INFORMÁTICA INDUSTRIAL
INDUSTRIAL COMPUTING

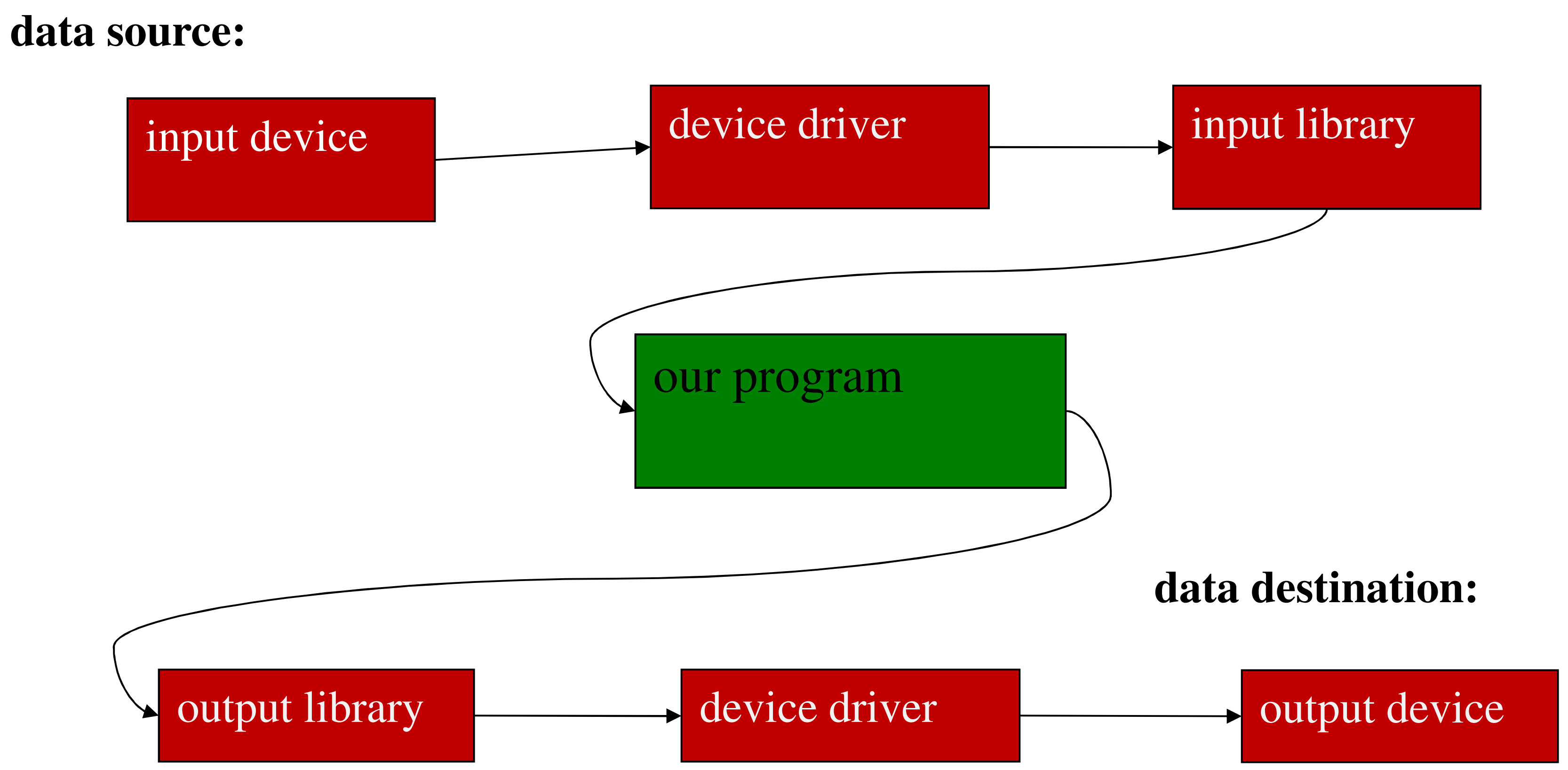
Second class of

BASICS OF
PROGRAMMING WITH C++

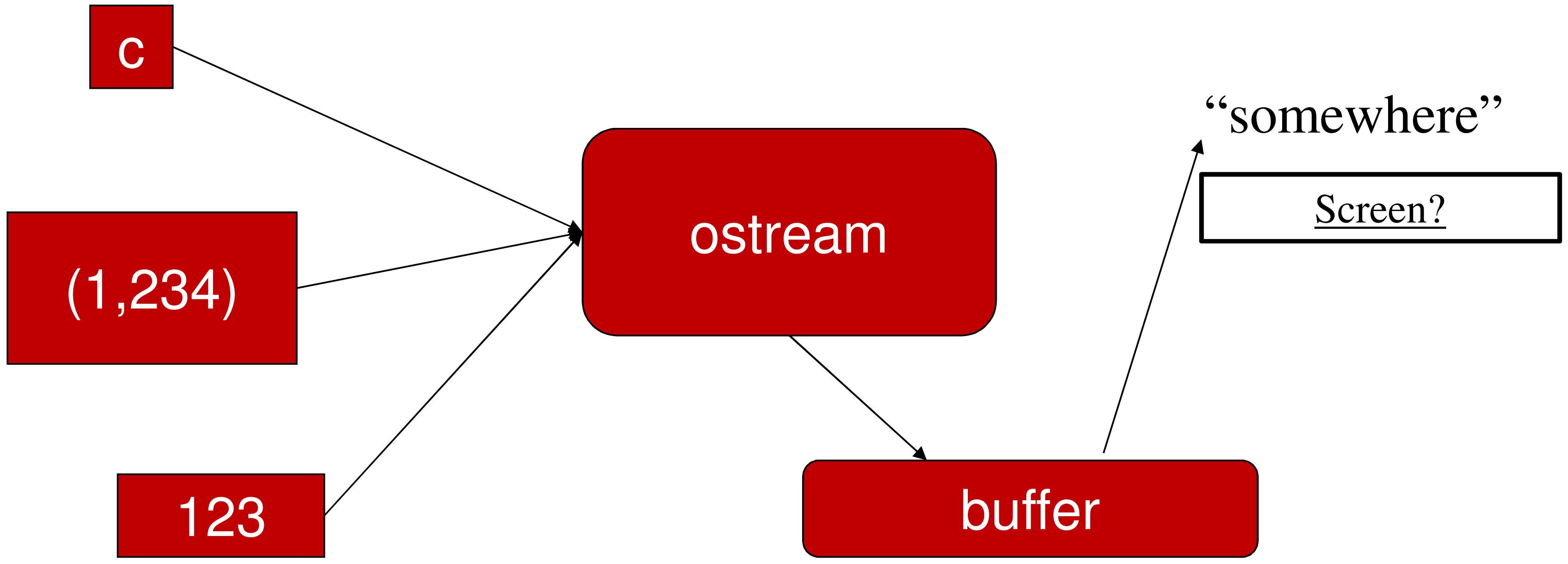
Profesor: Mohamed Abderrahim/Juan Gonzalez
Departamento de Ingeniería de Sistemas y Automática

4. Standard Input and Output

Input and Output

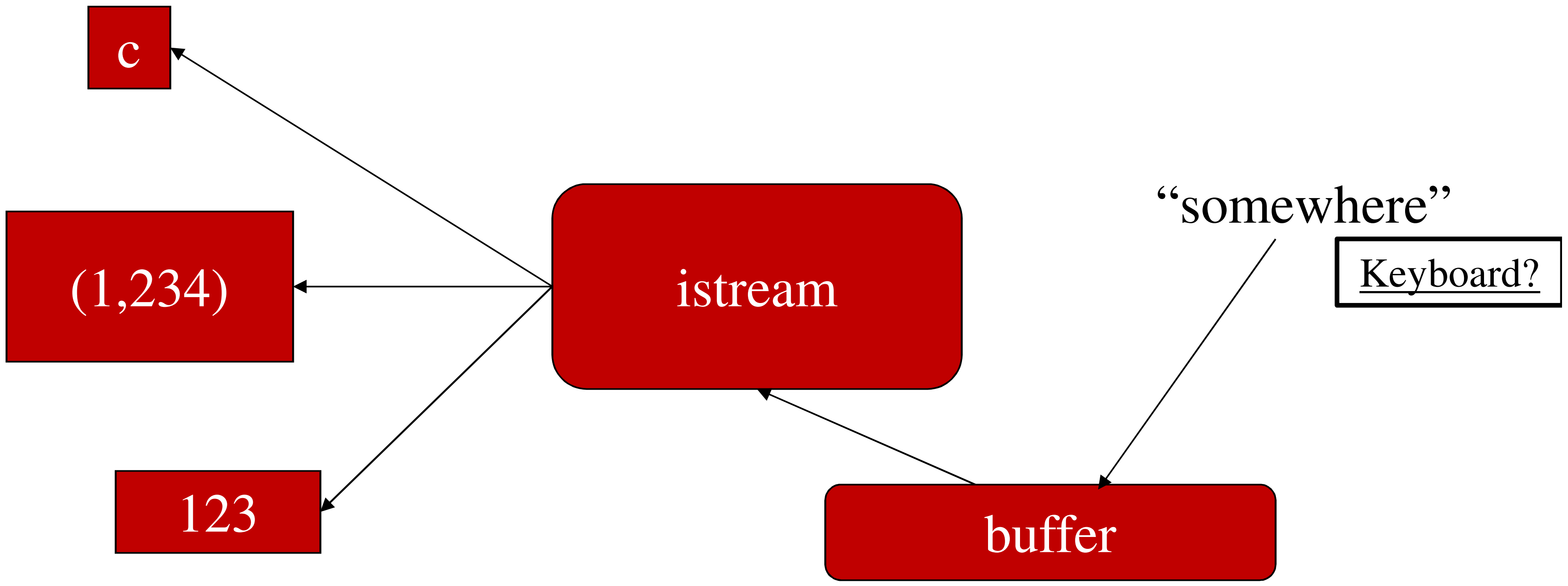


The stream model



- An **ostream**
  - turns values of various types into character sequences
  - sends those characters somewhere
    - *E.g.*, console, file, main memory, another computer

