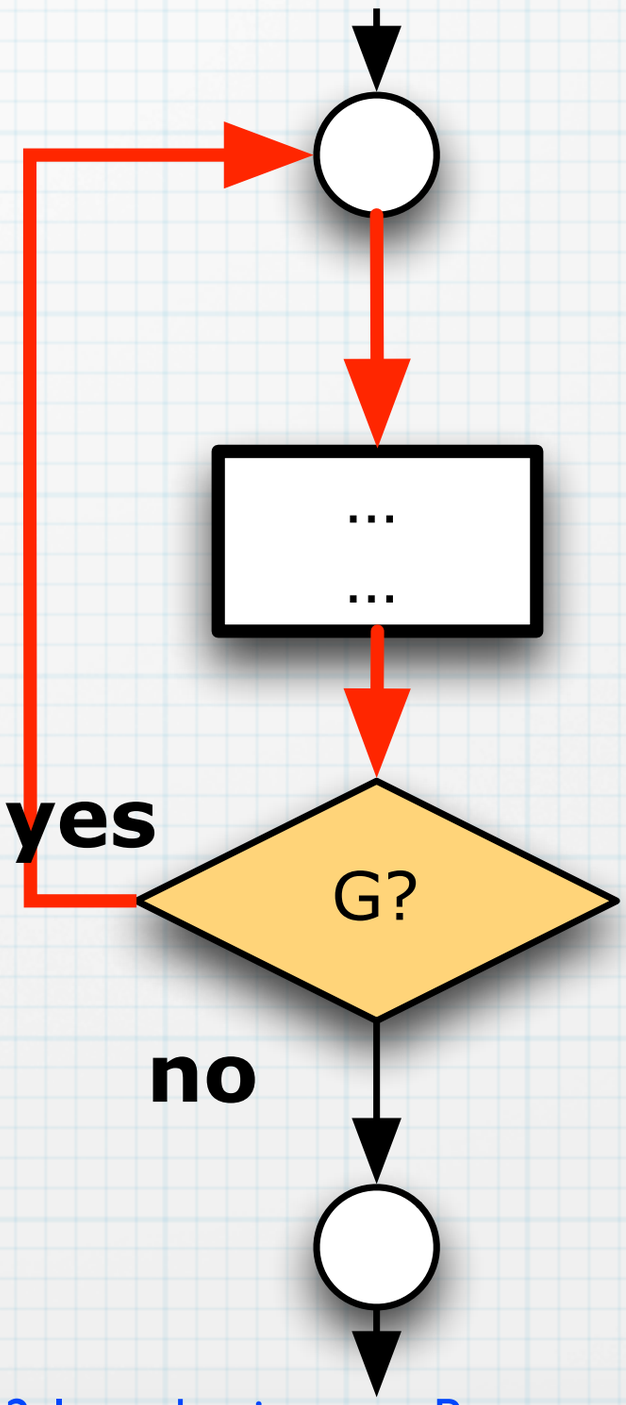
```
#include <iostream>
using namespace std;

int main()
{
  int i = 0;
  while ( i < 100 ) {
    cout << "I'll not throw paper airplanes in class" << endl;
    ++i;
  }
  return 0;
}
```

Types of Loops

while do...

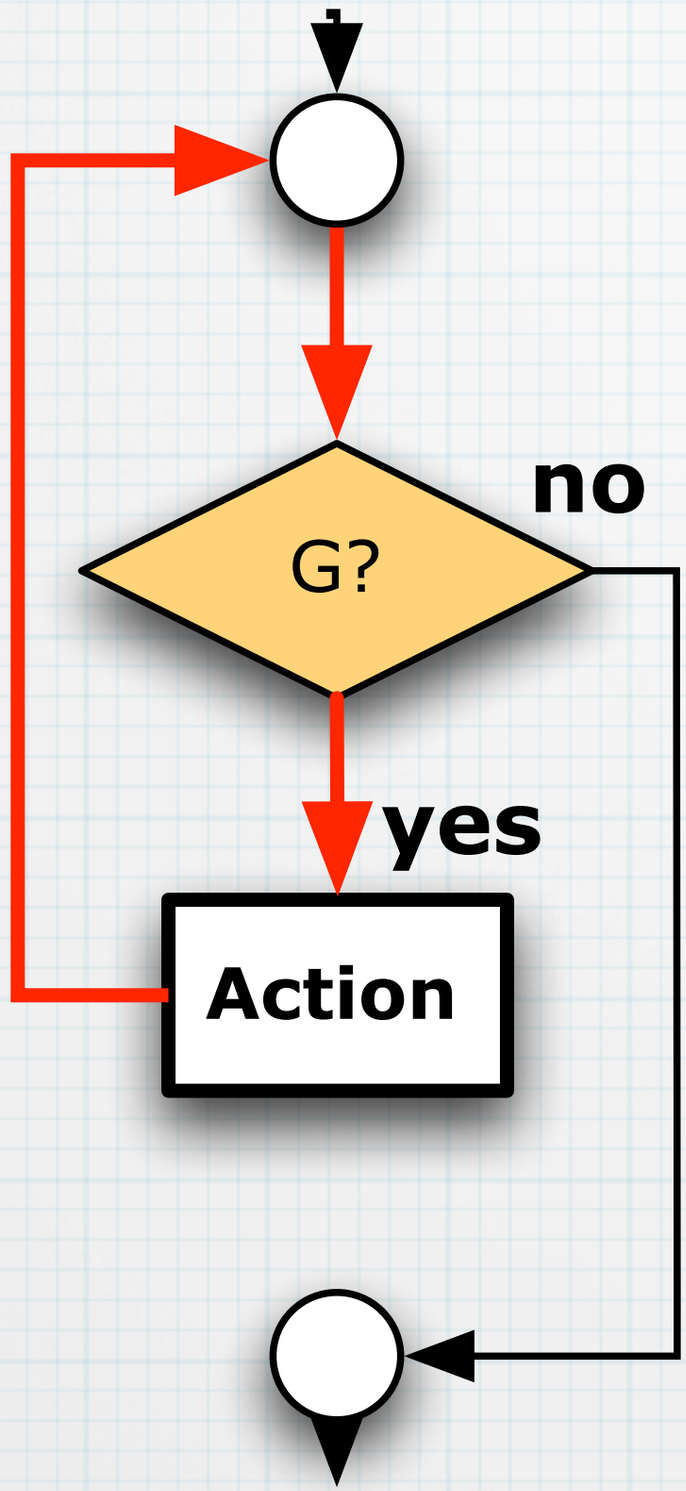


do while...



5) Control flow. while loop

while do...



