EEE Lecture Notes – Intro to Programming in C++

Week 1 :

No teaching for this module in Week 1 because it is Welcome Week.

Week 2 :

1.1 Getting started

What is a program, programming, coding? What is a programming language? And a compiler?

To answer these questions, let's get started.

Follow the instructions below and don't worry if there are terms you don't know or things that you don't entirely understand at this stage, it will all become clearer as we progress in the module.

Create an account on replit.com

Create a (free) account on https://replit.com/ which is an online development environment with real time collaborative coding. **Note that in free accounts the content is public** which means that anyone can see all the code on your account.

Even if you already have a C++ development environment installed on your computer, please create a replit.com account and follow the instructions below. You can keep using the environment you prefer to develop your programs but the collaborative features of replit are useful for example in case we want to discuss your code over a call on Teams and it makes it easier to keep all your code.

Regardless of whether you use replit or another development environment to develop your programs you should in any case keep all the programs you write for this module as we will ask to see them.

Create a C++ repl

REPL means “read–eval–print loop” and in this case it indicates the environment in which you will write and run your programs.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=300&tok=3b6f57&media=progeng2324:replit00.png

If you don't see the sidebar menu after logging in on replit.com, click on this icon.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=300&tok=1bfad1&media=progeng2324:replit1.png

Click on “Create Repl”.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=600&tok=0dfeb3&media=progeng2324:replit2.png

Search and select C++, give a name to your repl (for example the first letter of your first name followed by your surname) and confirm.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=800&tok=d37e14&media=progeng2324:replit3b.png

There will be a simple program already there. Click on “Run” to run it.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=600&tok=82fb33&media=progeng2324:replit4.png

You will see the result of the execution in the Console.

How to keep our programs when we write a new one

For our purposes the most practical approach is to keep all the code for all the programming tasks in the same repl (not just the same account, the very same repl).

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=250&tok=149a6f&media=progeng2324:replit6.png

“Add folder”.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=250&tok=70746c&media=progeng2324:replit7.png

Give a name to the folder, for example progressive numbers for each week of term.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=250&tok=e3d731&media=progeng2324:replit8.png

“Add a file”.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=600&tok=940dd7&media=progeng2324:replit9.png

Give a name to the file. Please note that **for files that we want to keep as archive (and not execute) it's important that the extension is NOT “.cpp”, I advise you use “.hpp”**.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=600&tok=1f1017&media=progeng2324:replit10.png

Cut and paste the previous program from “main.cpp” to the new file (in this case “hello.hpp”). There is no “save” button because programs are saved as you type.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=600&tok=c46e5c&media=progeng2324:replit11.png

Write a new program in the “main.cpp” file. For example in this case:

```
#include <iostream>

#include <string>


int main(){

    std::string name;

    std::cout << "what is your name?" << std::endl;

    std::cin >> name;

    std::cout << "hello, " << name << std::endl;
```

}

Don't worry if you do not understand the code at this stage! For now we are only concerned with testing the development environment (you can read the explanation of the code later on in section 1.2 Basic Input/Output, Text Variables).

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=800&tok=a6b370&media=progeng2324:replit12.png

Click on “Run” to run this program. In this case the program reads an input from the user, so I wrote my name (Max) in the console when prompted.

How to get a join link for the repl to enable collaborative editing

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=600&tok=11122e&media=progeng2324:invite1.png

Click on “Invite” and click on the switch to enable and generate a join link.

---

https://intranet.ee.ic.ac.uk/m.cattafi/scripts/dw/lib/exe/fetch.php?w=600&tok=9bdc10&media=progeng2324:invite2.png

Once the join link is generated you can copy it. **Please note that anyone with this link will be able also to edit your repl.**

1.2 Basic Input/Output, Text Variables

Printing

Hello

We begin with a simple program which just prints “hello” on the screen:

```
#include <iostream>



int main(){
```

```
    std::cout << "hello";

}
```

The meaning of the code line by line is explained below. In C++ lines that begin with `//` are comments that we can add without interfering with the actual code.

```
#include <iostream>



// includes the header iostream

// (input and output stream)

// headers are often needed in addition

// to the core language

// in this case the header is needed

// in order to perform input and output



int main(){

// begins a function called main

// functions will be covered later on

// for now we always include our instructions

// into something beginning with int main(){



    std::cout << "hello";



    // performs the actual printing
```

```
    // read it right to left:
    // the text "hello"
    // is printed (<<)
    // on std::cout

    // cout stands for console output
    // and it represents the screen
    // (std stands for standard,
    // we will see later on why)

    // note that the text needs to be
    // enclosed in quotes (")

    // note the ; at the end
    // C++ instructions end with a ;

}
// this curly brace ends function main
```

Exercise: hello everyone

Edit the program above so that it prints something else instead of “hello”. For example “hello everyone”.