The stream model



- An **istream**
  - turns character sequences into values of various types
  - gets those characters from somewhere
    - *E.g.*, keyboard, file, main memory, another computer

The stream model

- Reading and writing
  - Of typed entities
    - << (output) and >> (input) plus other operations
    - Type safe
    - Formatted
  - Typically stored (entered, printed, etc.) as text
    - But not necessarily (see binary streams in Files' class later)
  - Extensible
    - The user can define any I/O operations for the user defined types
  - A stream can be attached to any I/O or storage device

I/O error handling

- Sources of errors
  - Human mistakes
  - Files that fail to meet specifications
  - Specifications that fail to match reality
  - Programmer errors
  - Etc.

- iostream reduces all errors to one of four states
  - **good()** // *the operation succeeded*
  - **eof()** // *we hit the end of input ("end of file")*
  - **fail()** // *something unexpected happened*
  - **bad()** // *something unexpected and serious happened*

Observation

- As programmers we prefer regularity and simplicity
  - But, our job is to meet users´ expectations
- People are very fussy/particular/picky about the way their output looks
  - They often have good reasons to do it that way …
  - Convention/tradition rules
    - What does 123,456 mean?
    - What does (123) mean?
  - The world (of output formats) is dictated by needs

Numerical Base Output

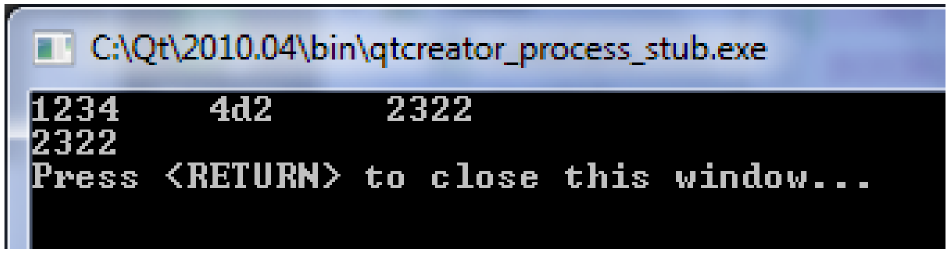
- You can change “base”
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
    cout << dec << 1234 << "\t(decimal)\n"
              << hex << 1234 << "\t(hexadecimal)\n"
              << oct << 1234 << "\t(octal)\n";
// The '\t' character is “tab” (short for “tabulation character”)

// results:
    1234    (decimal)
    4d2     (hexadecimal)
    2322    (octal)
```

Manipulators

- You can change “base”
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
    cout << 1234 << '\t'
               << hex << 1234 << '\t'
               << oct << 1234 << '\n';
    cout << 1234 << '\n';       // the octal base is still in effect

// results:
    1234      4d2      2322
    2322
```

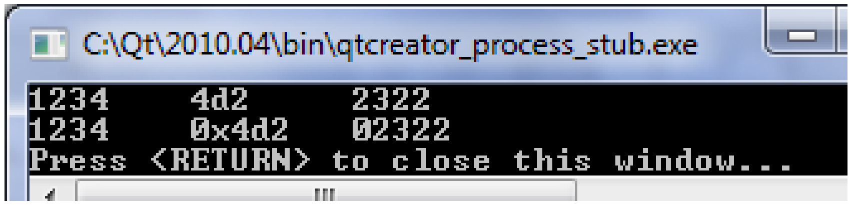

C:\Qt\2010.04\bin\qtcreator_process_stub.exe

```
1234    4d2     2322
2322
Press <RETURN> to close this window...
```

Other Manipulators

- You can change “base”
  - Base 10 == decimal; digits: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; digits: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; digits: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// simple test:
    cout << 1234 << '\t'
                << hex << 1234 << '\t'
                << oct << 1234 << endl;                    // '\n'
    cout << showbase << dec;          // show bases
    cout << 1234 << '\t'
                << hex << 1234 << '\t'
                << oct << 1234 << '\n';
// results:
    1234        4d2       2322
    1234        0x4d2   02322
```


C:\Qt\2010.04\bin\qtcreator_process_stub.exe

```
1234    4d2     2322
1234    0x4d2   02322
Press <RETURN> to close this window...
```

Floating-point Manipulators

- You can change floating-point output format
  - general – **iostream** chooses best format using **n** digits (this is the default)
  - **scientific** – one digit before the decimal point plus exponent; **n** digits after **.**
  - **fixed** – no exponent; **n** digits after the decimal point

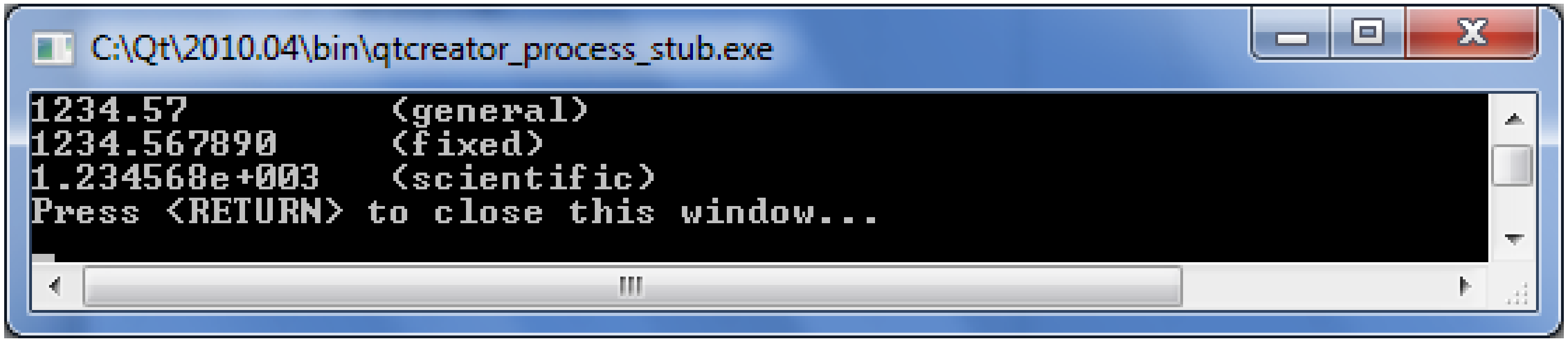
```
// simple test:
cout << 1234.56789 << "\t\t(general)\n"      //  \t\t  to line up columns
     << fixed << 1234.56789 << "\t(fixed)\n"
     << scientific << 1234.56789 << "\t(scientific)\n";
```

Floating-point Manipulators

// simple test:

cout << 1234.56789 << "\t\t(general)\n" *// \t\t to line up columns*
<< fixed << 1234.56789 << "\t(fixed)\n"
<< scientific << 1234.56789 << "\t(scientific)\n";

// results:



```
1234.57         (general)
1234.567890     (fixed)
1.234568e+003   (scientific)
Press <RETURN> to close this window...
```

Output field width

- A width is the number of characters to be used for the next output operation
  - Beware: width applies to next output only (it doesn't “stick” like precision, base, and floating-point format)
  - Beware: output is never truncated to fit into field
    - (better a bad format than a bad value)

```
// example:
    cout << 123456 <<'|'<< setw(4) << 123456 << '|'
            << setw(8) << 123456 << '|' << 123456 << "|\n";
    cout << 1234.56 <<'|'<< setw(4) << 1234.56 << '|'
            << setw(8) << 1234.56 << '|' << 1234.56 << "|\n";
    cout << "asdfgh" <<'|'<< setw(4) << "asdfgh" << '|'
            << setw(8) << "asdfgh" << '|' << "asdfgh" << "|\n";
```

```
// results:
    123456|123456|  123456|123456|
    1234.56|1234.56| 1234.56|1234.56|
    asdfgh|asdfgh|  asdfgh|asdfgh|
```

Observation

- This kind of detail is what you need textbooks, manuals, references, online support, etc. for
  - You **always** forget some of the details when you need them

5. Arrays y Strings

Arrays

- A set of variables of the same type refered to with commun name

```
int alumno1, alumno2, alumno3;
```

$\Rightarrow$

```
int alumnos[3];
```

- declaration:

```
type name_of_array[size];
```

Arrays

- Form of use:
  - Often elements are used seperately
- int alumnos[3]
  - alumnos[0]   1st element of array
  - alumnos[1]   2nd element of array
  - alumnos[2]   3rd element of array
- The element alumnos[3]
  - Can cause errors in the program

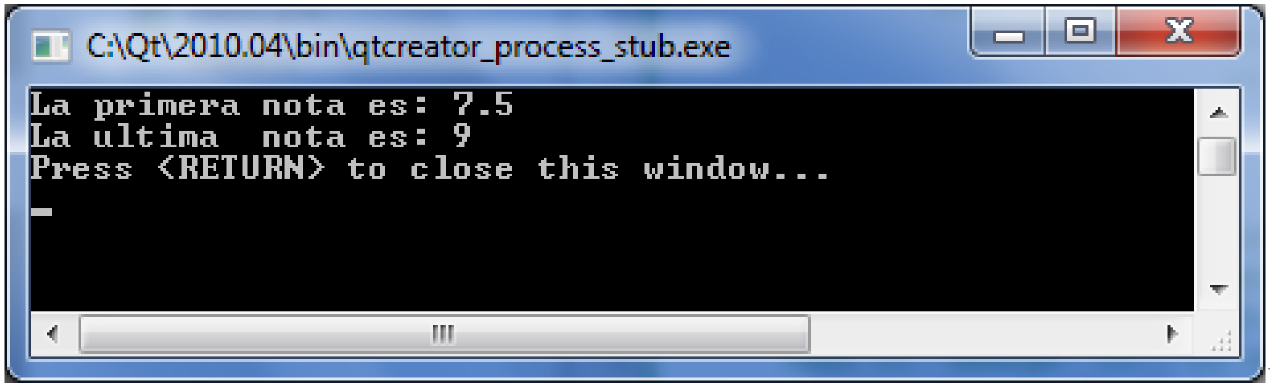
Arrays

- Can be initialized in a vector of elements between curly brackets and separated by comas .
- Or given values one by one

```
char initials[3]={‘a’ , ’c’ , ’d’};

intials[2]=‘g’;
```

Arrays

```
 int main()
{
      float notas[3]={7, 5.5 , 9 } ;
      notas[0]=7.5;
      cout << "La primera nota es: " << notas[0] << endl;
      cout << "La ultima  nota es: " << notas[2] << endl;

 return 0;
}
```



C:\Qt\2010.04\bin\qtcreator_process_stub.exe

```
La primera nota es: 7.5
La ultima  nota es: 9
Press <RETURN> to close this window...
_
```

Strings

- char cadena[8];
- Use of “” to define strings, and ‘ ’ to define characters.
- A string ends with a NULL character ´\0´
- “hola” is:

h	o	l	a	´\0´

- sizeof(‘hola’) is **4,** sizeof(“hola”) is **5**
- “b” is **not** equal to ´b´
  - “b” is an array of an element ´b´+´\0´
  - ´b´ is a character

Strings

- The'\0' is important to know where the end the string is
- This way, the same variable can store strings of different lengths
- If the '\0' is not added, the string will include all characters until casually it finds another '\0'

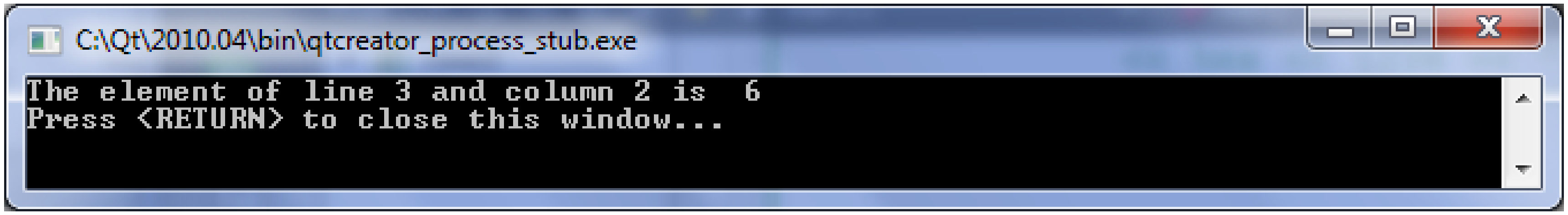
Multidimensional Arrays

- **Type** name[size1][size2][size3];
- int matriz[4][2]= {

```
int matriz[4][2]=     {
                      {1,2},
                      {3,4},
                      {5,6},
                      {7,8}
                           };
```

Multidimensional Arrays

```
void main(){
        int matriz[4][2] = {{1,2},{3,4},{5,6},{7,8}};
        cout<<“The element of line 3 and column 2 is  "<<matriz[2][1]<<endl;
}
```



C:\Qt\2010.04\bin\qtcreator_process_stub.exe

```
The element of line 3 and column 2 is  6
Press <RETURN> to close this window...
```

6. Control Flow

Control Flow

```
For each student
    Do
        Study
        Set exam
        If not happy with the marks  ↙
                            Go to revision
    While the subject has not passed
```

Operators for Control Flow

- Relations

  < > <= >=

- Equality - inequality

  = = (note, is not =, assignment) !=

- Logic

  && || !

- All give:
  - false if condition is not verified
  - true when condition is fulfilled

- En general any number different from zero means *true*

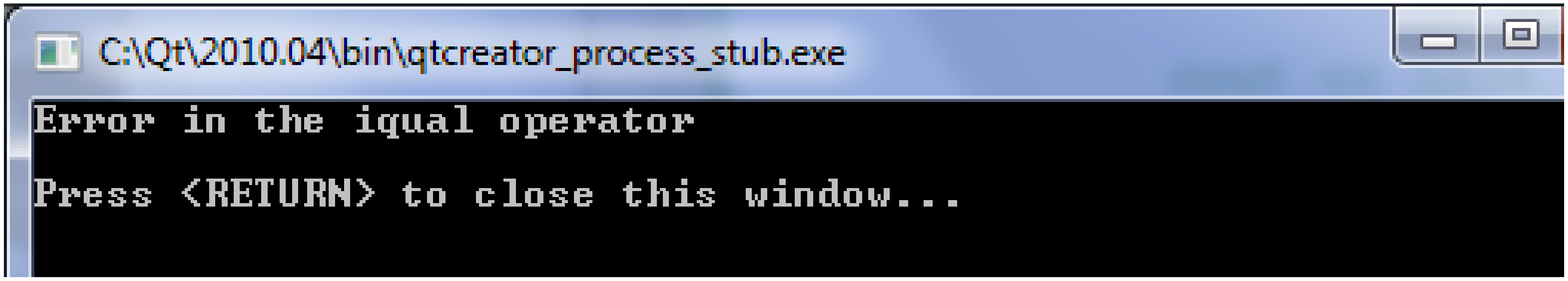
if

```
if(expression)
        statements1;
statements2;
```

- if expression evaluates to true statements1 is executed (and then statements2)
- if expression evaluates to false, statements2 are directly executed

if

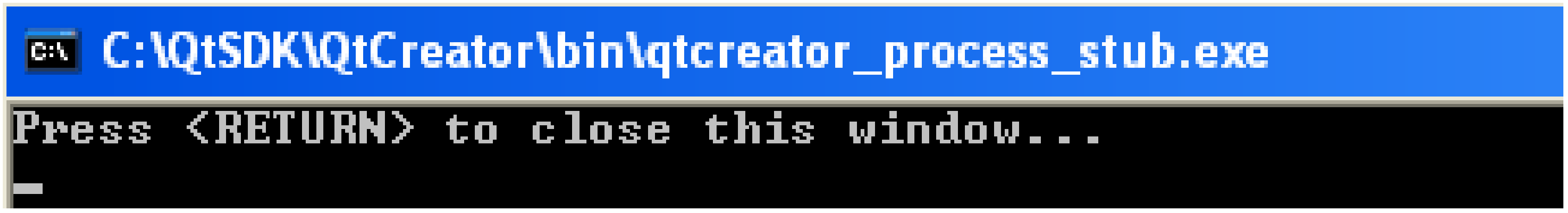
```
void main(){
        int i=0;
        if(i!=1)
                cout<<"Error in the iqual operator ";
}
```


C:\Qt\2010.04\bin\qtcreator_process_stub.exe

```
Error in the iqual operator
Press <RETURN> to close this window...
```

If

```
void main(){
        int i=0;
        if(i==1)
                cout<<"Error in the iqual operator ";
}
```



```
C:\QtSDK\QtCreator\bin\qtcreator_process_stub.exe
Press <RETURN> to close this window...
_
```

if-else

```
if(expression)
  statement1;
else
  statement2;
following statements;
```

if-else

If expression is true statements1 is executed

If not statement2 is executed

After that and in all cases, the following statements are executed

if-else

```
main()
{
	int numero;
	scanf(“%d”,&numero);
	if(numero < 0)
			printf(“Número negativo”);
	else
	{
			printf(“Numero positivo\n”);
			printf(“La raíz cuadrada es %f”,sqrt(numero));
	}
}
```

Nested If-else

```
#include <stdio.h>
main()
{
int i,j;
i=3;
j=-3;
if(i<0)
    if(j<0)
            printf(“i j menores que 0\n");
    else
    printf("i no es menor que cero\n");
}
```