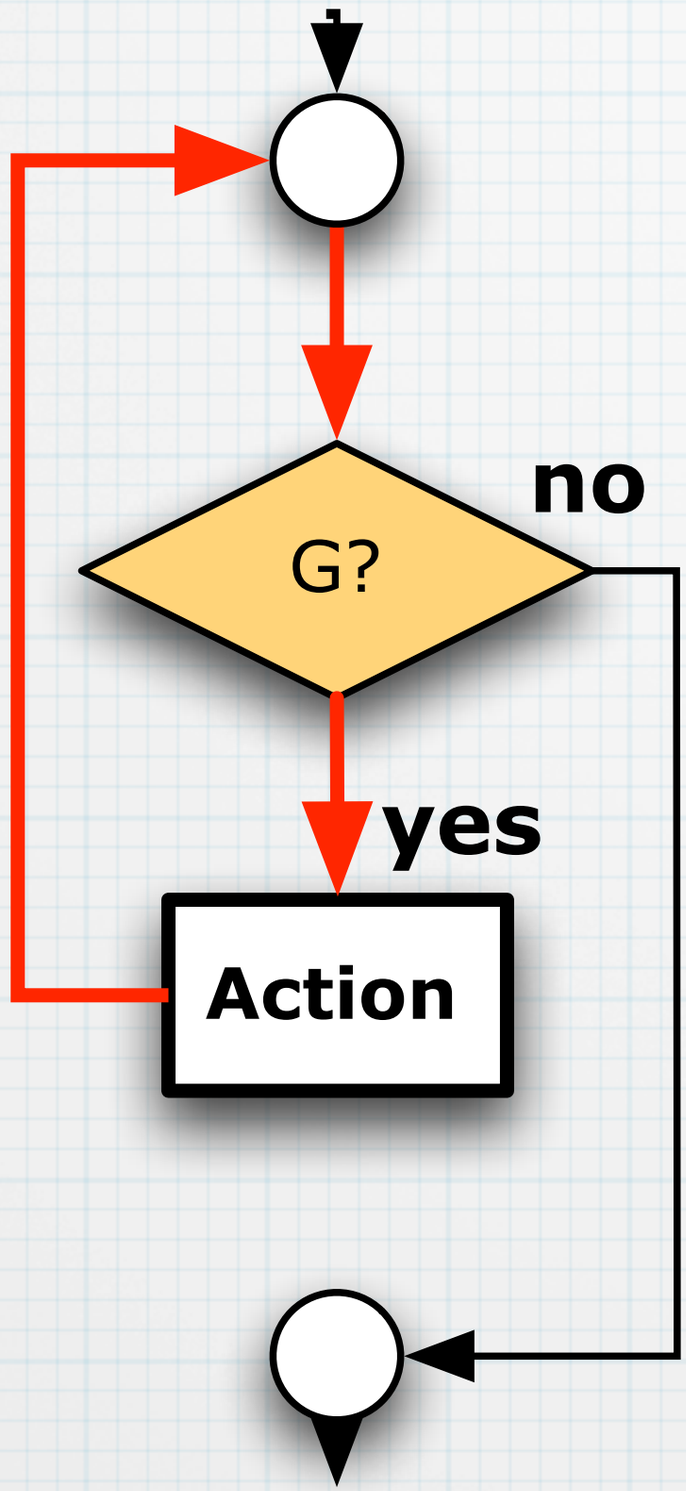
while

```
while (cond) {
    action;
}
```

- previous situation might avoid the any execution of the Action

5) Control flow. while loop

while do...



while

```
while (cond) {
    action;
}
```

Ask before any action

- previous situation might avoid the any execution of the Action

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

int main()
{
    float x;

    cout << "Enter a number (0 ends): ";
    cin >> x;
    while ( x > 0) {
        cout << sqrt(x) << endl;
        cin >> x;
    }

    return 0;
}
```

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

int main()
{
    int i = 0;
    while (i < 5) {
        cout << '*';
        ++i;
    }
    return 0;
}
```

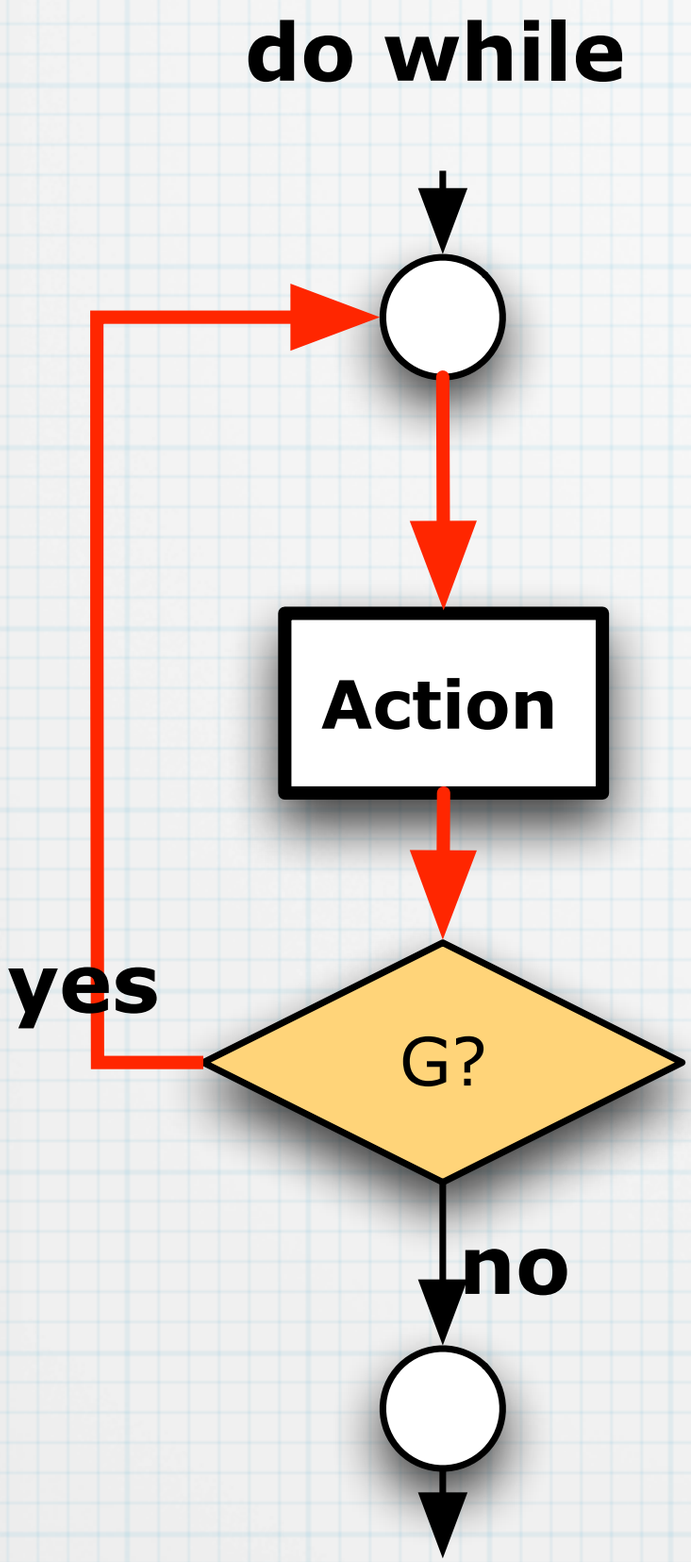
```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    float x;

    cout << "Enter a number (0 ends): ";
    while ( cin >> x and x > 0) {
        cout << sqrt(x) << endl;
    }

    return 0;
}
```