Hello Goodbye

We now want to write a program that prints “hello” followed by “goodbye” on a different line, something like this:

```
hello

goodbye
```

We may think this is how we do it:

```
#include <iostream>



int main(){

    std::cout << "hello";

    std::cout << "goodbye";

}
```

But this will not actually work as intended.

Exercise: lack of newline

Run the program above to confirm that it doesn't work as intended.

In order to achieve the intended result we need to include an instruction printing a new line like this:

```
#include <iostream>
```

```cpp
int main(){

    std::cout << "hello";

    // we print hello




    std::cout << std::endl;

    // we print a newline

    // std::endl represents a new line

    // endl stands for endline

    // again std is for standard



    std::cout << "goodbye";

    // we print goodbye

}
```

Exercise: blank line

Edit the program above so that it prints something like this (there is one blank line between “hello” and “goodbye”):

```
hello



goodbye
```

Usually we include an end of line at the end of almost all our printing instructions. The reasons for this will become clearer later on. If you want a preview of the reason (and of the difference between the newline with `std::endl` as opposed to `\n`) you can read the discussion on https://stackoverflow.com/questions/213907/stdendl-vs-n. For now let's take the habit of writing our printing instructions like this:

```
#include <iostream>



int main(){

    std::cout << "hello" << std::endl;

    // we print hello, then print a newline

    // this is equivalent to

    // std::cout << "hello";

    // std::cout << std::endl;



    std::cout << "goodbye" << std::endl;

    // we print goodbye, then print a newline

}
```

And this will print:

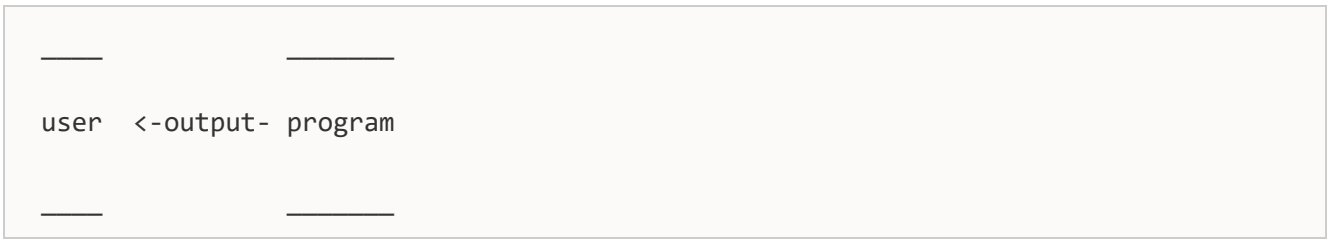
```
hello

goodbye
```

Exercise: test the program

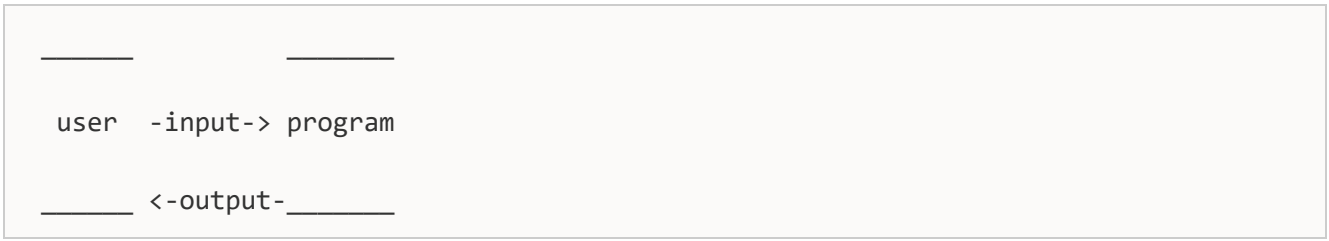
Run the program above to confirm that it works as intended.

Reading (into a variable) and printing

So far we have seen examples of programs producing an output to the user by printing on the screen.



Usually a program will interact with the user and get also some information in input (and store it into a **variable**) that will influence what's the output of the program.



Hello, you

The following program asks for your name and then greets you.

```cpp
#include <iostream>

#include <string>



int main(){

    std::string user_name;

    std::cout << "what is your name?" << std::endl;

    std::cin >> user_name;

    std::cout << "hello, " << user_name << std::endl;

}
```

The explanation line by line is provided below.

```
#include <iostream>

#include <string>

// this includes the header for text variables



int main(){



    std::string user_name;

    // this is the declaration of a variable:

    // "user_name" is the name of this variable

    // std::string is the type of the variable



    // variables are like boxes that contain information

    // type std::string variables can contain text



    std::cout << "what is your name?" << std::endl;

    // this is just a normal printing instruction



    std::cin >> user_name;

    // this instruction reads information from the keyboard

    // (std::cin stands for console input)

    // and stores it in variable "user_name"
```

```
    std::cout << "hello, " << user_name << std::endl;
    // this printing instruction does the following:
    // prints the fixed text "hello, "
    // prints the content of variable user_name
    // prints a newline
}
```