```
#include <stdio.h>
main()
{
int i,j;
i=3;
j=-3;
if(i<0){
    if(j<0)
            printf(”i j menores que 0\n");
}
else
    printf("i no es menor que cero\n");
}
```

Nested If-else

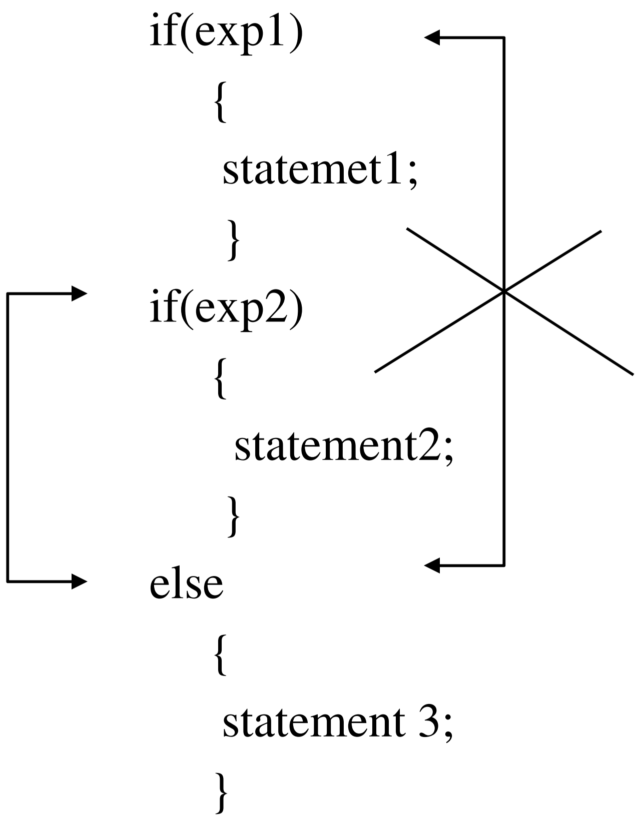
```
if(exp1)
{
    if(exp2)
    {
      statement1;
    }
    else
    {
      statement2;
    }
}
else
{
    statement3;
}
```

Nested If-else

```
#include <stdio.h>
main()
{
int i,j;
i=3;
j=-3;
if(i < 0)
{
      if(j < 0)
               printf ("i j menores que 0\n");
      else
               printf("j no es menor que cero, aunque i sí lo sea\n");}
else
      printf("i no es menor que cero\n");
}
```

Nested If-else

The else is associated with
The nearest if



```
if(exp1)
     {
      statemet1;
      }
if(exp2)
     {
       statement2;
      }
else
     {
      statement 3;
     }
```

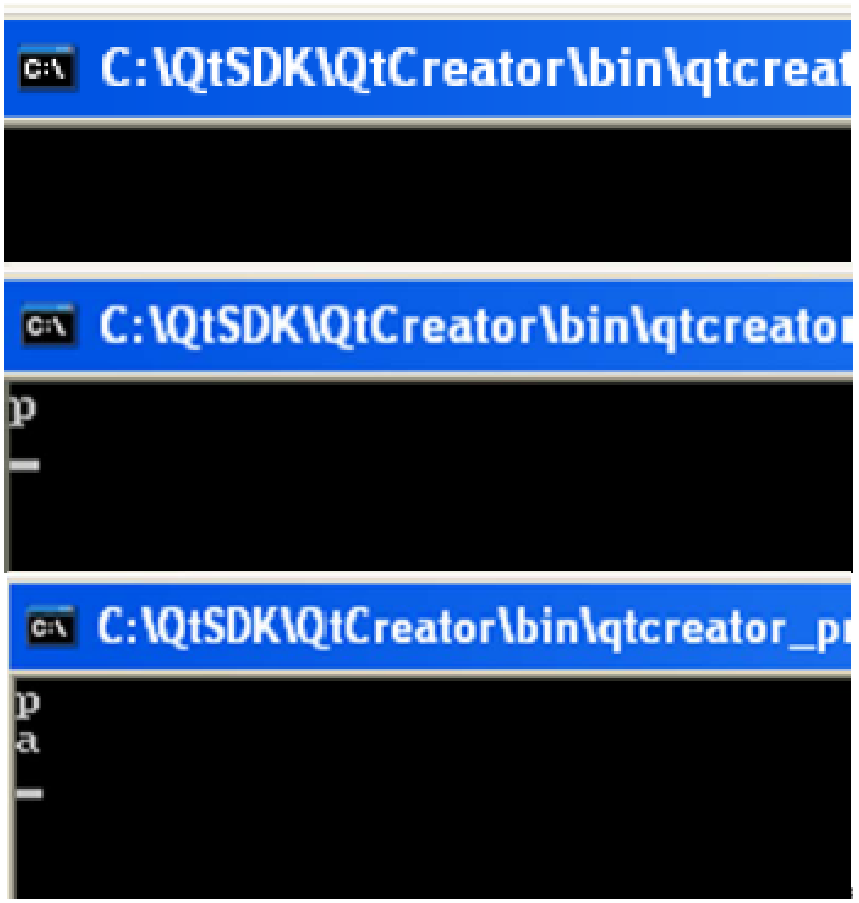
while

- If we want to execute some statements while a condition is met

```
while(expression)
{
    statement1;
}
```

while

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

int main()
{
          char c= '\0' ;
          while (c!= 't')
                    cin >> c;
          return  0;
}
```



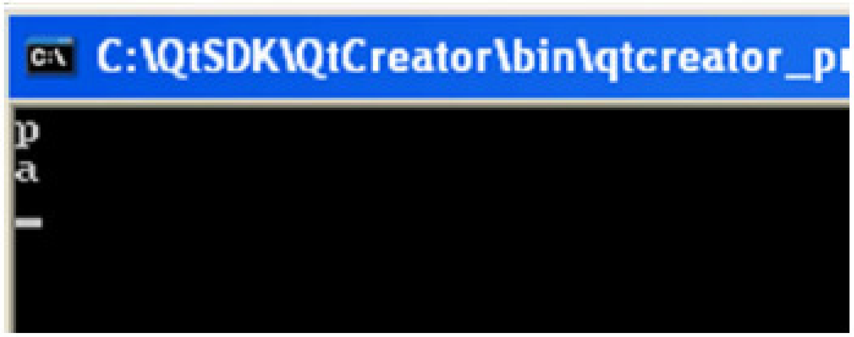
```
C:\QtSDK\QtCreator\bin\qtcreat
```

```
C:\QtSDK\QtCreator\bin\qtcreato
p
_
```

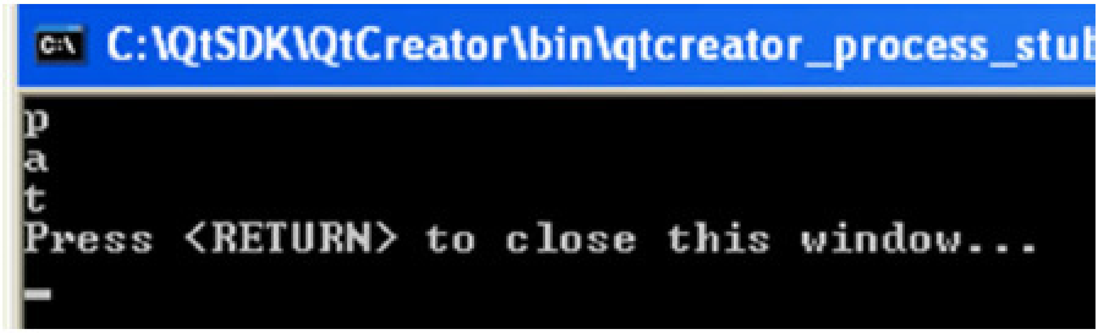
```
C:\QtSDK\QtCreator\bin\qtcreator_p
p
a
_
```

while

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

int main()
{
        char c= '\0' ;
        while (c!= 't')
                cin >> c;
        return  0;
}
```

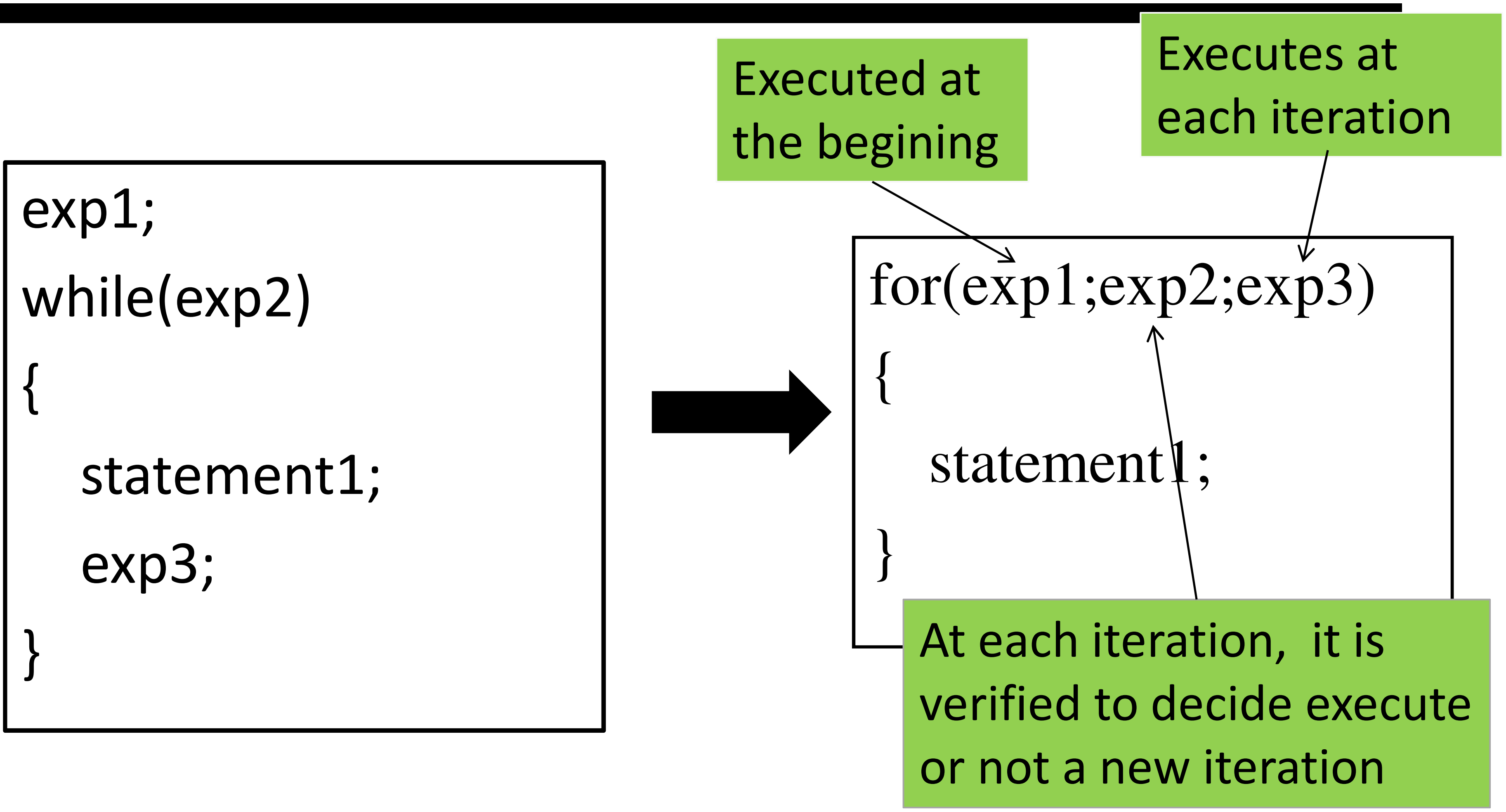