- Exercices:

1. Write a program that asks the user for a numbers and sum all of them until the user enters 0. Print the sum after that

2. … print them the average of them as well

5) Control flow. do-while loop

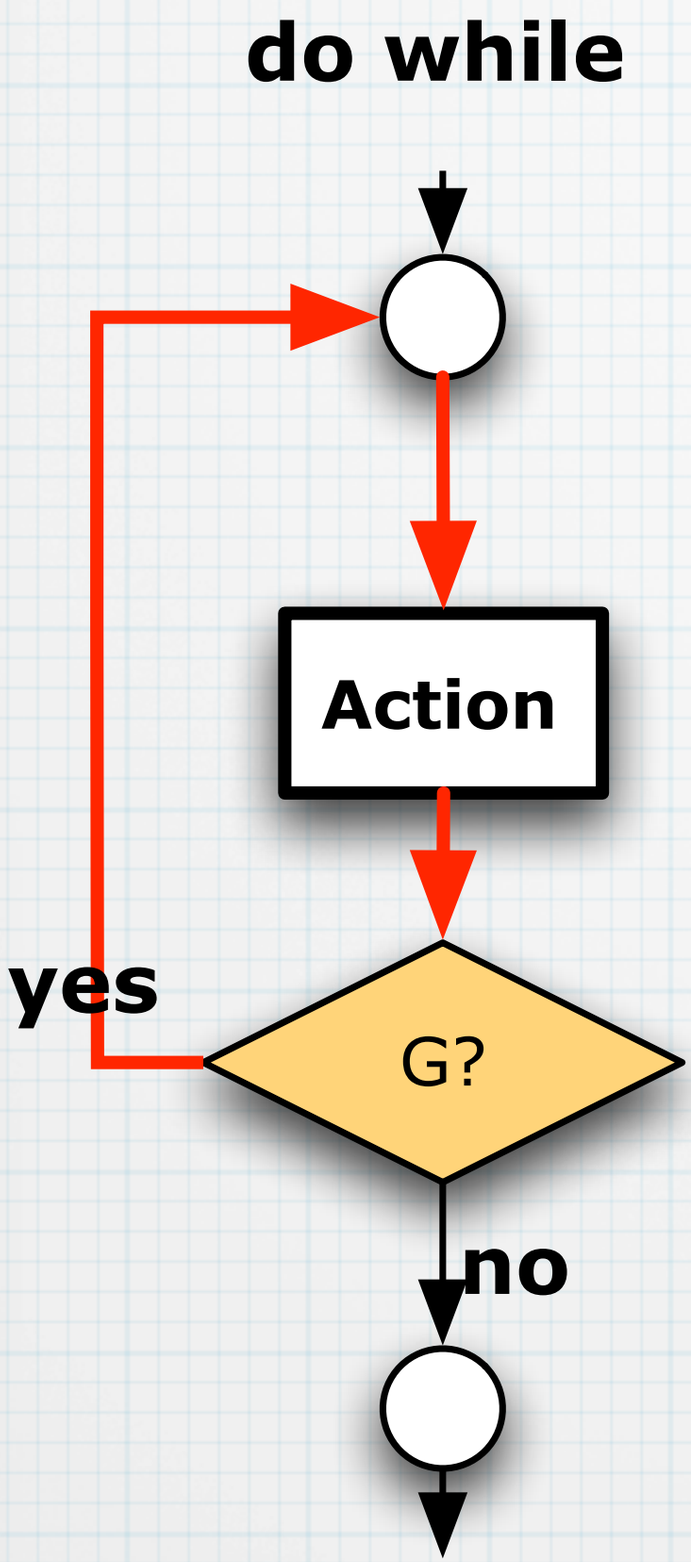


do • while

```
do {
    action;
} while
```

- Previous state does not matter. Action is always executed at least once

5) Control flow. do-while loop



do • while

```
do {
    action;
} while
```

First do

then ask!!

- Previous state does not matter. Action is always executed at least once

5) Control flow. do-while loop

```
// dowhiledemo.cpp

#include <iostream>
using namespace std;

int main() {
    int secret = 15;
    int guess; // No initialisation needed
    do {
        cout << "guess the number: ";
        cin >> guess; // Initialisation
    }   while(guess != secret);
    cout << "You got it!" << endl;

    return 0;
}
```

5) Control flow. do-while loop

```
// dowhiledemo.cpp

#include <iostream>
using namespace std;

int main() {
    int secret = 15;
    int guess; // No initialisation needed
    do {
        cout << "guess the number: ";
    } while(cin >> guess and guess != secret);
    cout << "You got it!" << endl;

    return 0;
}
```

for(; ;)

- For loops are specialised kinds of loops in which a **previously known range** of values is traversed
- Typically an **integer** ranging from a value up to another is traversed

5) Control flow. For loops

for(; ;)

```
for ( int i = 0; i < 10; ++i ) {
  cout << i << " squared = " << i*i << endl;
}
```

- For loops are specialised kinds of loops in which a **previously known range** of values is traversed
- Typically an **integer** ranging from a value up to another is traversed

5) Control flow. For loops

for(; ;)

```
for ( int i = 0; i < 10; ++i ) {
  cout << i << " squared = " << i*i << endl;
}
```

```
0 squared = 0
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
5 squared = 25
6 squared = 36
7 squared = 49
8 squared = 64
9 squared = 81
```

- For loops are specialised kinds of loops in which a **previously known range** of values is traversed
- Typically an **integer** ranging from a value up to another is traversed

5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```

5) Control flow. For loops. How do they work?