Exercises

Fix Hello

The following program has several issues, fix them all and test it.

```
// this program needs to be fixed

#include <iostring>

int main(){
    std::cout >> "hello"
```

Name and Surname

Complete and test the following program.

```
#include <iostream>
#include <string>
```

```
// this includes the header for text variables


int main(){


    std::string user_name;


    // TODO: add another declaration

    // for a text variable called user_surname


    std::cout << "what is your name?" << std::endl;

    std::cin >> user_name;


    // TODO: add another printing instruction

    // asking for the surname


    // TODO: read the surname from the user

    // and store it in variable user_surname


    // TODO: add a printing instruction

    // that says hello followed by the

    // name and surname of the user

}
```

1.3 Numerical Variables, Expressions

Adding two numbers

The following program reads two numbers from the user and prints their sum.

```
#include <iostream>



int main(){

    double n1, n2, sum;

    // this line declares three variables of a numerical type

    // n1, n2, sum are the names of the variables

    // double is the type, which represents general decimal numbers



    std::cout << "please enter the first number" << std::endl;

    std::cin >> n1;



    std::cout << "please enter the second number" << std::endl;

    std::cin >> n2;



    sum = n1 + n2;

    // here the result of n1 + n2 is computed

    // and it is stored in variable sum



    std::cout << n1 << " + " << n2 << " = " <<  sum << std::endl;
```

```
    // this is a printing instruction printing

    // the value in n1

    // the symbol + (with spaces)

    // the value in n2

    // the symbol = (with spaces)

    // the value in sum

    // (and a newline at the end as usual)

}
```

Exercise: multiplying two numbers

Edit the program above so that it computes and print the product of the two numbers in input instead of their sum.

What does = really mean in C++?

The `=` in C++ is not like the `=` in maths and this can cause some confusion at the beginning.

In maths if we write

`c = a + b`

this is for example equivalent to

`a + b = c`.

In C++ however

`c = a + b;`

means “compute the result of a + b and store it in c”

whereas

`a + b = c;`

is not a valid instruction.

In C++ it's better not to read `c = a + b;` as “c is equal to a + b”, rather read it as “assign [the value of] a + b to c” and mentally it may be helpful to visualise it as an arrow going from right to left such as `c ← a + b`.

Exercise: understanding =

Consider the program below, follow the instructions in the comments and test the program.

```
#include <iostream>



int main(){

    double a, b, c;



    a = 1;

    b = 2;

    c = a + b;



    std::cout << c << std::endl;

    // this line will print 3



    a = 2;
```

```
    std::cout << c << std::endl;

    // TODO: before testing the program

    // write in a comment below

    // what you expect to be printed


    c = a + b;


    std::cout << c << std::endl;

    // TODO: before testing the program

    // write in a comment below

    // what you expect to be printed


}
```

Integers

Type `double` is the numerical type for general integer number. In C++ there is also a type `int` for integer numbers. It will become clearer later on why it is important to have a different type for integer numbers.

For now consider the following example which reads an integer from the user and detects whether it is even or odd.

```
#include <iostream>



int main(){
```

```
    int n, rem;



    std::cout << "please enter a number" << std::endl;

    std::cin >> n;



    rem = n % 2;

    // the operator % is not a percentage (bad choice of symbol, I know)

    // it computes the remainder of the integer division

    // between n and 2, which will be

    // 0 if the number is even and

    // 1 if the number is odd



    std::cout << "in the following line 0 means even and 1 means odd" <<
std::endl;

    std::cout << rem << std::endl;



    // later we will upgrade this example to instead print a message saying

    // "the number is even" or "the number is odd"

    // depending on the value of the remainder

}
```

Exercise: using the right type

Change the type of the variables in the program above from `int` to `double` to confirm that in that case the program would not run.

This is because the remainder of the integer division in C++ can only be computed if the numbers are of type `int`.

Unexpected Integers

What would you expect the following program to print?

```
#include <iostream>



int main(){

    std::cout << 5 / 2 << std::endl;

}
```

If you answered `2.5` that's understandable but not the correct answer.

Exercise: testing unexpected integers

Test the program above to confirm that it doesn't print `2.5`.

This is because the `5` and the `2` in the division are interpreted as integers, therefore you get the quotient of the integer division.

In order for it to be interpreted as a division with decimals at least one of the operands needs to be interpreted as non-integer. For example as in the following code.

```
#include <iostream>
```

```
int main(){

    std::cout << 5 / 2.0 << std::endl;

}
```

Exercise: testing the correct version

Test the program above to confirm that it does print `2.5` as we would expect.

Exercises

Rectangles

Write a program that reads from the user the length of two (different) sides of a rectangle and prints its perimeter and area.