```
C:\QtSDK\QtCreator\bin\qtcreator_p
p
a
_
```



```
C:\QtSDK\QtCreator\bin\qtcreator_process_stub
p
a
t
Press <RETURN> to close this window...
_
```

for



```
exp1;
while(exp2)
{
    statement1;
    exp3;
}
```

```
for(exp1;exp2;exp3)
{
    statement1;
}
```

Executed at the begining

Executes at each iteration

At each iteration, it is verified to decide execute or not a new iteration

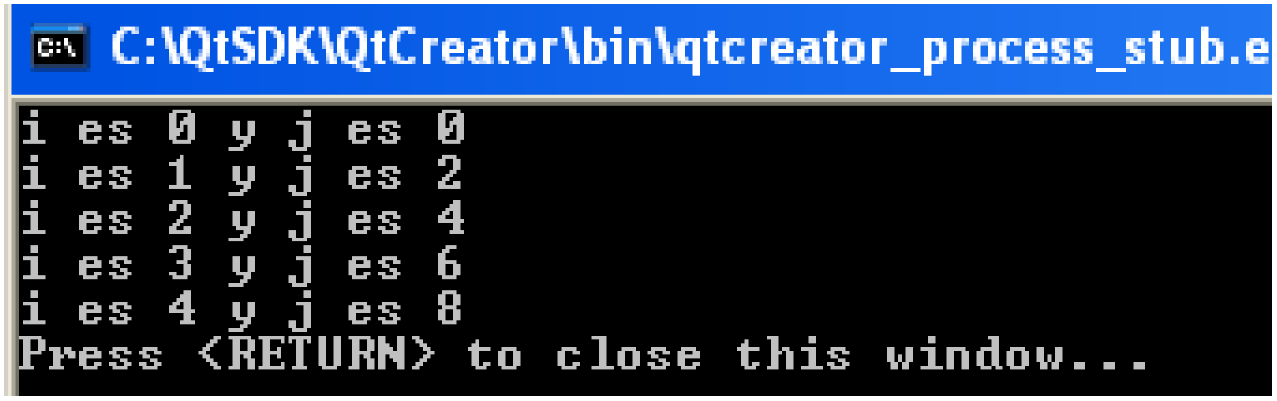
for

- The common use of “for” is a loops with a known number of iterations

```
int a[10];    test
for(i=0;i<10;++i)
{     statements
    printf(“Element %d is %d\n”,i,a[i]);
}
```

for

```
#include <iostream>
using namespace std;
int main() {
        int i,j;
        for(i=0,j=0; i<5 && j<20; i++, j+=2)
                cout<<"i es "<<i<<" j es "<<j<<endl;
        return 0;
}
```



C:\QtSDK\QtCreator\bin\qtcreator_process_stub.e

```
i es 0 y j es 0
i es 1 y j es 2
i es 2 y j es 4
i es 3 y j es 6
i es 4 y j es 8
Press <RETURN> to close this window...
```

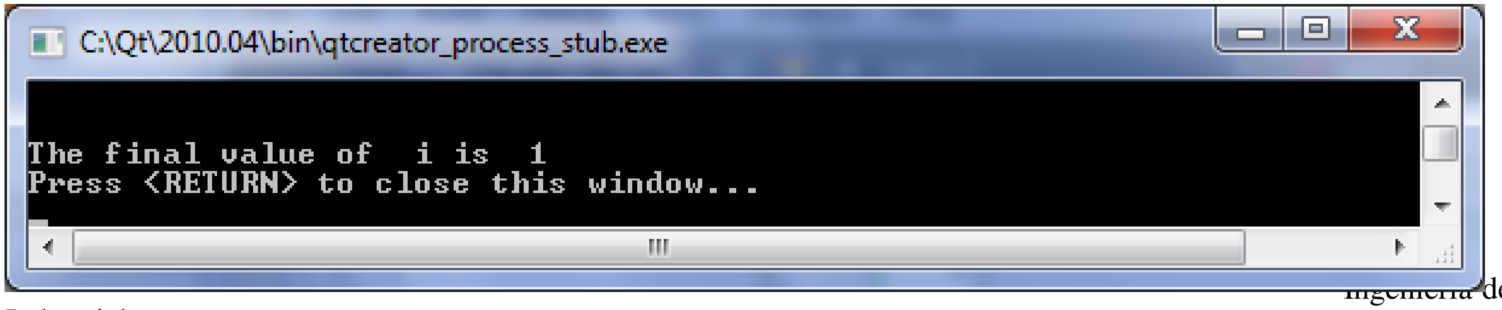
do-while

- It is similar to while, but it is used when we want to execute a set of statements at least once (even when the expression is false)

```
do
{
    sentencia1;
}
while(expression);
```

do-while

```
#include <iostream>
using namespace std;
int main() {
        int i=0;
        do{
                i+=1;
        }while(i<0);
        cout<<"The final value of  i is  "<<i<<endl;
        return 0;
}
```



C:\Qt\2010.04\bin\qtcreator_process_stub.exe

```
The final value of  i is  1
Press <RETURN> to close this window...
```

break and continue

- **break** causes the exit from the nearest loop (internal)

```
while(1)
{
    cin >> x ;
    if(x<0.0)
            break;
    else
            cout << “square root ” << sqrt(x);
 }
```

break and continue

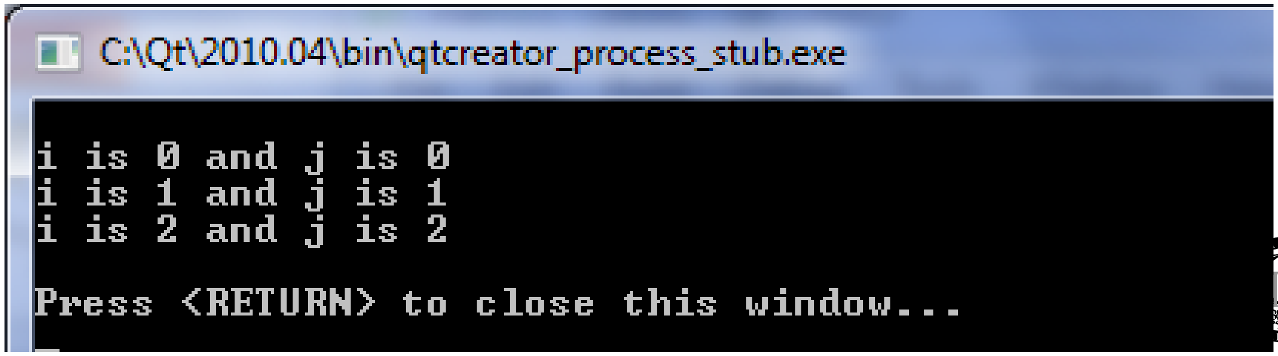
- **continue** interrupts the current iteration of the loop to go to next iteration

```
for(i=0;i<1000;i++)
{
        c=getchar();
        if(´0´<=c && c<=´9´)
                continue;
  // do something here
}
```

- Used only with **for**, **while** and **do**

break and continue