before ①

```
for ( ① ; ② ; ③ ) {
  actions;
}
```

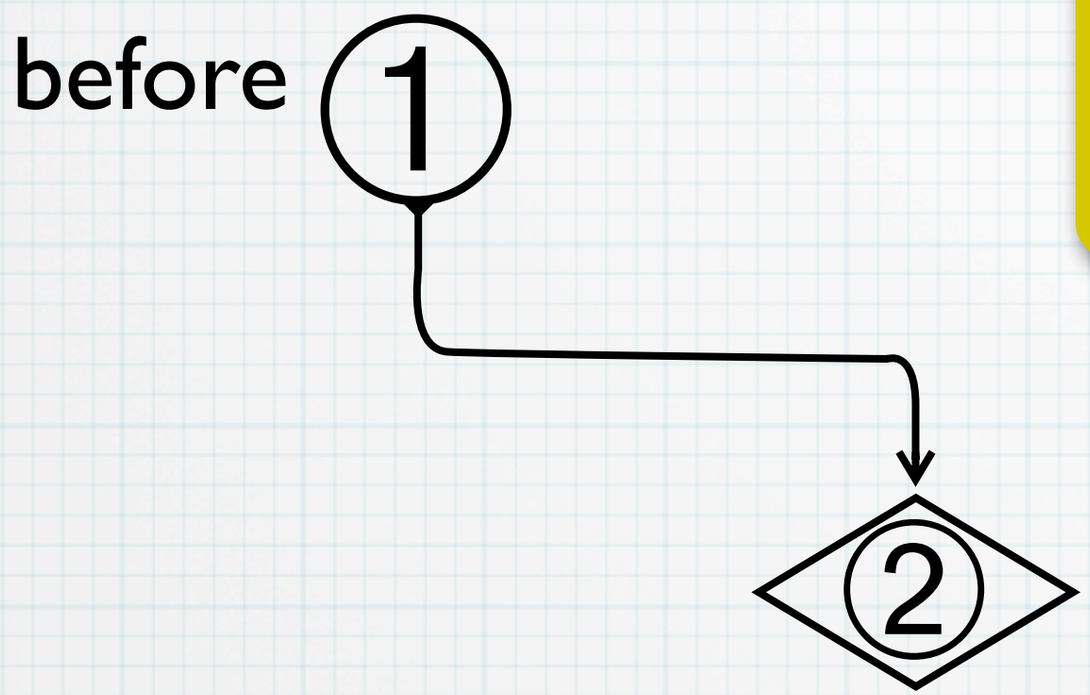
5) Control flow. For loops. How do they work?

before ①

```
for ( ① ; ② ; ③ ) {
  actions;
}
```

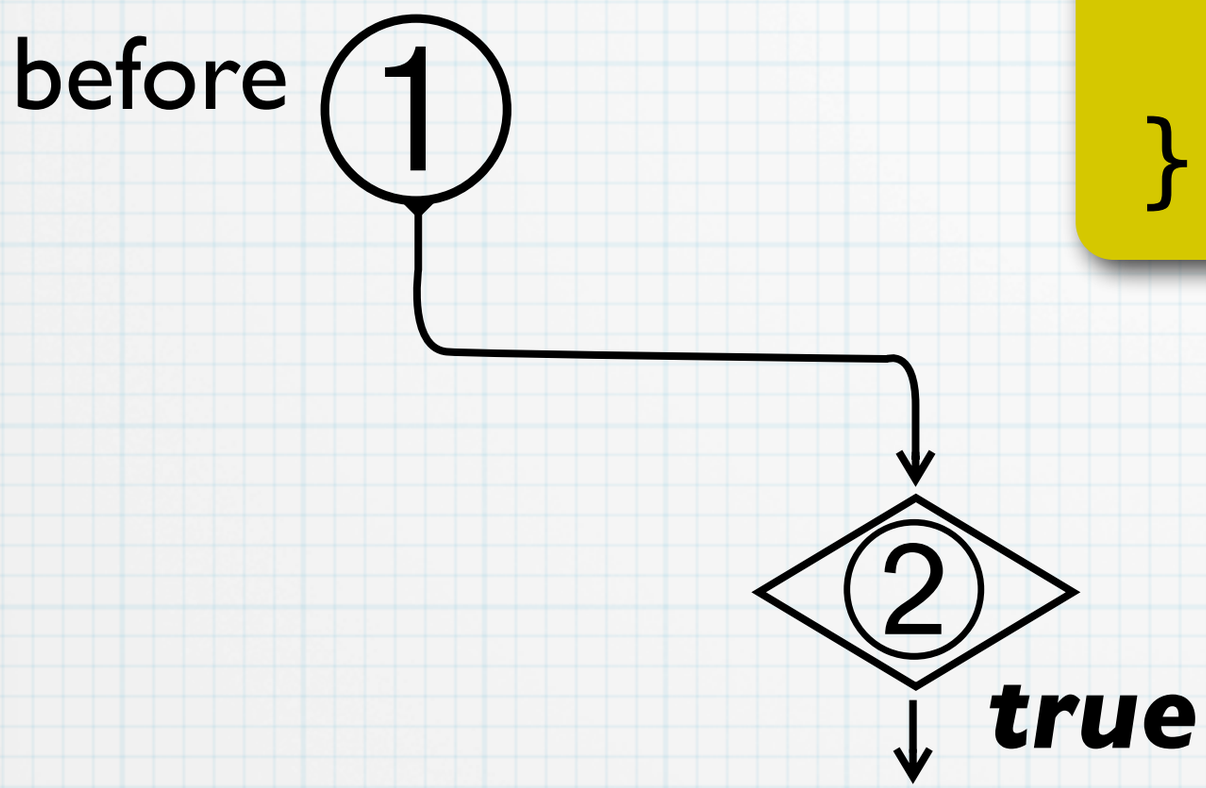
5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```