Currency Conversion

Write a program which reads from the user

- an amount of money in British Pounds
- the current exchange rate to Euros

and prints the equivalent amount in Euros.

Temperature Conversion

Write a program which reads from the user a temperature in Celsius degrees and prints the equivalent temperature in Fahrenheit degrees.

2.1 Conditionals

Even or Odd

Let's upgrade the “even or odd” example. The previous version is shown below as a reminder.

```
#include <iostream>
```

```cpp
int main(){
    int n, rem;

    std::cout << "please enter a number" << std::endl;
    std::cin >> n;

    rem = n % 2;

    std::cout << "0 means even and 1 means odd:" << std::endl;
    std::cout << rem << std::endl;

    // later we will upgrade this example to
    // instead print a message saying
    // "the number is even" or "the number is odd"
    // depending on the value of the remainder
}
```

What we would like to implement is along the lines of:

```
if the remainder of the integer division of n by 2 is 0 then:
    print that the number is even
otherwise:
```

```
    print that the number is odd
```

We can implement this in C++ using **conditionals** (also known as **if statements** or **if-else statements**), as in the following example.

```cpp
#include <iostream>



int main(){

    int n, rem;



    std::cout << "please enter a number" << std::endl;

    std::cin >> n;



    rem = n % 2;



    if(rem == 0){

        // if the remainder is 0...

        std::cout << "the number is even" << std::endl;

    }

    else{

        // otherwise...

        std::cout << "the number is odd" << std::endl;

    }

}
```

In this code:

```
    if(rem == 0){

        std::cout << "the number is even" << std::endl;

        // this part is called the if branch

    }

    else{

        std::cout << "the number is odd" << std::endl;

        // this part is called the else branch

    }
```

The part inside the parentheses of the `if()` is called the **condition** (not the conditio**nal**). The condition must contain something that represents a yes/no (or true/false) value or expression. If the answer is yes (or true) then the code in the “if branch” is executed, otherwise the code in the “else branch” is executed.

Exercise: multiple of 3 or not

Edit the program above so that it detects whether the number entered by the user is a multiple of 3. For example:

```
please enter a number

6

the number is a multiple of 3

please enter a number

11

the number is not  multiple of 3
```

What is == and what is the difference with = ?

The `==` sign **asks a yes/no (or true/false) question**. In the example above the question is: “is the value stored in `rem` equal to 0?” and the answer will be yes or no, depending on the value initially in `n` and therefore on the value of the remainder of the division of `n` by 2.

This is different from the `=` sign which assigns a value to a variable. A common mistake is to use `a = ` instead of `a ==` inside the condition of an “if statement”. This may not be detected as unintended by the compiler and the program will simply not work as expected, so make sure you double check.

Two variations on Even or Odd

A longer version

```
#include <iostream>



int main(){

    int n, rem;



    std::cout << "please enter a number" << std::endl;

    std::cin >> n;



    rem = n % 2;



    bool is_even;

    // yes/no (true/false) values have also a type, called bool



    is_even = (rem == 0);
```

```
    // read this right to left:
    // is the value of rem equal to 0?
    // store the corresponding yes/no answer
    // in the bool variable is_even

    // we wouldn't normally write this program like this
    // this is just to show what is going on with the yes/no values
    // and it will become more useful later on

    if(is_even){
        // in this condition we use a
        // yes/no (bool) value
        // previously computed
        // instead of a yes/no (bool) expression
        // as in the previous example

        std::cout << "the number is even" << std::endl;
    }
    else{
        std::cout << "the number is odd" << std::endl;
    }
}
```

This example introduces variables of type `bool`, which is short for Boolean, named after George Bool and his Boolean algebra. Bool(ean) variables, values, and expressions are yes/no or, more properly, true/false.

A shorter version

```
#include <iostream>



int main(){

    int n;



    std::cout << "please enter a number" << std::endl;

    std::cin >> n;



    // this version does two things within the condition of the if:

    // it first computes the remainder of the division of n by 2

    // then checks whether it is equal to 0 or not



    if((n % 2) == 0){



        std::cout << "the number is even" << std::endl;



    }

    else{
```

```
        std::cout << "the number is odd" << std::endl;


    }
}
```

This example is how we would normally write a program like this, just more concise than our initial one.

