C++ in 90 minutes

Antonio Mucherino

Laboratoire d'Informatique, École Polytechnique, Palaiseau
mucherino@lix.polytechnique.fr

Amphi Lagarrigue, École Polytechnique,

September 15th 2009, 8:30 - 10:00

What's C++

what we are going to learn today

C++:

- is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language;
- is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features;
- was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C programming language and originally named "C with Classes";
- was renamed to C++ in 1983;
- is one of the most popular languages ever created;
- is widely used in software industries.

Aim of this lecture

what you should know at the end of the lecture

The aim of this lecture is to discuss the main features of C++.

- C++ basis
- Control structures
- Functions
- Programming paradigms
- Classes
- Inheritance and polymorphims

This lecture includes more than 50 slides and it is going to last 90 minutes. So, we can devote less than 2 minutes per slide.

Note that general concepts will be learned,
that will be understood well only with practice.

Functions in C++

why are we starting with functions?

Whatever language you want to learn, they will always teach you, at first, how to write the program known as Hello World!

What do we need to know for writing this program?

- I/O system of C++
- how to write a function in C++

It is clear the I/O is needed, but why should we know about functions in C++?

Answer: Each program in C++ is a function, which is called `main` function:

```
main()
{
   // the program in C++ is written here
};
```

by the way ..... the symbol ***//*** indicates that what follows is a comment for the programmer, and it is ignored by the compiler.

I/O in C++

let's learn how to write now

How to write a string on the screen:

```
cout << "This is a message on the screen" << endl;
```

A few comments:

- the symbol **<<** indicates that *all the things* on the right must be *sent to the left*;
- `cout` refers to the console output;
- it is defined in the header file `iostream.h`, that must be therefore included in the programs that use the I/O system of C++;
- **endl** is the delimiter that indicates the end of the line.

Hello World!!

we are ready for our first program

This is our first program:

```
#include <iostream.h>

main()
{
    // printing the message
    cout << "Hello World!!" << endl;
};
```

If this program is compiled and executed, you will obtain the following output:

```
Hello World!!
```

Variables

where to store data

Variable: a symbol representing a quantity capable of assuming any of a set of values.

Data type: it defines the set of values that a variable can assume.

Standard data types in C++:

- integer: int
- real: float (single precision) and double (double precision)
- boolean: bool
- character: char

This part of code declares an integer variable called a and assigns to it the value 5:

```
int a;

a = 5;
```

A more interesting World

Hello World!! messages with dates

Let's write a program that says "Hello World!!" and then it also says today's date:

```
#include <iostream.h>
main()
{
    // declaring the variables
    int day,month,year;

    // assigning a value to the variables
    day = 15; month = 9; year = 2009;

    // printing the messages
    cout << "Hello World!!" << endl;
    cout << "Today's date: ";
    cout << day << "-" << month << "-" << year << endl;
};
```

The output of this program will always be:

```
Hello World!!
Today's date:  15-9-2009
```

Arrays

ordered lists of variables of the same type

If we have to store n variables of the same type, we can use n different variables, but it is usually preferable to consider only one **array of variables**.

In C++, we can declare an array as follows:

```
int a[10];
double v[3];
char ch[5];
```

and elements of an array can be assigned as follows:

```
a[3] = 1;
v[1] = 3.23;
ch[0] = 'x';
```

Note that the elements of an array are ordered from 0 to $n-1$, where n is the dimension of the array specified during the declaration.

Same program - different syntax

let's use an array


```
#include <iostream.h>
main()
{
    // declaring the variables
    int date[3];

    // assigning a value to the variables
    date[0] = 15; date[1] = 9; date[2] = 2009;

    // printing the messages
    cout << "Hello World!!" << endl;
    cout << "Today's date: ";
    cout << date[0] << "-" << date[1] << "-" <<
            date[2] << endl;
};
```

Note that:

- we could substitute an array with the three variables `day`, `month` and `year` because they are all of the same kind;
- the number of elements of the array is known a priori.

Static vs. Dynamic allocation of memory

do you know how long your array must be?