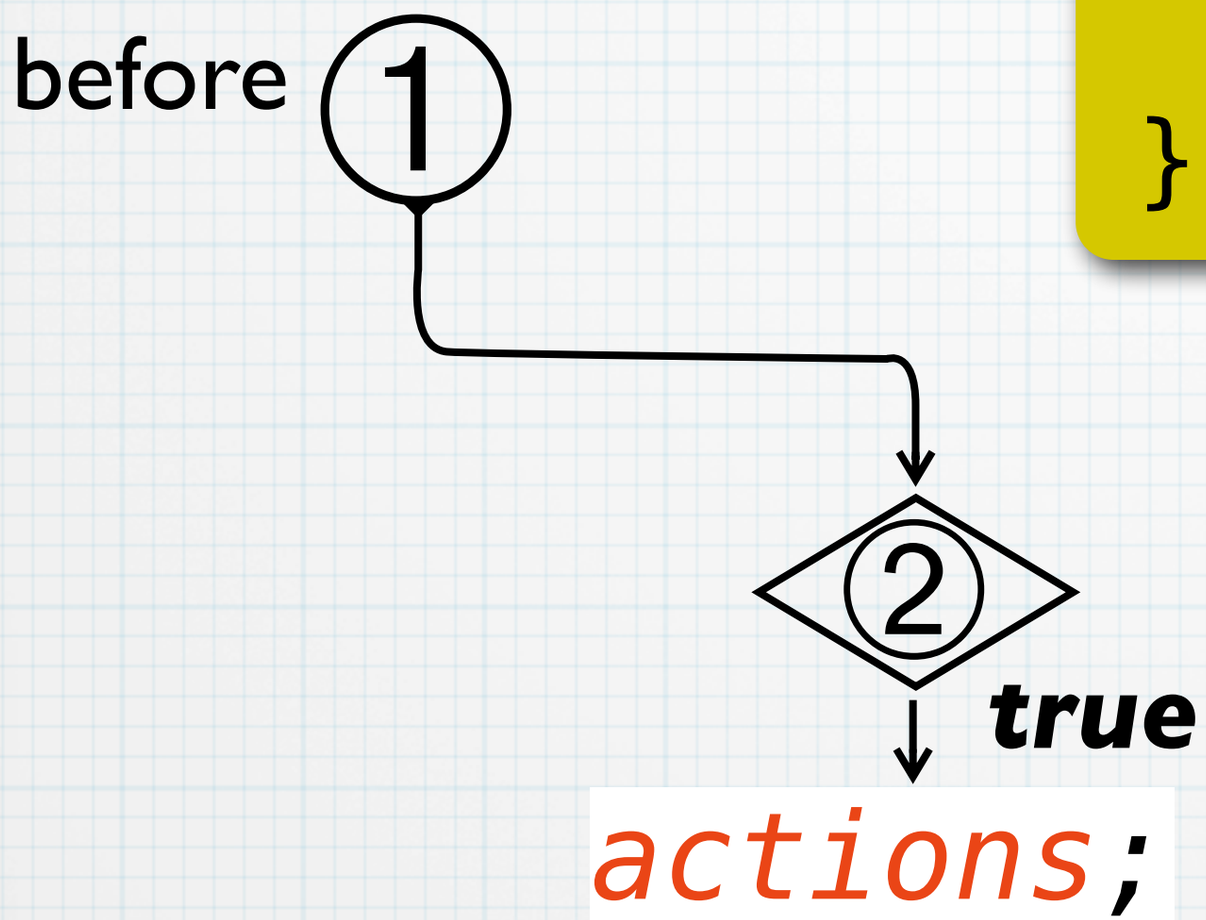
5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```



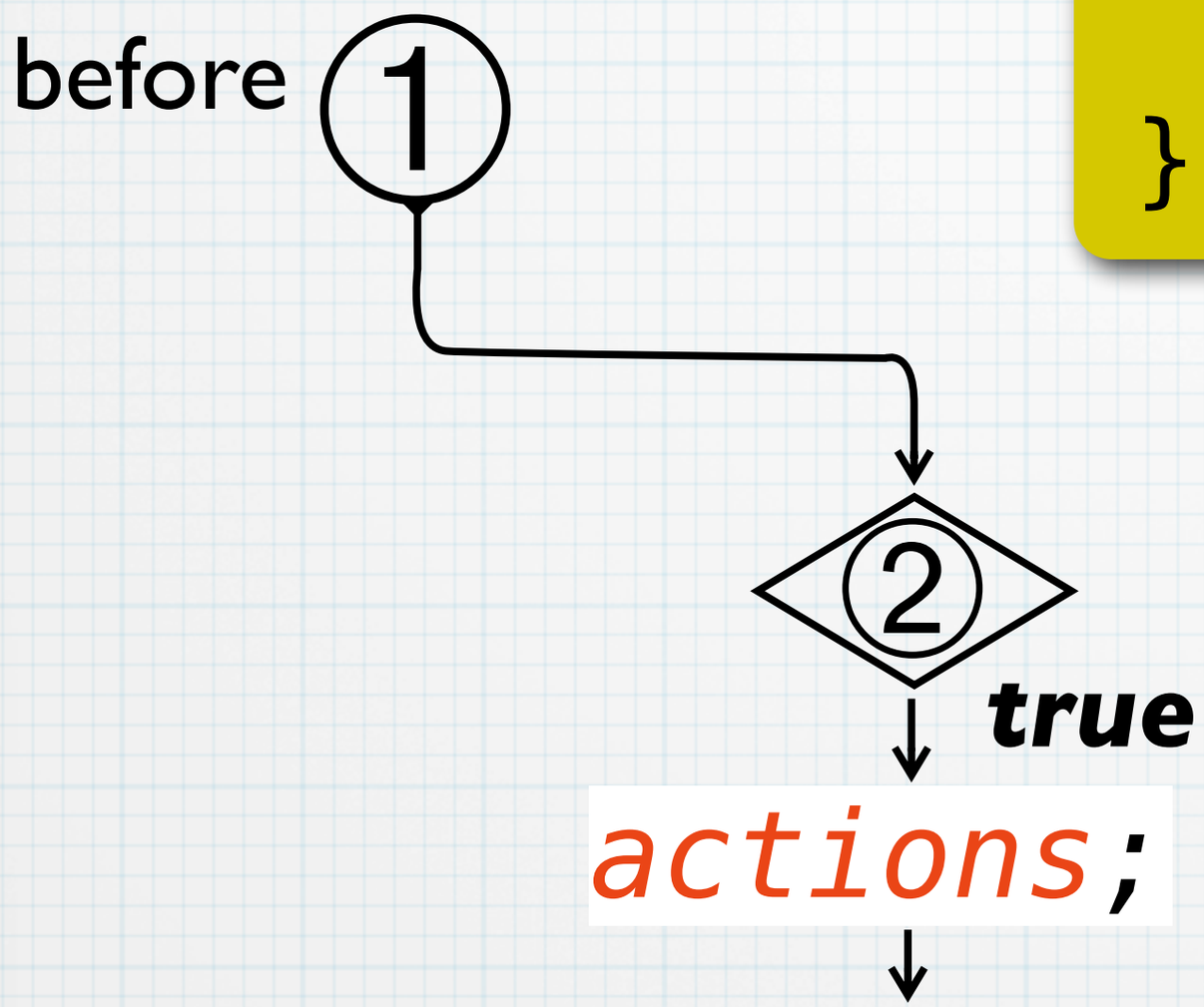
5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```