Exercise: Absolute Value

Informally we can say that the absolute value of a real number `n` (that we indicate as `abs(n)` or `|n|`) is `-n` (i.e. we change the sign of `n`) if `n` is negative, or the same number `n` if it's non-negative:

```
if n is less than 0 then:

    the absolute value of n is -n

otherwise:

    the absolute value of n is n
```

Complete and test the following code:

```
#include <iostream>



int main(){

    double n, absv;

    // n for the input, absv will be

    // the absolute value we compute
```

```
    // TODO: ask for a number in input
    // TODO: read the input number in n

    if(n < 0){ // yes/no question: is n less than 0?
        // if yes, its absolute value is the number changing the sign
        absv = -n;
    }
    else{
        // if not, its absolute value is the same as n
        // TODO: assign the value of n to absv
    }

    std::cout << "|" << n << "| = " << absv << std::endl;

}
```

If without else

Note that an if statement doesn't need to have also an else. This can be shown in the following variation on the previous example:

```
// this is not meant to be a good example to imitate!

#include <iostream>
```

```cpp
int main(){
    double n;
    // only one variable for both input and output

    std::cout << "please enter a number" << std::endl;
    std::cin >> n;

    if(n < 0){
        n = -n;
        // if n is negative we change its sign
    }
    // otherwise we don't have to do anything

    // we print the absolute value
    std::cout << "absolute value: " << n << std::endl;
}
```

As written in the comment at the top of the code, this is not necessarily a good example of how to write programs. Often it's better to have separate variables for the input and for the output as it makes the code more readable and there isn't a gain from having fewer variables in this case.

Devise exercises involving variables, expressions, operators, conditionals

Now that you have practised using variables, expressions, operators, conditionals apply your creativity to devise some exercises that involve these concepts (even better if it's something that can have a practical purpose) and implement the solutions.

Week 3 :

2.2 Case study with boolean operators: Unit Conversion

In this section we consider how to incrementally develop a program by adding features that use in particular boolean operators.

We want to write a program that converts miles to km and the other way around. The conversion rate is `1 mile = 1.609 km`.

We want the program to work as shown in the examples below, in which you can also notice that:

- We don't include a prompt to the user asking for input (just to keep it concise). So the user just enters the quantity to convert followed by the unit and gets in output the result of the conversion to the other unit.
- For now we don't use the plural “miles” for the unit name, but always just “mile”.
- There will be approximation errors (and that's ok).

```
1 mile

1.609 km

16.3 mile

26.2267 km

3.2 km

1.98881 mile
```

The implementation of the program can be as follows:

```
#include <iostream>

#include <string>
```

```cpp
int main(){

    double length_in, length_out;
    std::string unit_in, unit_out;

    // we declare variables for the input and the output
    // the length is a number, the unit name is text

    const double mile_to_km = 1.609;
    // useful way to define constants

    std::cin >> length_in >> unit_in;
    // we don't print a prompt, we just read the input

    if(unit_in == "km"){
        unit_out = "mile";
        length_out = length_in / mile_to_km;
    }
    else{
        unit_out = "km";
        length_out = length_in * mile_to_km;
```

```
    }


    std::cout << length_out << " " << unit_out << std::endl;

}
```

Exercise: test unit conversion

Test the program above.

Exercise: test unit conversion with wrong units

Note that since in the conditional included in the program above we are only checking for `km`, anything other than `km` will be assumed to be `miles`, for example:

```
1 litre

1.609 km
```

Test the unit conversion example program above with “wrong units” in input (i.e. anything either than km or mile) to check that this is indeed the behaviour.

Exercise: temperature conversion

Write a temperature conversion program that can convert from Celsius to Fahrenheit and the other way around. The program should read a number and a unit and perform the conversion accordingly, for example:

```
451 F

232.78 C

100 C
```

```
212 F
```

One more check

If we want we can add one more check to the conditional in our initial program:

```
#include <iostream>

#include <string>



int main(){



    double length_in, length_out;

    std::string unit_in, unit_out;



    const double mile_to_km = 1.609;



    std::cin >> length_in >> unit_in;



    if(unit_in == "km"){

        unit_out = "mile";

        length_out = length_in / mile_to_km;



        std::cout << length_out << " " << unit_out << std::endl;
```

```
    }
    else if(unit_in == "mile"){
        unit_out = "km";
        length_out = length_in * mile_to_km;

        std::cout << length_out << " " << unit_out << std::endl;
    }
    else{
        std::cout << "error, unit not recognised" << std::endl;
    }

}
```

This example shows how to write an if statement with three branches (or three different outcomes). We can add any amount of branches keeping this structure.

You may have heard (or may come across in the future) of something called the “switch” statement that does something similar. Some people like it, some people don't. In my opinion this structure of if-else if-…-else statements is generally a better choice than the switch statement, in particular if you are using text variables. So I won't introduce the switch statement in these notes.

Exercise: test unit conversion error message

Test the unit conversion example program above with inputs other than km or mile for the unit name to check that you would now indeed get the `error, unit not recognised` message.