This is the declaration of an array where the memory is allocated statically:

```
int a[10];
```

because the dimension of the array is 10 and it will always be 10.

What if we don't know the dimension of the array when we declare it?

Solution: dynamic allocation

```
int *a; // the dimension is not specified
...
a = new int [n]; // memory for a is allocated here
...
delete [] a; // the memory is deallocated here
```

Note that:

- the variable n must be an integer that contains the desired dimension for a;
- n is defined during the execution of the program.

Pointers

I want to know where my data are

In C++,

```
int *a;
```

declares a *pointer* to integer variables.

Pointer: a variable that holds the address of another variable or the first address of an array of variables.

Once the memory has been allocated

```
a = new int [n];
```

we can refer to the elements of the array as follows

```
a[i] = 1;
```

where `i` must be an integer variable between `0` and `n-1`.

Note that:

- arithmetic operations can be performed on pointers (ex. `a+1` is another pointer);
- different pointers can refer to the same memory address.

Strings

let your program remember your sentences

In C++, a string can be declared as an array of characters:

```
char string[100];
```

Note that:

- memory for strings can be allocated statically or dynamically;
- in any case, `string` is a pointer to `char`;
- the name `string` refers to a string, whereas any `string[i]` refers to a character;

Example:

```
char string[5];
string[0] = 'C'; string[1] = 'i';
string[2] = 'a'; string[3] = 'o';
cout << string << endl;
```

produces as output:

```
Ciao
```

C++ provides addictional support for the management of strings
(*I can't tell you more, you would understand only after slide 30*).

Input arguments

how to pass information to our programs

Information to our programs can be passed through two variables, an integer and a pointer to strings:

```
main(int argc,char **argv)
{
   // the program in C++ is written here
};
```

These two variables contain particular values:

- `argc` is the number of arguments passed to our program;
- be aware that each program has at least one argument, which is the program name;
- `argv` points to an array of pointers `char*`, each of them pointing to a string containing an argument;
- the arguments are sorted as they are specified by the user.

Every day “Hello World!!”

another version of the program

Let’s put all what we just learned in the Hello World!! program.

We suppose that the current date is passed by the user through the input arguments:

```
#include <iostream.h>
main(int argc, char **argv)
{
    // printing the messages
    cout << "Hello World!!" << endl;
    cout << "Today’s date: ";
    cout << argv[1] << "-" << argv[2] << "-" <<
            argv[3] << endl;
};
```

If the user specifies as arguments I, dont and know, our output will be:

```
Hello World!!
Today’s date: I-dont-know
```

The product of numbers

a little more complex program

This program computes the product between two numbers:

```
#include <iostream.h>
#include <stdlib.h>

main(int argc, char **argv)
{
   int a,b,p;
   a = atoi(argv[1]);
   b = atoi(argv[2]);
   p = a*b;
   cout << "The product is " << p << endl;
};
```

Note that:

- `atoi` converts a string into an integer;
- the header file `stdlib.h` must be included for using it;
- even though we provide more than 2 numbers, the program always computes the product of the first 2 only.

for

how many times do you want to do it?

The **for loop** repeats a set of instructions a predetermined number of times.

Its general format is:

```
for (initialization; condition; change) instruction(s);
```

where:

- `initialization` defines the first value of the counter;
- `condition` indicates when the loop must stop;
- `change` indicates how to modify the counter at each iteration.

An example:

```
for (i = 10; i > 5; i--)
{
   cout << i << endl;
};
```

The product of numbers

let's include a **for** loop

This program computes the product among *n* numbers:

```
#include <iostream.h>
#include <stdlib.h>

main(int argc, char **argv)
{
   int i,p;

   p = 1;
   for (i = 1; i < argc; i++)
   {
      p = atoi(argv[i])*p;
   };
   cout << "The product is " << p << endl;
};
```

This program is an extention of the previous one, is this the best program for computing products of numbers?

if

when your program takes decisions

The **if** keyword is used to execute an instruction or a block of instructions only when a certain condition is satisfied.