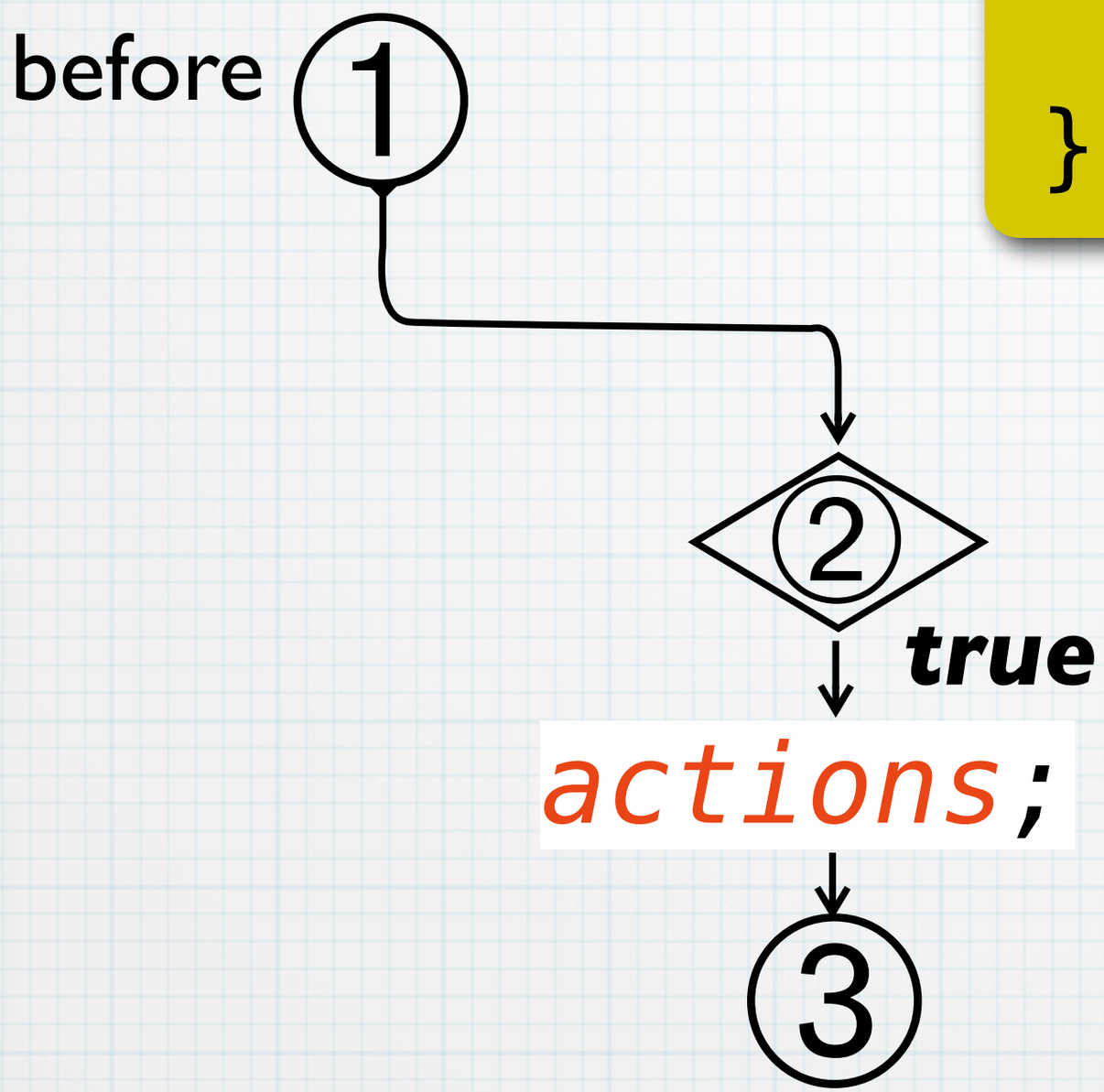
5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```



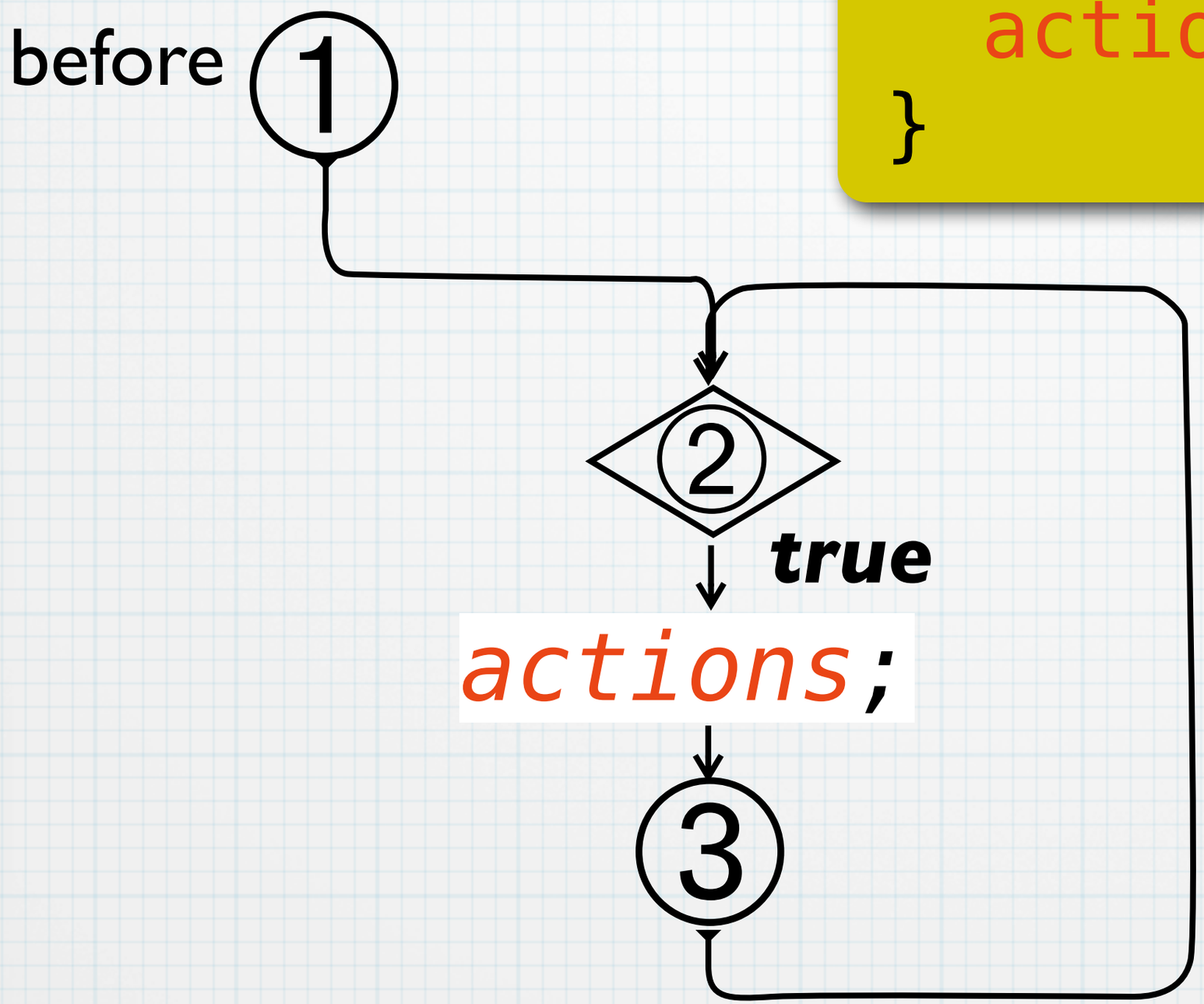
5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```