Removing duplicated lines

In the example above, the line `std::cout « length_out « " " « unit_out « std::endl;` is duplicated. Having the exact same line duplicated is usually an indication that the code should be given a better, more elegant, structure. For example because if we want to change the format of the output we would need to do it twice instead of just once. The following example addresses this issue:

```
#include <iostream>

#include <string>



int main(){



    double length_in, length_out;

    std::string unit_in, unit_out;



    const double mile_to_km = 1.609;



    // stage 1: input



    std::cin >> length_in >> unit_in;



    bool valid_unit = true;

    // we assume the user will input a valid unit



    // stage 2: data processing
```

```
if(unit_in == "km"){
    unit_out = "mile";
    length_out = length_in / mile_to_km;

}
else if(unit_in == "mile"){
    unit_out = "km";
    length_out = length_in * mile_to_km;
}
else{
    valid_unit = false;
    // if the user inputs an invalid unit name
    // we update this variable
}

// stage 3: output

if(valid_unit){
    std::cout << length_out << " " << unit_out << std::endl;
}
else{
```

```
        std::cout << "error, unit not recognised" << std::endl;

    }



}
```

As you can see in this example, we can assign values like `true` and `false` to variables of type `bool`.

Now the program is more elegant and it is neatly divided into the three fundamental stages of input, processing, and output.

Exercise: restructure temperature conversion

Upgrade and restructure your temperature conversion program following the example above so that it includes a check and error message if the temperature unit in input is something other than `C` or `F`. For example:

```
1 c

error, unit not recognised

1 litre

error, unit not recognised
```

Unit Conversion with boolean operators

Finally, maybe we should allow the user to also enter “miles” as input as it's a common way of writing the unit.

```
#include <iostream>

#include <string>
```

```
int main(){


    double length_in, length_out;

    std::string unit_in, unit_out;


    const double mile_to_km = 1.609;


    std::cin >> length_in >> unit_in;


    bool valid_unit = true;


    if(unit_in == "km"){

        unit_out = "miles";

        length_out = length_in / mile_to_km;


    }

    else if( (unit_in == "mile") || (unit_in == "miles") ){ // the || means "or"

        unit_out = "km";

        length_out = length_in * mile_to_km;

    }

    else{

        valid_unit = false;
```

```
    }




    if(valid_unit){

        std::cout << length_out << " " << unit_out << std::endl;

    }

    else{

        std::cout << "error, unit not recognised" << std::endl;

    }

}
```

In this example we introduce the boolean operator `||` which means “or”. There are also `!` for “not” and `&&` for “and”. These operators can be used to build boolean expressions.

The `!` (“not”) is a unary operator. It takes a boolean value and inverts it: `true` becomes `false` and `false` becomes `true`.

The `||` (“or”) is a binary operator. It takes two boolean values and the result is `false` if they are both `false` and `true` in all other cases.

The `&&` (“and”) is a binary operator. It takes two boolean values and the result is `true` if they are both `true` and `false` in all other cases.

Exercise: temperature conversion with boolean operators

Upgrade your temperature conversion program so that it work regardless of whether the unit entered is lowercase or uppercase (an error message should still be printed for any unit other than `F`, `f`, `C`, `c`). For example:

```
451 F

232.78 C
```

```
451 f

232.78 C

100 C

212 F

100 c

212 F

1 litre

error, unit not recognised
```

3.1 While Loops

Computing x^n

So far we have seen two of the three components we need to implement algorithms in structured programming: sequential execution of instruction and conditionals (“if statements”). The third one is loops.

To illustrate loops let's consider the algorithm that we can use to compute `x^n`, i.e. `x power n`, with `n` in this case being an integer greater than or equal to 0. We know that this can be computed by repeated multiplications, for example `2^3 = 2 * 2 * 2`.

```
to compute x^n:



initialise a variable e.g. "temp" to 1



do the following step n times:

    multiply temp times x, and store the result back in temp
```

```
at the end the number in temp will be x^n
```

So first of all let's see how to do something n times.

Doing something n times

The following program reads a number `n` from the user (there is no input prompt to keep it concise) and prints “hello” n times.

```
#include <iostream>



int main(){

    int n, counter = 0;

    // we have a variable counter

    // counting how many times we print

    // initially we have printed 0 times



    std::cin >> n;



    // the following line begins a while loop

    // the while loop has a condition in parentheses,

    // if the condition is true, then we enter the loop

    // and execute the instructions inside

    // otherwise the execution goes past the loop



    while(counter < n){
```

```
        std::cout << "hello" << std::endl;


        counter = counter + 1;
        // after each print we increase
        // the counter by 1
    }
}
```

Exercise: rename counter

The name `counter` for the variable is not a special name, it's just the name of a variable. Replace all the occurrences of counter with a different variable name, for example `i` which will be a typical name in the future, and check that the program still works.