Its general format is:

```
if (condition)
{
   instruction(s) A;
}
else
{
   instruction(s) B;
};
```

where:

- `condition` is a logical condition;
- A marks the instructions that are executed if `condition` is true;
- B marks the instructions that are executed if `condition` is false.

The product of numbers

let's use an **if**

This program computes the product among *n* numbers more efficiently:

```
#include <iostream.h>
#include <stdlib.h>

main(int argc, char **argv)
{
   int i,p;

   p = 1;
   for (i = 1; i < argc; i++)
   {
      if (p != 0)
      {
         p = atoi(argv[i])*p;
      };
   };
   cout << "The product is " << p << endl;
};
```

is there a way to avoid useless steps?

while

do it until I'll tell you

The **while** loop is used to execute an instruction or a block of instructions while a given condition is satisfied.

Its general format is:

```
while (condition)
{
   instruction(s);
};
```

where:

- `condition` is a logical condition;
- `instruction(s)` represents the instruction or the block of instructions that are executed while `condition` satisfied.

Note that there is also another kind of loop that is called repeat . . . until.

The product of numbers

let's use a **while** loop

This program computes the product among *n* numbers more efficiently:

```
#include <iostream.h>
#include <stdlib.h>

main(int argc, char **argv)
{
   int i,p;

   i = 1; p = 1;
   while (i < argc && p != 0)
   {
      p = atoi(argv[i])*p;
      i = i + 1;
   };
   cout << "The product is " << p << endl;
};
```

Ok, this is efficient enough . . . . .

Functions

let's discuss a little more on functions

We already know that all the instructions of a program must be included into a C++ function called `main`.

A more general example of function in C++ is:

```
int funct(int a,double *b,char *c);
```

Note that:

- the function has a returning value, whose data type is specified on the left of the function name;
- the list of input arguments of the function is after the function name, between parentheses.

Important:

- new copies of the variables are placed in memory when the function is called, so that variables modified *inside* the function are unchanged *outside*;
- there is actually a way for having a variable *modificable* inside a function, but we will not discuss about this.

The function **prod**

we split the program in two functions

This is a C++ function that computes the product among *n* numbers:

```
int prod(int n, int *a)
{
   int i,p;

   i = 0; p = 1;
   while (i < n && p != 0)
   {
      p = a[i]*p;
      i = i + 1;
   };

   return p;
};
```

In order to call `prod` when executing another function (like the function `main`), we need to use the following syntax:

```
p = prod(n,a);
```

where n is an integer and a is a pointer to integers.

The function **prod**

we split the program in two functions

This is the new main function:

```
#include <iostream.h>
#include <stdlib.h>

int prod(int n, int *a);

main(int argc, char **argv)
{
   int i,n,*a,p;

   n = argc - 1;
   a = new int [n];
   for (i = 0; i < n; i++) a[i] = atoi(argv[i+1]);
   p = prod(n,a);
   cout << "The product is " << p << endl;
};
```

Overload of functions

my functions have the same name

C++ allows to define more than one function with the same name.

For example, we may have two functions `prod`:

```
int prod(int a,int b);
int prod(int n,int *a);
```

where:

- the first one computes the product between a and b;
- the second one computes the product among the n elements of a.

The compiler can understand which function to call on the basis of the input arguments that are passed to the function.

This is a kind of polymorphism in C++.

Another version of our program

exploiting the overload


```
#include <iostream.h>
#include <stdlib.h>

int prod(int a,int b);
int prod(int n,int *a);

main(int argc, char **argv)
{
   int i,n,*a,p;

   n = argc - 1;
   a = new int [n];
   for (i = 0; i < n; i++) a[i] = atoi(argv[i+1]);
   if (n == 2)
   {
      p = prod(a[0],a[1]);
   }
   else
   {
      p = prod(n,a);
   }
   cout << "The product is " << p << endl;
};
```

Procedural programs

when the program is split in **functions**

Main characteristics:

- the program is divided in subprograms and subsubprograms, each of them represented by a single function;
- the data can be shared by all the functions;
- each subprogram is a *mathematical function*, which, in theory, provides the same output when the same input arguments are given;
- it is good when the considered problem has a well defined solution, just like the problem of evaluating mathematical functions;
- easier to projet, preferable for small, medium-small sized projects.