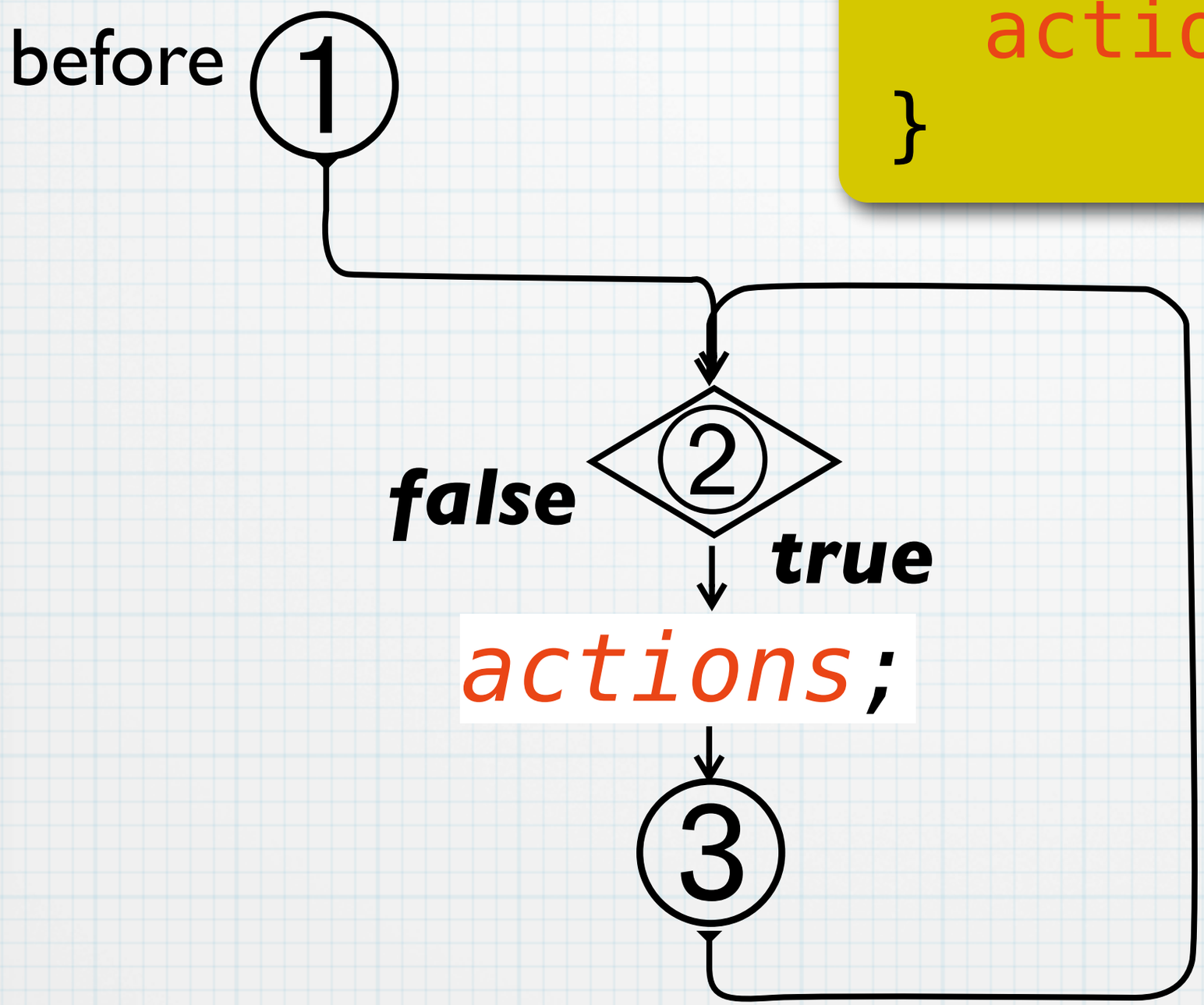
5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```



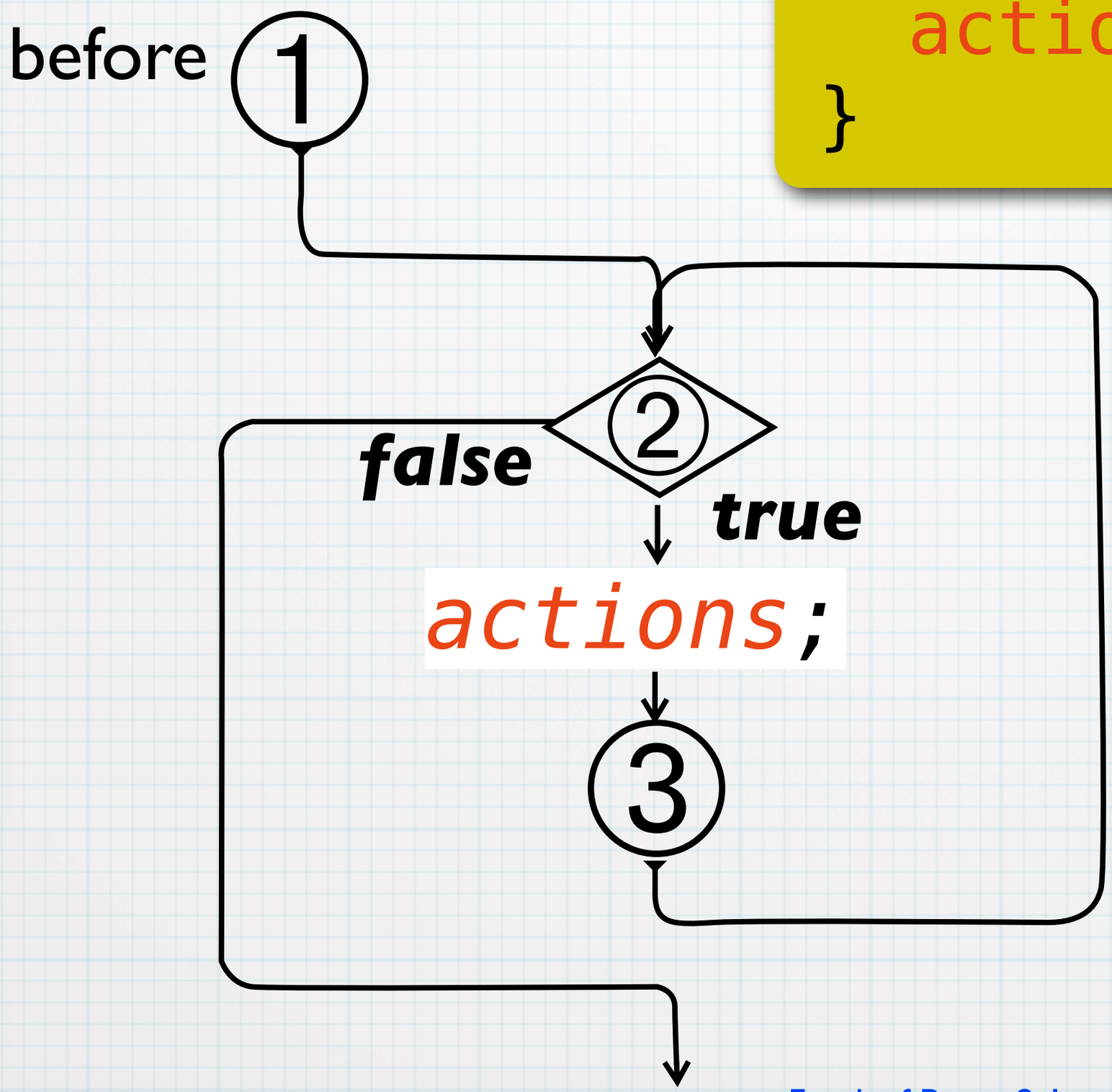
5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```



5) Control flow. For loops. How do they work?

```
for ( ① ; ② ; ③ ) {
  actions;
}
```



5) Control flow. For loops. Examples

```
for (int i = 0; i < 80; i++)
  cout << "*";
```

What does the next one do?