```
#include <iostream>
using namespace std;
int main() {
        int i,j=0;
        for(i=0; i<5; i++){
                cout<<"i  is "<<i<<"  and j is "<<j<<endl;
                if(j>1)
                        break;
                j++; //j+=1; o j=j+1;
        }
        return 0;
}
```

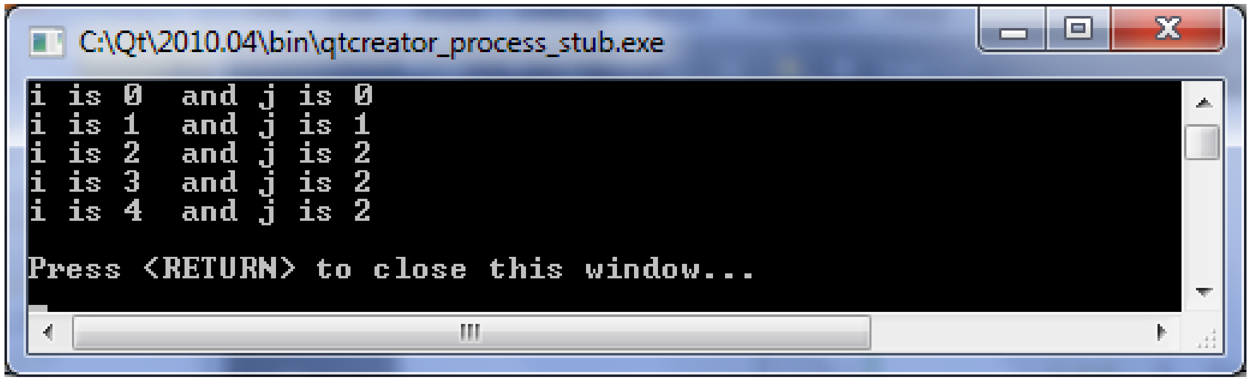


C:\Qt\2010.04\bin\qtcreator_process_stub.exe

```
i is 0 and j is 0
i is 1 and j is 1
i is 2 and j is 2

Press <RETURN> to close this window...
```

break and continue

```
#include <iostream>
using namespace std;
int main() {
        int i,j=0;
        for(i=0; i<5; i++){
                cout<<"i  is "<<i<<"  and j is "<<j<<endl;
                if(j>1)
                        continue;
                j++; //j+=1; o j=j+1;
        }
        return 0;
}
```



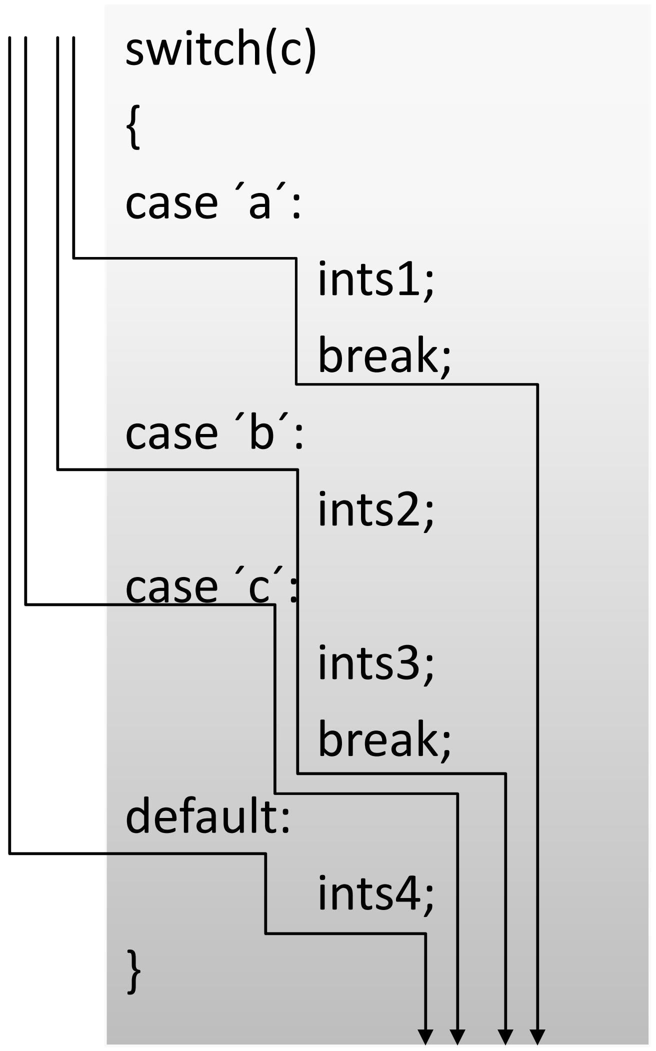
```
i is 0  and j is 0
i is 1  and j is 1
i is 2  and j is 2
i is 3  and j is 2
i is 4  and j is 2

Press <RETURN> to close this window...
```

switch

- Generalizes the use of multiple statements if-else
- Often used when several options are possible
- According to the value of a given variable one of cases is entered or in the default case.

switch

```
switch(c)
{
case ´a´:
          ints1;
          break;
case ´b´:
          ints2;
case ´c´:
          ints3;
          break;
default:
          ints4;
}
```



```
switch(c)
{
case ´a´:
          ints1;
          break;
case ´b´:
          ints2;
case ´c´:
          ints3;
          break;
default:
          ints4;
}
```

switch

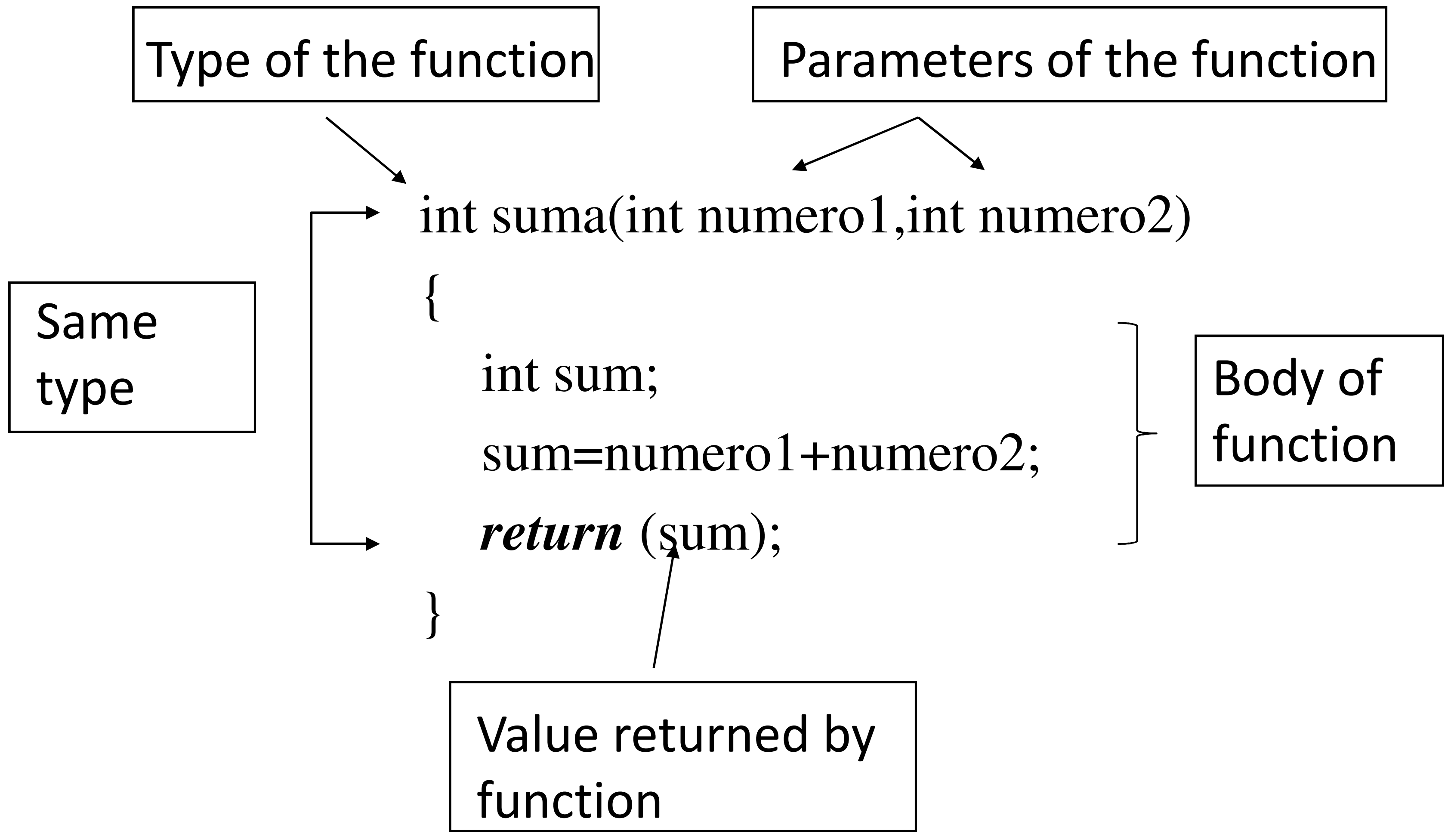
- It is typical to put a break at the end of the case to avoid entering the following case.
- It is very useful for menus with selection (“Type 1 to calculate the sum, 2 to calculate subtraction, …”).

7. Functions

Functions

- The C++ language is based on the use of functions (member function in classes):
  - System functions.
  - Device Functions .
  - Designed by the user .
- Ideas:
  - Not reinvent the wheel
  - Not repeat unnecessarily
  - Better clarity

Functions



```
int suma(int numero1,int numero2)
{
    int sum;
    sum=numero1+numero2;
    return (sum);
}
```

Function Prototype