Object Oriented programs

multiple independent intelligent agents

Main characteristics:

- the program is diveded in objects;
- each object contains its data, and methods that may act on these data (*encapsulation*);
- objects are reusable self-containing programming modules;
- objects can assume particular *states*, described by their data, and methods (with the same list of arguments) can provide different answers when called;
- objects prevent accidents with data;
- objects allow a simpler management of large-scale projects.

Definition of Class

this is the most important thing you're learning today

What is a Class?

- Classes define the common features of a group of objects;
- they allow the definition of a new data type;
- they also allow to define methods that manipulate the new generated data type;
- classes are at the basis of the *Object-Oriented* programming.

Important to note: every time a new object is declared, it has a concrete existence in the memory of the computer.

Class

An example

This code defines a class called `list`:

```
class list
{
   int n;
   int *v;

   int sum();
   int prod();
   void remove(int x);
};
```

All the objects belonging to this class contain an integer `n` and an array of integers `v`, together with the three methods `sum`, `prod` and `remove`.

Note that: all the members of a class are private by default (only the other class members can access to it), the keyword `public` must precede all the members that needs to be public.

Class

An example

In this case all the methods are public:

```
class list
{
   int n;
   int *v;

   public:
   int sum();
   int prod();
   void remove(int x);
};
```

Data hiding helps the programmer to reduce memory errors, because only the methods that are allowed to use the data can access them.

Somebody says that the *real purpose* of Object-Oriented programming is data hiding . . . . . .

Class

how to refer to data and methods

Once a class has been defined, how do I declare an object of that class?

```
list l;
```

Once an object has been declared, how do I access its data?

```
l.n = 1;
l.v[0] = 0;
```

How can I say to the compiler that this is the code for the method `sum` belonging to the class `list`?

```
int list::sum(void)
{
   // the code for the method goes here
};
```

How do I call this method?

```
mysum = l.sum();
```

Constructors and Destructors

how to construct and how to destroy an object

In many cases, before using an object, the data it contains need to be initialized.

This class contains a constructor and a destructor:

```
class list
{
   int n;
   int *v;

   public:
   list(); // constructor
   ~list(); // destructor
   int sum();
   int prod();
   void remove(int x);
};
```

They are two methods of the class. They are called automatically when the object is created (constructor) or when the object is deleted (destructor).

Constructors and Destructors

how to construct and how to destroy an object

This is the code for a constructor:

```
list::list()
{
   ...
};
```

and this is the code for a destructor:

```
list::~list()
{
   ...
};
```

In our example, memory needs to be assigned to the array of integers v.

How can we say to the constructor how much memory we need?

Parameterized constructors

construct it as I tell you

```
class list
{
   int n;
   int *v;

   public:
   list(int dim); // constructor
   ~list(); // destructor
   int sum();
   int prod();
   void remove(int x);
};
```

This can be the code for the parameterized constructor:

```
list::list(int dim)
{
   n = dim;
   v = new int [dim];
   for (int i = 0; i < dim; i++) v[i] = 0;
};
```

Again, the purpose of this example is to clarify the presented concepts, there are actually other solutions in C++ for the management of lists.

Parameterized constructors

construct it as I tell you

```
class list
{
   int n;
   int *v;

   public:
   list(int dim); // constructor
   ~list(); // destructor
   int sum();
   int prod();
   void remove(int x);
};
```

This can be the code for the parameterized constructor:

```
list::list(int dim)
{
   n = dim;
   v = new int [dim];
   for (int i = 0; i < dim; i++) v[i] = 0;
};
```

Again, the purpose of this example is to clarify the presented concepts, there are actually other solutions in C++ for the management of lists.

A theoretical class

let's define a class that we'll consider in the following examples

In the following, we will consider the class `instrument`:

```
class instrument
{
   int type;
   char name[100];
   double cost;

   public:
   instrument();
   ~instrument();
   int method1(double x);
};
```

It represents the class of musical instruments, where:

- the data are represented through `type`, `name` and `cost`;
- `instrument()` and `~instrument()` are the constructor and the destructor;
- `method1` is a method of the class.