5) Control flow. For loops. Examples

```
for (int i = 0; i < 80; i++)
  cout << "*";
```

What does the next one do?

```
int s=0;
for ( int i = 1; i <= 100; ++i ) {
  s += i*i;
}
cout << s;
```

PROBLEMS

$$\sum_{i=1}^{1000} i$$

Compute and print the sum of all the numbers from 1 to 1,000

Do it with ***for***

$$1 + 2 + 3 + 4 + \cdots + 1000 = ?$$

Ask the user for a int number *n* and compute and print its factorial *n*! using **for**

$$n! = 1 \times 2 \times 3 \cdots \times n$$

Ask the user for a series of ints.
Finish the reading when **0** is entered
(0 is not part of the series)

```
while ( cin >> x and x != 0) {
```

the loop should be counting the position of each number

After the loop the program will print the position the last **12** entered

números	8	9	7	12	13	24	12	56	9	9	9	2	0
posiciones	1	2	3	4	5	6	$\boxed{7}$	8	9	10	11	12	

Ask the user for a series of ints.
Finish the reading when **0** is entered
(0 is not part of the series)

After the loop the program will print the positions of the **first** and the **last 12** entered

números	8	9	7	12	12	24	12	56	9	9	9	2	0
posiciones	1	2	3	$\boxed{4}$	5	6	$\boxed{7}$	8	9	10	11	12	

Is a user given number ***prime***?

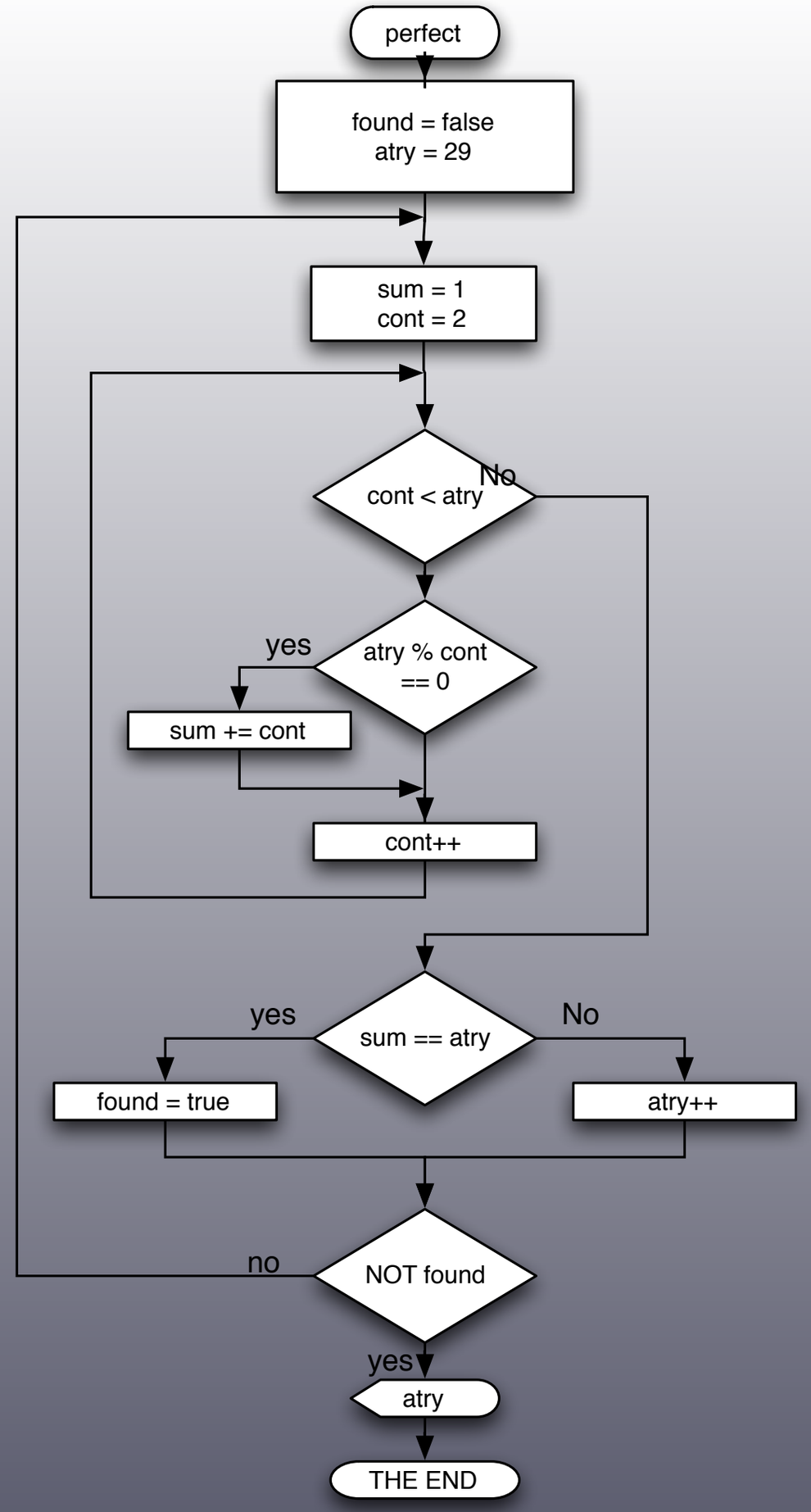
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, …

Is prime?

```
// isprime_simple.cpp
// juanfc 2012-10-03
#include <iostream>
using namespace std;

int main() {
  int i = 2, n;
  cout << "n?: "; cin >> n;
  while (i < n and n % i != 0)
    ++i;
  if (i >= n)
    cout << "YES, is prime" << endl;
  else
    cout << "NO, it is not prime" << endl;
  return 0;
}
```

Find if a number is *perfect*

A number is *perfect*, when it is the sum of its divisors



Find if a number is *perfect*

A number is *perfect*, when it is the sum of its divisors

The End