```
int suma(int numero1,int numero2);

main()
{
    i=suma(j,6);
}

int suma(int numero1,int numero2)
{
    int sum;
    sum=numero1+numero2;
    return(sum);
}
```

Function declaration:
type name (parameters);

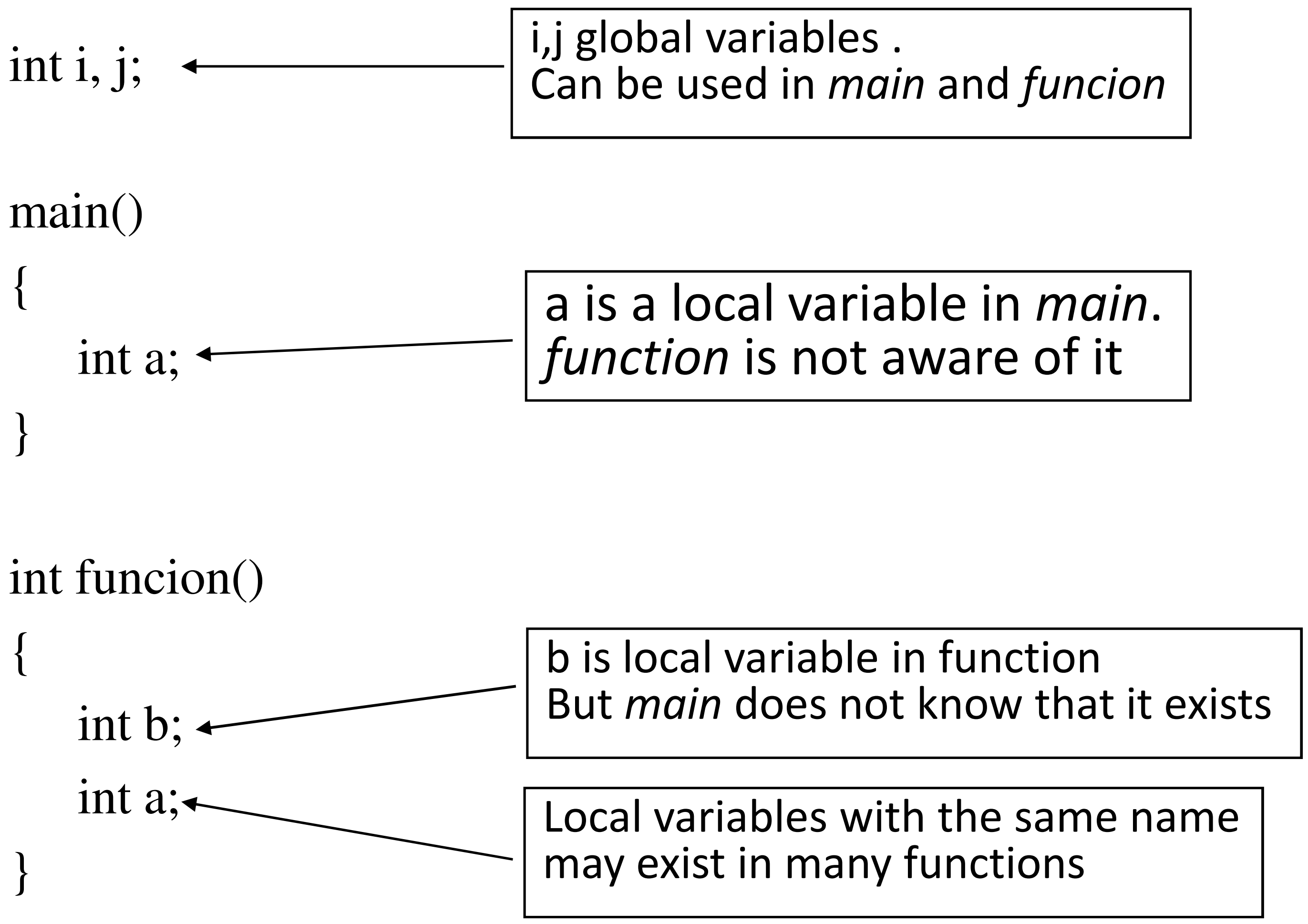
Function Call : number and type of parameters are verified

Implementation or definition of the function

Local and global variables

- The variables defined in the body of a functions are accessible only inside the function (**local** variables).
- The variables defined outside functions are known to all functions (**global** variables)

Locales and globales variables



```
int i, j;

main()
{
    int a;
}

int funcion()
{
    int b;
    int a;
}
```

i,j global variables .
Can be used in *main* and *funcion*

a is a local variable in *main*.
function is not aware of it

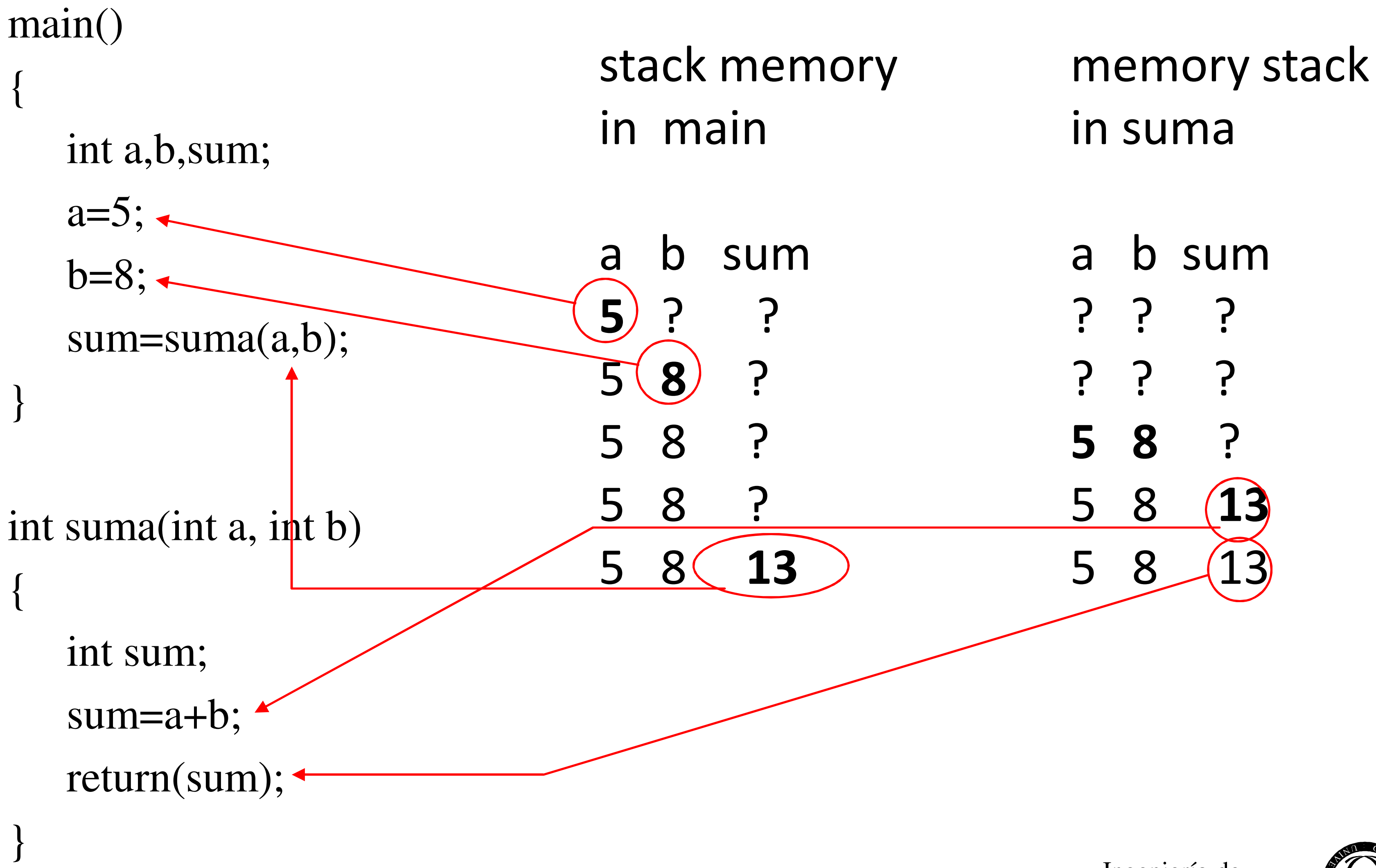
b is local variable in function
But *main* does not know that it exists

Local variables with the same name
may exist in many functions

Call by value

- A function is executed by invoking its name and parameters.
- The parameters are passed by a call by value:
  - Each argument is evaluated and its value is used locally.
  - The formal parameters is not passed to the functions (the variable)

Call by value



```
main()
{
    int a,b,sum;
    a=5;
    b=8;
    sum=suma(a,b);
}

int suma(int a, int b)
{
    int sum;
    sum=a+b;
    return(sum);
}
```

stack memory in main

a	b	sum
5	?	?
5	**8**	?
5	8	?
5	8	?
5	8	**13**

memory stack in suma

a	b	sum
?	?	?
?	?	?
5	**8**	?
5	8	**13**
5	8	13

Call by value