Arrays and Pointers of Objects

we can treat the objects as the predefined data types

Arrays of objects and pointers to objects can be used, as in the following example:

```
main (int argc,char **argv)
{
   instrument band[10];
   instrument *m;

   ...

   cout << "Type of first instrument in band = ";
   cout << band[0].type << endl;

   ...

   m = new band [100];

   ...

   delete [] m;
};
```

Friend functions

classes' friends

Functions that are not member for a class can access the private data of the class if they are declared as **friend** of the class.

```
class instrument
{
   int type;
   char name[100];
   double cost;

   public:
   instrument();
   ~instrument();
   int method1(double x);

   friend int func(instrument a,instrument b);
};
```

In the function `func`, the members of the class can be accessed through the syntax `a.type`, `a.name`, etc.

Inheritance

this is very important in object-oriented programming

Inheritance allows to create classes which are derived from other classes, so that they automatically include some of its *parent's members*, plus its own.

Example: this class is derived from `instrument`:

```
class guitar :  instrument
{
   char strings[6];

   public:
   void set_strings(strings);
   ...
};
```

The data and the methods in `instrument` are *inherited* by `guitar`.

public, private and protected

three different behaviors for the class members

- **Public**
  - public members are accessible by their own class and by any other class;
- **Private**
  - private members are accessible by their own class only;
  - any other class, even derivate classes, cannot access them;
- **Protected**
  - protected members are accessible by their own class;
  - derivate classes can also access them;
  - other classes cannot access them.

Polymorphism in execution

the second type of polymorphism in C++

The overload of functions allows the polymorphism in compilation in C++.

What about the polymorphism in execution?

- a method in a class can be defined as **virtual**;
- this means that all the derived classes can have their local implementation of the method;
- during the execution, the implementation of the method is chosen depending on the pointer type which is used to invoke the method;
- note that pointers to base classes can also point to derived classes.

Example of polymorphism in execution

definition of the classes


```
class instrument
{
   int type; char name[100]; double cost;

   public:
   instrument(); ~instrument();
   ...
   virtual double get_cost(void);
};

class guitar :  instrument
{
   char strings[6];
   ...
   virtual double get_cost(void);
};

class piano :  instrument
{
   char keys[88];
   ...
   virtual double get_cost(void);
};
```

Example of polymorphism in execution

an example of main function


```
main ()
{
   instrument *pointer;
   guitar g;
   piano p;

   pointer = &guitar;
   guitar_cost = (*pointer).get_cost();
   ...

   pointer = &piano;
   piano_cost = (*pointer).get_cost();
   ...
};
```

Note that

```
(*pointer).get_cost()
```

is equivalent to

```
pointer->get_cost().
```

Handling exceptions

I didn't want this to happen!

There might be situations in which a program is not able to proceed its execution.

For example:

- an input argument that was expected to be positive is instead negative;
- there is no memory enough on the computer for all the arrays needed to the program;
- ...

What can we do when we have these exceptions?

The management of exceptions in C++ is possible by using the three keywords

- try
- throw
- catch

Handling exceptions

An example

In order to catch exceptions, we need to place a portion of code under exception inspection:

```
try
{
   if (n < 0) throw 0;
};
```

If we discover an exception, we have to use the keyword `throw`, that accepts an argument.

The following portion of code must be located just after `try` and it describes how to handle the exceptions:

```
catch (int i)
{
   if (i == 0)
   {
      cout << "Wrong input parameter:  n" << endl;
      return 1;
   }
   ...
};
```

Bugs

are there bugs in your program?

A **software bug** is the common term used to describe an error, flaw, mistake, failure, or fault in a computer program that produces an incorrect or unexpected result, or causes it to behave in unintended ways.

Why are there bugs around?

It's our fault!

Bugs are often caused by:

- bad software designs (incorrect class definitions, ...);
- mistakes in the code (a **+** operator used instead of a – operator, ...).

In order to reduce the number of bugs in our programs, we can follow some important rules when programming.

