Introduction to programming 2014 - 2015

Corina Forăscu
corinfor@info.uaic.ro

http://profs.info.uaic.ro/~introp/

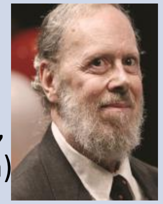
Course 1: agenda

- C++ - an overview
- Fundamental data types
- Strings, reserved words
- Variables, expressions, assignments
- Operators
- Boolean expressions, precedence
- Constants

C++: history

- 1979: Bjarne Stroustrup – Simula, C with classes … ->
- 1983: C++ (compared to C) classes
  - basic inheritance
  - inlining (inline keywords and Class Member Functions)
  - default function arguments
  - strong type checking
  - virtual functions
  - function overloading
  - references with the `&` symbol
  - `const` keyword
- 1998: STL
- …

Computer languages (1)

1957-1959	FORTRAN (FORmulaTRANslation LISP (List Processor) COBOL (Common Business-oriented language) Oldest languages still in use High-level	Terminator's vision has samples of COBOL source code	Supercomputing appl., AI devel., business software	NASA ATMs, Credit cards
1970	PASCAL (<- Blaise Pascal) High-level, for teaching data structuring	Niklaus Wirth (Turing '84)	Teaching programming	skype
1972	C Low-level, general-purpose, with many derivatives (C#, Java, Perl, PHP, Python)	Dennis Ritchie (Turing '83, K.Thomson) 	Cross-platform programming, System prog., Unix prog, computer game devel.	UNIX® early www servers & clients

Computer languages (2)

1979-1983-…	**C++** High-level, OO, expands C	Bjarne Stroustrup 	Commercial appl. devel., embedded software, server/client applic., video games	
1983	**Objective C** (OO extension of C) High-level, general purpose, OO, expands C with messages based on Smalltalk	Brad Cox & Tom Love 	Apple programming (OS X and iOS) WWDC '**14**: Swift = Objective C without C	
1987	**PERL** (Practical Extraction and Reporting Language) High-level, general-purpose, interpreted, multi-paradigm language	Larry Wall 	Text processing, graphics programming, system administration, network programming, finance, bioinformatics	Priceline, Ticketmaster

Computer languages (3)

1989-1991	PYTHON (for *Monty Python's Flying Circus*) High-level, general purpose, multi-paradigm language	Guido van Rossum 	WAD, software devel., information security, biologists, bioinformatics	
1993	RUBY (a collab's birthstone) High-level, general purpose, OO. Designed (Ada, C++, Perl, Lisp Python) for productive & enjoyable programming	Yukihiro **Matsumoto** 	Web appl. devel.,	
1995	JAVA (for the coffee consumed) High-level, general-purpose, cross-platform, multi-paradigm language	James Gosling (Sun Microsystems) 	network programming, WAD, GUI devel., software devel.	

Computer languages (4)

1995	PHP ("Personal Home Page" -> Hypertext Pre-processor) General purpose, open source for building dynamic web pages; influenced by C/++, Perl, Java	Rasmus Lerdorf	Building / maintaining dynamic web pages, server side devel.	
1995	Java Script (after "Mocha") High-level, scripting, OO, imperative, functional. Designed (C, Java, Python, Scheme) for web programming (esp. client side	Brendan Eich	Dynamic web development, PDF docs, web browsers, widgets,	

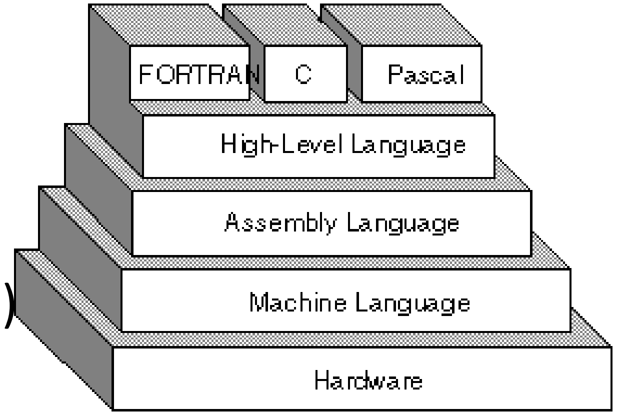
C++: essentials (1)