Tracing the execution

The program above can be traced more or less like this:

```
initially the counter is 0

let's assume we read 3 from the user, stored in n



we get to the loop for the first time, the question is:

is 0 less than 3?
```

```
yes so we enter the loop and print:

hello



then counter becomes 1 (0 + 1)



now we go back to the top of the loop, the question is:

is 1 less than 3?



yes so we enter the loop and print:

hello



then counter becomes 2 (1 + 1)



we go back to the top of the loop, question:

is 2 less than 3?



yes so we enter the loop and print:

hello



then counter becomes 3 (2 + 1)



we go back to the top of the loop, question:
```

```
is 3 less than 3?



no! so we move forward beyond the loop

to the end of the program
```

Exercise: print counter

Consider the following variation in which instead of printing “hello” we print the value of variable counter every time the loop is executed and n is set to 4 and not read from the user. Read the code and make a note somewhere of what you expect this program to print. Then test the program to check your expectation.

```
#include <iostream>



int main(){

    int n=4, counter = 0;



    while(counter < n){

        std::cout << counter << std::endl;

        counter = counter + 1;

    }

}
```

There are good reasons to keep the counting like this and this will become clearer later on.

Doing something infinite times

It is very common to accidentally write a loop in such a way that it keeps looping forever (so called infinite loop). Usually this is because one forgets to add an instruction that at some point changes the condition of the loop from `true` to `false`.

Keep in mind that the program keeps entering the loop if the condition is `true`, therefore if it enters the loop once and the loop doesn't contain any instructions that change the condition, the condition will always stay `true` and therefore the loop will never terminate.

In the example above this would look more or less like this:

```
// WARNING: infinite loop!

// don't run this program if it's not the right time

// to have an infinite loop running on your computer!



// depending on the tools you are using this may crash

// your development environment, your browser, other programs

// or the whole operating system



// before you run this program, make sure that all your work is saved

// that you know how to stop a program manually etc



#include <iostream>



int main(){

    int n, counter = 0;
```

```
    std::cin >> n;



    while(counter < n){

        std::cout << "hello" << std::endl;



        // if we remove or forget to include

        // the instruction updating counter

        // if the condition is true the first time

        // it will always stay true and the loop never ends



    }

}
```

Exercise: infinite loop

If it's the right time to do so (you have saved your work, you know how to stop a program manually etc) test the program above to confirm it keeps printing “hello” and never stops. You will have to force the stop manually.

Back to computing x^n

Previously we introduced the following algorithm sketch:

```
to compute x^n:



initialise a variable e.g. "temp" to 1
```

```
do the following step n times:
    multiply temp times x, and store the result back in temp

at the end the number in temp will be x^n
```

We can now write it in code:

```
#include <iostream>

int main(){

    double x, temp = 1;
    int n, counter = 0;

    std::cin >> x >> n;

    while(counter < n){
        temp = temp * x;
        counter = counter + 1;
    }

    std::cout << x << "^" << n <<  " = " << temp << std::endl;
```

```
}
```

Exercise: multiplication as repeated sum

Similar to the example above in which we compute `x^n` as `x * x * x…` done `n` times (with `n` being an integer >= 0), write a program that computes `x * n` as `x + x + x…` done `n` times.

Echo

Although loops that perform an operation `n` times are quite common, that's not the general case. In the following example we have a program that keeps writing back the number integers that the user writes unless it's 0.

Should the 0 itself be written back to the user? If yes then we would write the program like this:

```
#include <iostream>



int main(){

    int n = 1;



    while(n != 0){ // != means different or not equal

        std::cin >> n;

        std::cout << n << std::endl;

    }

}
```

Note that we must initialise the value of variable `n` to something other than `0` because uninitialised variables get random values and it may well be `0`, which would mean that we never enter the loop!

If we don't want the `0` to be printed then we have to structure the while loop as follows:

```
#include <iostream>



int main(){

    int n;



    std::cin >> n;



    while(n != 0){

        std::cout << n << std::endl;

        std::cin >> n;

    }

}
```

In this case we read the value of the variable before the condition of the loop is tested, therefore we do not need to initialise it.

Note that we have a read instruction outside the loop and another one inside the loop after the printing. This allows us to always test the condition between the reading and printing so that the `0` that stops the loop is not printed.

Exercise: echo within range

Now let's assume we want the repetition to take place only if the user inputs a number between -10 and 10, extremes included. You may think that we can write this condition in the loop as:

```
while(-11 < n < 11){ // wrong

    ...

}
```

But this would actually not be correct. The program would run but not in the correct way. This must be written using the boolean operators as follows:

```
while( (-11 < n) && (n < 11) ){

    ...

}
```

Complete and test this version of the program.

Exercise: echo with text

Write a variation of the program “Echo” that instead of working with integer numbers works with text. The program stops when the user enters “STOP” (the “STOP” shouldn't be printed).

Exercise: echo with several stops

Same as above but the program stops also if the user enters “Stop” or “stop” (or “STOP” as above). You will need to use boolean operators to implement this condition.