Tips for the programmer

some good rule

A way to *reduce the number of bugs* in our programs is to follow these three simple rules:

- code clear to read
  - it is not only important that our compiler understands our codes, it is also important that the programmer, and everybody else who knows the syntax, can read it;
- indentation
  - it's an efficient way to emphasize where blocks of code start and end;
  - it helps in producing a clearer code, but it's not the only thing to do for having a clear code;
- comments
  - they allow the programmer to take notes;
  - they are important for who writes the program, and they are very important for who reads the code of the program.

The International Obfuscated C Code Contest

obfuscate = to darken

Goal of the Contest:

- To write the most Obscure/Obfuscated C program under the rules below.
- To show the importance of programming style, in an ironic way.
- To stress C compilers with unusual code.
- To illustrate some of the subtleties of the C language.
- To provide a safe forum for poor C code. :-)

`http://www.ioccc.org/`

The International Obfuscated C Code Contest

An example: details can be found on the official web site

This code:

```
                                #include\
                                <stdio.h>
                    #include                <stdlib.h>
                    #include                <string.h>

                     #define w "Hk~HdA=Jk|Jk~LSyL[{M[wMcxNksNss:"
                    #define r"Ht@H|@=HdJHtJHdYHtY:HtFHtF=JDBIl"\
                   "DJTEJDFllMIlM:HdMHdM=I|KIlMJTOJDOIlWITY:8Y"
                  #define S"IT@I\\@=HdHHtGH|KILJJDIJDH:H|KID"\
                 "K=HdQHtPH|TIDRJDRJDQ:JC?JK?=JDRJLRI|UItU:8T"
                #define _(i,j)L[i=2*T[j,O[i=O[j-R[j,T[i=2*\
               R[j-5*T[j+4*O[j-L[j,R[i=3*T[j-R[j-3*O[j+L[j,
              #define t"IS?I\\@=HdGHtGIDJILIJDIItHJTFJDF:8J"


    #define y                   yy(4),yy(5),                    yy(6),yy(7)
  #define yy(               i)R[i]=T[i],T[i ]               =O[i],O[i]=L [i]
#define Y _(0           ], 4] )_ (i ], 5] )_ (2       ], 6] )_ (3 ], 7] )_=i
#define v(i)(       (( R[ i ] * _ + T [ i ]) * _ + O [ i ]) * _ + L [ i ]) *2
double b = 32  ,l ,k ,o ,B ,_ ; int Q , s , V , R [8 ], T[ 8] ,O [8 ], L[ 8] ;
#define q( Q,R ) R= *X ++ % 64 *8 ,R |= *X /8 &7 ,Q=*X++%8,Q=Q*64+*X++%64-256,
# define  p       "G\\QG\\P=GLPGTPGdMGdNGtOGlOG"    "dSGdRGDPGLPG\\LG\\LHtGHtH:"
#  define W          "Hs?H{?=HdGH|FI\\II\\GJlHJ"     "lFL\\DLTCMlAM\\@Ns}Nk|:8G"
# define   U            "EDGEDH=EtCElDH{~H|AJk}"        "Jk?LSzL[|M[wMcxNksNst:"
#  define u                    "Hs?H|@=HdFHtEI"               "\\HI\\FJLHJTD:8H"
char  *   x                    ,*X , ( * i )[                  640],z[3]="4_",
*Z = "4,8O4.8O4G" r U "4M"u S"4R"u t"4S8CHdDH|E=HtAIDAIt@IlAJTCJDCIlKI\\K:8K"U
 "4TDdWDdW=D\\UD\\WF\\FFdHGtCGtEIDBIDDIlBIdDJT@JLC:8D"t"4UGDNG\\L=GDJGLKHL\
FHLGHtEHtE:"p"4ZFDTFLT=G|EGlHITBH|DIlDIdE:HtMH|M=JDBJLDKLAKDALDFKtFKdMK\
\\LJTOJ\\NJTMJTM:8M4aGtFGlG=G|HG|H:G\\IG\\J=G|IG|I:GdKGlL=G|JG|J:4b"W
S"4d"W t t"4g"r w"4iGlIGlK=G|JG|J:4kHl@Ht@=HdDHtCHdPH|P:HdDHdD=It\
BIlDJTEJDFIdNI\\N:8N"w"4lID@IL@=HlIH|FHlPH|NHt^H|^:H|MH|N=J\\D\
J\\GK\\OKTOKDXJtXItZI|YIlWI|V:8^4mHLGH\\G=HLVH\\V:4n" u t t
"4p"W"IT@I\\@=HdHHtGIDKILIJLGJLG:JK?JK?=JDGJLGI|MJDL:8M4\
rHt@H|@=HtDH|BJdLJTH:ITEI\\E=ILPILNNtCNlB:8N4t"W t"4u"
p"4zI[?Il@=HlHH|HIDLILIJDII|HKDAJ|A:JtCJtC=JdLJtJL\
THLdFNk|Nc|\
:8K"; main (
int C,char**        A) {for(x=A[1],i=calloc(strlen(x)+2,163840);
C-1;C<3?Q=_=       0,(z[1]=*x++)?((*x++==104?z[i]^=32:--x), X =
strstr(Z,z))      &&(X+=C++):(printf("P2 %d 320 4 ",V=b/2+32),
V*=2,s=Q=0,C     =4):C<4?Q-->0?i[(int)((l+=o)+b)][(int)(k+=B)
]=1:_?_-=.5/    256,o=(v(2)-(l=v(0)))/(Q=16),B=(v(3)-(k=v(1)
))/Q:*X>60?y   ,q(L[4],L[5])q(L[6],L[7])*X-61||(++X,y,y,y),
Y:*X>57?++X,  y,Y:*X >54?++X,b+=*X++%64*4:--C:printf("%d "
,i[Q][s]+i[Q ][s+1]+i[Q+1][s]+i[Q+1][s+1])&&(Q+=2)<V||(Q=
0,s+=2)<640
||(C=1));}
```