```
main()
{
int a,b,c;
a=5;
b=8;
c=funcion(a,b);
}
int funcion(int a, int b)
{
int sum;
++a;++b;
sum=a+b;
return(sum);
}
```

Memory stack of main

a	b	c
5	?	?
5	**8**	?
5	8	?
5	***8***	?
5	8	?
5	8	**15**

memory stack of funcion

a	b	sum
?	?	?
?	?	?
5	**8**	?
6	**9**	?
6	9	**15**
6	9	15

Arrays as arguments

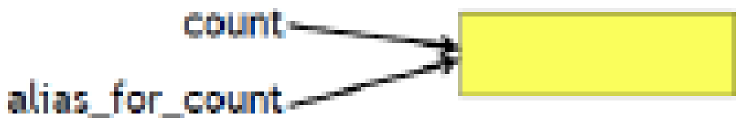
- Three forms:
  - Declare the array with dimensions
    - function(int t[10])
  - Declare the array without dimensions
    - function(int t[])
  - Will be seen in pointers
    - function(int *t)

Passing arguments by reference

- C++ accounts for a special type of a variable called “reference”, which can be used as a function parameter to refer to the original value
- A ***reference*** is an alias to another variable.
- Any change done on the reference act directly on the referenced variable
- To declare a reference to a variable, the symbol “&” is inserted just before the referenced variable
- References are “syntactic sugar” and work as pointers in truth.

Passing argument by reference

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

void main( )
{

      int count = 1;
      Int  &alias_for_count = count;
      cout << count << alias_for_count << endl;

      alias_for_count++;
      cout << count << alias_for_count << endl;

      count++;
      cout << count << alias_for_count << endl;
}
```

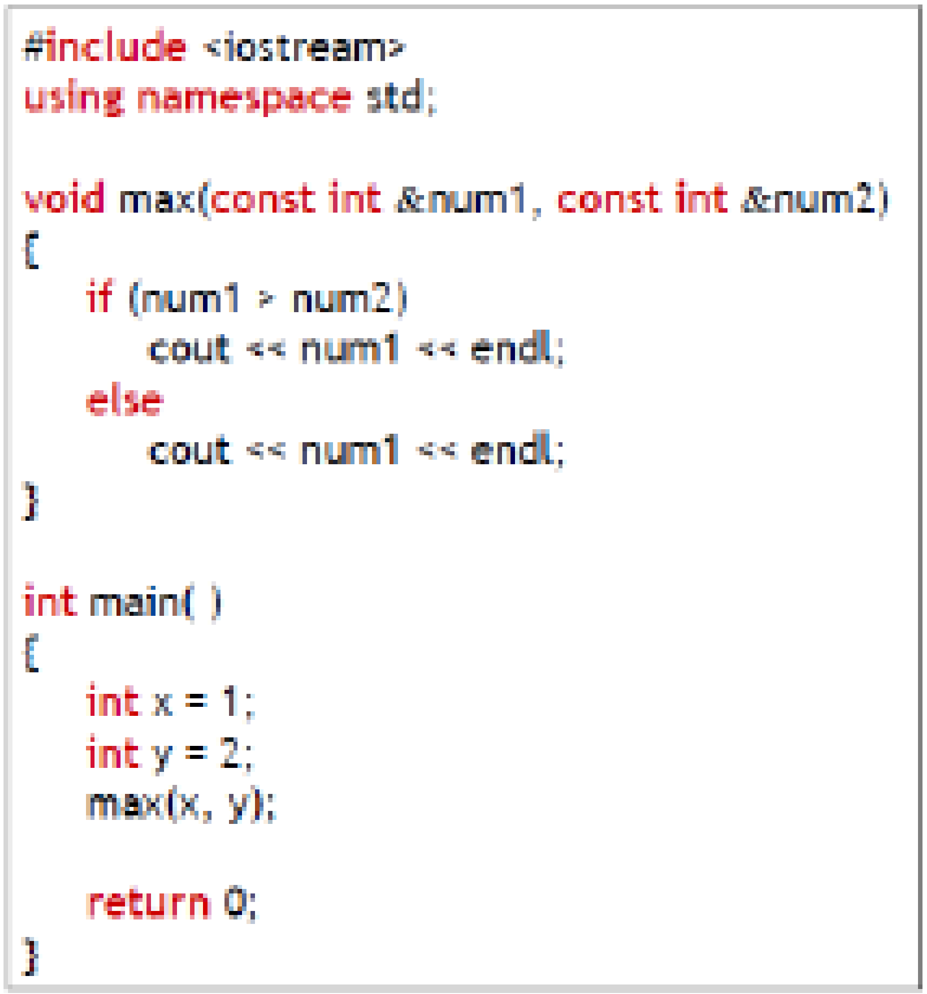


```
1   1

2   2

3   3
```

Passing argument by reference

- When arguments are passed by reference, the argument must be a variable
- When an argument is passed by value, it can a be literal, a variable, an expression or even a return value of another function

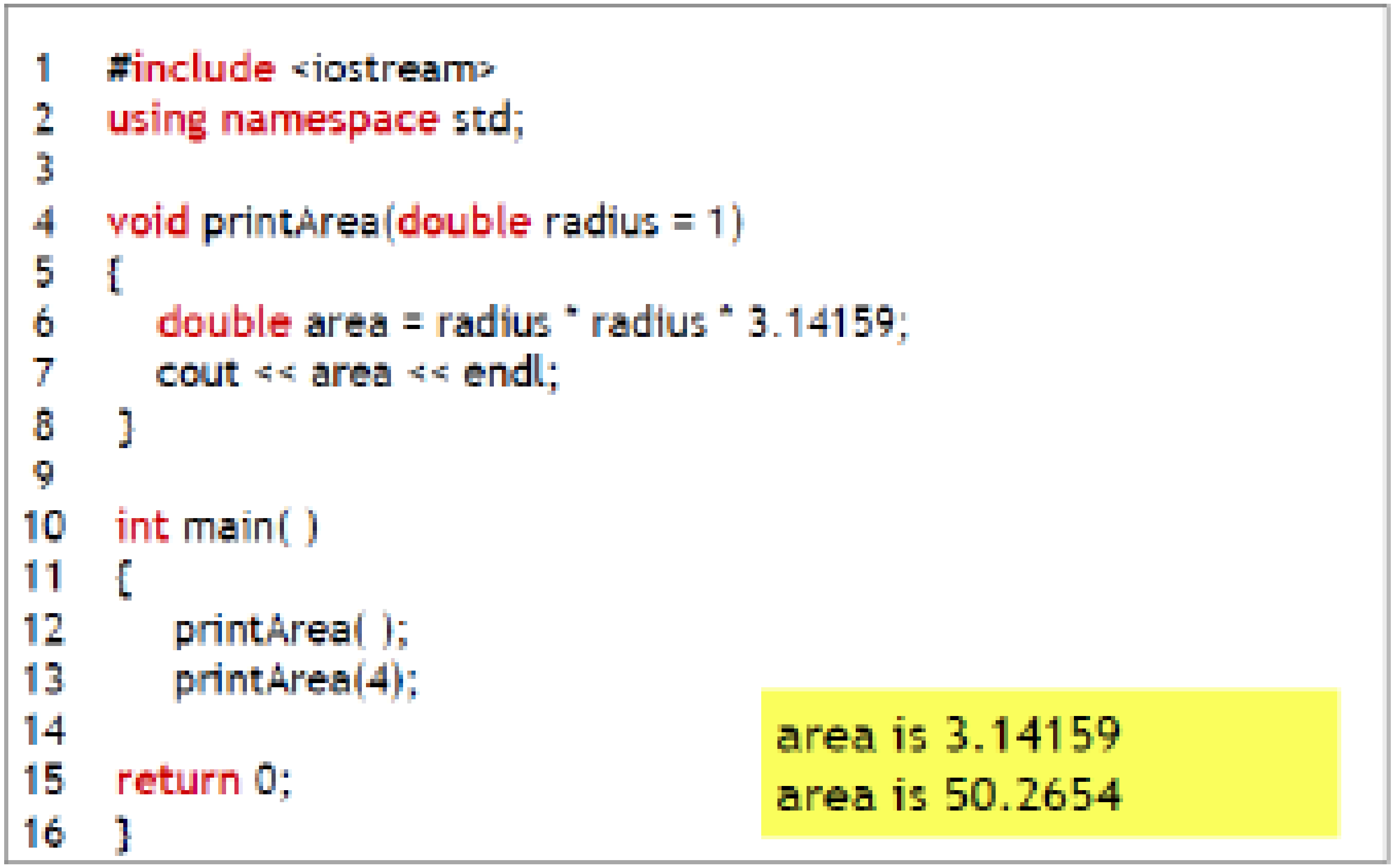
Passing argument by reference


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

void max(const int &num1, const int &num2)
{
    if (num1 > num2)
        cout << num1 << endl;
    else
        cout << num1 << endl;
}

int main( )
{
    int x = 1;
    int y = 2;
    max(x, y);

    return 0;
}
```

Argument values by default

- C++ Allows the declaration of functions with arguments values by defaults
- The by-defaults values are passed to the function if it is invoked without arguments

Arguments values by default



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

void printArea(double radius = 1)
{
   double area = radius * radius * 3.14159;
   cout << area << endl;
}

int main( )
{
    printArea( );
    printArea(4);

 return 0;
}
```

```
area is 3.14159
area is 50.2654
```

Arrays as arguments of functions

- Three formats:
  - Declaring the array with dimensions
    - function(int t[10])
  - Declaring array without dimensions
    - function(int t[])
  - Through pointers
    - function(int *t)