Exercises

The sum so far

Write a program that keeps reading integers from the users (stopping when the user enters 0) and after each number read prints what's the sum of the numbers at that point. For example:

```
enter a number:
```

```
3

the sum so far is:

3

enter a number:

5

the sum so far is:

8

enter a number:

-1

the sum so far is:

7

enter a number: 2

the sum so far is:

9

enter a number: 0
```

The average so far

Write a program that keeps reading integers from the users (stopping when the user enters 0) and after each number read prints what's the average of the numbers at that point. Note that the numbers in input are integers but the average will not always be an integer. For example:

```
enter a number:

3

the average so far is:

3
```

```
enter a number:

5

the average so far is:

4

enter a number:

-1

the average so far is:

2.3333

enter a number: 2

the average so far is:

2.25

enter a number: 0
```

Division as repeated subtraction

Consider the case in which we have integers `a` and `b` and we want to compute the integer division `a / b` which includes a quotient `q` and a remainder `r` such that `q * b + r = a`.

We know that in C++ we can already compute this by using `q = a / b` and `r = a % b`.

However for the sake of the exercise we want to try and do this manually as repeated subtraction. Consider the following example:

```
we want to perform the integer division of 10 by 3



10 is greater than 3 so we subtract 3 from 10 (first subtraction)

we get 7
```

```
7 is greater than 3 so we subtract 3 from 7 (second subtraction)

we get 4



4 is greater than 3 so we subtract 3 from 4 (third subtraction)

we get 1



1 is not greater than 3 so we terminate the loop



the quotient is the number of subtractions (3)

the remainder is the number we have left at the end (1)
```

Write a program that performs this task.

3.2 For Loops

How do "for" loops help?

Consider again the infinite loop example:

```
// WARNING: infinite loop!

#include <iostream>



int main(){

    int n, counter = 0;



    std::cin >> n;
```

```
    while(counter < n){
        std::cout << "hello" << std::endl;

        // if we remove or forget to include
        // the instruction updating counter
        // if the condition is true the first time
        // it will always stay true and the loop never ends

    }
}
```

This is a rather common occurrence and it can help to use a different kind of C++ loop, the “for loop”. (Note that in C++ the for loop is not like the for loop in some other languages.)

To understand for loops let's consider the three elements of the loop (initialisation, condition and increment/update) labeled in the (correct) version of the example below:

```
#include <iostream>


int main(){
    int n, counter = 0; // initialisation of counter

    std::cin >> n;

    while(counter < n){ // condition involving counter
```

```
        std::cout << "hello" << std::endl;

        counter = counter + 1; // increment/update of counter

    }

}
```

This can be written as a for loops as follows:

```
#include <iostream>



int main(){

    int n;



    std::cin >> n;



    for(int counter = 0; counter < n; counter = counter + 1){

        std::cout << "hello" << std::endl;

    }

}
```

Now initialisation, condition and increment/update are all together and all in one line, so that the structure is more compact and it's much harder to forget one of the components.

Having to do something `n` times is very common in programming. Most of the times this would be written like this:

```
for(int i = 0; i < n; i++){

    // include here what needs to be done n times
```

```
}
```

This involves the following details:

- We usually or conventionally name the variable used for counting `i` (the reason for this will be clearer later on).
- The increment of the variable is done using the shorthand `++`: `i++` just means `i = i + 1`.
- The variable we use for counting is initialised at 0 (not 1).
- The condition uses the strict less than (not less than or equal) n (`i < n`).

Exercise: print i

Consider the following program. Read the code and make a note somewhere of what you expect this program to print. Then test the program to check your expectation.

```
#include <iostream>



int main(){

    for(int i = 0; i < 7; i++){

        std::cout << i << std::endl;

    }

}
```

Exercise: x^n using a for loop

Write a program that computes `x^n` (whith n integer >= 0) as repeated multiplications by using a for loop.

Exercise: factorial

The factorial of `n` (`n!`) in general can be computed as `n * (n - 1) * (n - 2)… * 2 * 1`. For example: `5! = 5 * 4 * 3 * 2 * 1 = 120`, `2! = 2`, etc. Also `0!` is defined and `0! = 1`.

Write a program that reads a number from the user and prints its factorial.

When should we use a while loop and when a for loop?

Let's rewrite the Echo example using a for loop:

```
#include <iostream>



int main(){

    int n;



    std::cin >> n; // initialisation



    for( ; n != 0; ){ // condition

        std::cout << n << std::endl;

        std::cin >> n; // update

    }

}
```

As you see, it can be done, but since we end up writing the initialisation, condition and update on three different lines anyway, this is not a case in which we have any advantages from writing this as a for loop.

For loops are the straightforward choice for all the cases in which we need to do something n times, or when there is a similar counting pattern. In other cases a while loop may be preferable.

Devise exercises involving while loops and for loops

Now that you have practised using `while` loops and `for` loops, apply your creativity to devise some exercises that involve these concepts (even better if it's something that can have a practical purpose) and implement the solutions.

Aim in particular to devise at least one exercise for which a `while` loop would be indicated and one for which a `for` loop would be indicated.

You can also use concepts from previous sections.