can produce as result:

Compiling

generating the machine code

For Linux/Unix users:

```
g++ -o myprog myprog_p1.cpp myprog_p2.cpp
```

Note that:
g++ is the C++ GNU Compiler, it is free and it is usually available on all Linux/Unix machines.

For Windows users:
you can download and install on your pc

- the version for Windows of the GNU compiler:
  `http://gcc.gnu.org/install/specific.html#windows`
- Microsoft Visual C++ (some versions are free):
  `http://msdn.microsoft.com/en-us/visualc/`

Note that . . .

. . . the codes showed in the lecture are for teaching purposes only

When you will try to compile the program:

```
#include <iostream.h>

main()
{
    // printing the message
    cout << "Hello World!!" << endl;
};
```

you will probably have the following *warning message*:

#warning This file includes at least one deprecated or antiquated header. Please consider using one of the 32 headers found in section 17.4.1.2 of the C++ standard. Examples include substituting the <X> header for the <X.h> header for C++ includes, or <iostream> instead of the deprecated header <iostream.h>. To disable this warning use -Wno-deprecated.

How to avoid that?

Note that . . .

. . . the codes showed in the lecture are for teaching purposes only

The warning is not given when trying to compile this version of the program:

```
#include <iostream>

main()
{
    // printing the message
    std::cout << "Hello World!!" << std::endl;
};
```

or this version of the program:

```
#include <iostream>

using namespace std;

main()
{
    // printing the message
    cout << "Hello World!!" << endl;
};
```

Some References

where to continue your studies

Books

- Herbert Schildt, *C++: A Beginner's Guide*, 2^{nd} edition, McGraw-Hill.
- Claude Delannoy, *Apprendre le C++*, Eyrolles.

Slides and notes on the Internet

- `cplusplus.com` - The C++ Resources Network.
- Leo Liberti, *C++ Notes*, available on the author's web site.

YouTube

- Brian Harvey, *Object Oriented Programming*, Berkeley University channel on YouTube.
- Jerry Cain, *Programming Paradigms*, Stanford University channel on YouTube.

You can easily find other material by a Google search.

```
cout << "Good luck with C++!!" << endl;
```

Antonio Mucherino