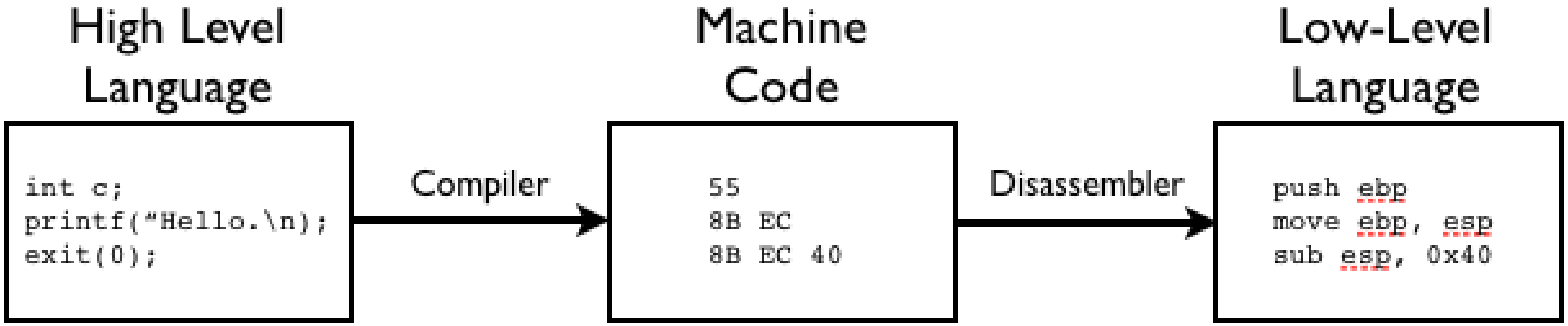
- One among the other approximately 2000
- C++ Origins

– Low-level languages (Machine, assembly)

– High-level languages (C, ADA, COBOL, FORTRAN)

– OOP





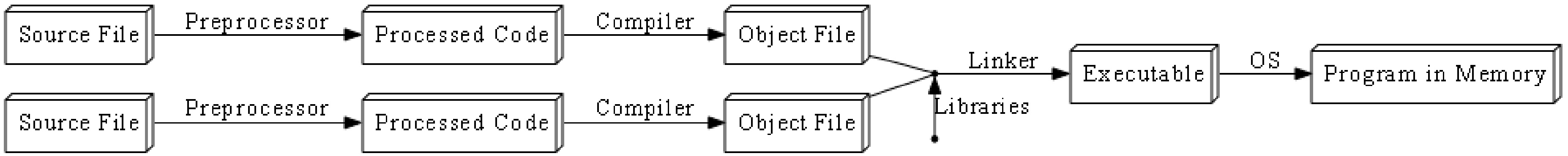
```
int c;
printf("Hello.\n);
exit(0);
```

```
55
8B EC
8B EC 40
```

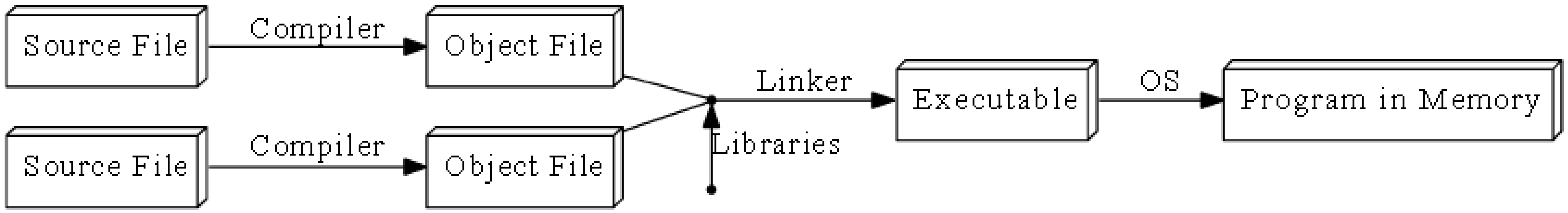
```
push ebp
move ebp, esp
sub esp, 0x40
```

Compilation

- in C++:



- generic



Debugging

- Bug – programming error
- <u>Compilation errors</u>: problems catched and raised by the compiler, generally resulting from violations of the **syntax** rules or **misuse** of types; usually caused by typos and the like.
- <u>Runtime </u>errors: problems that can only be spotted when the program is run: the program doesn’t do what it was expected to; these are usually more tricky to catch, since the compiler won’t tell you about them.

C++: essentials (2)

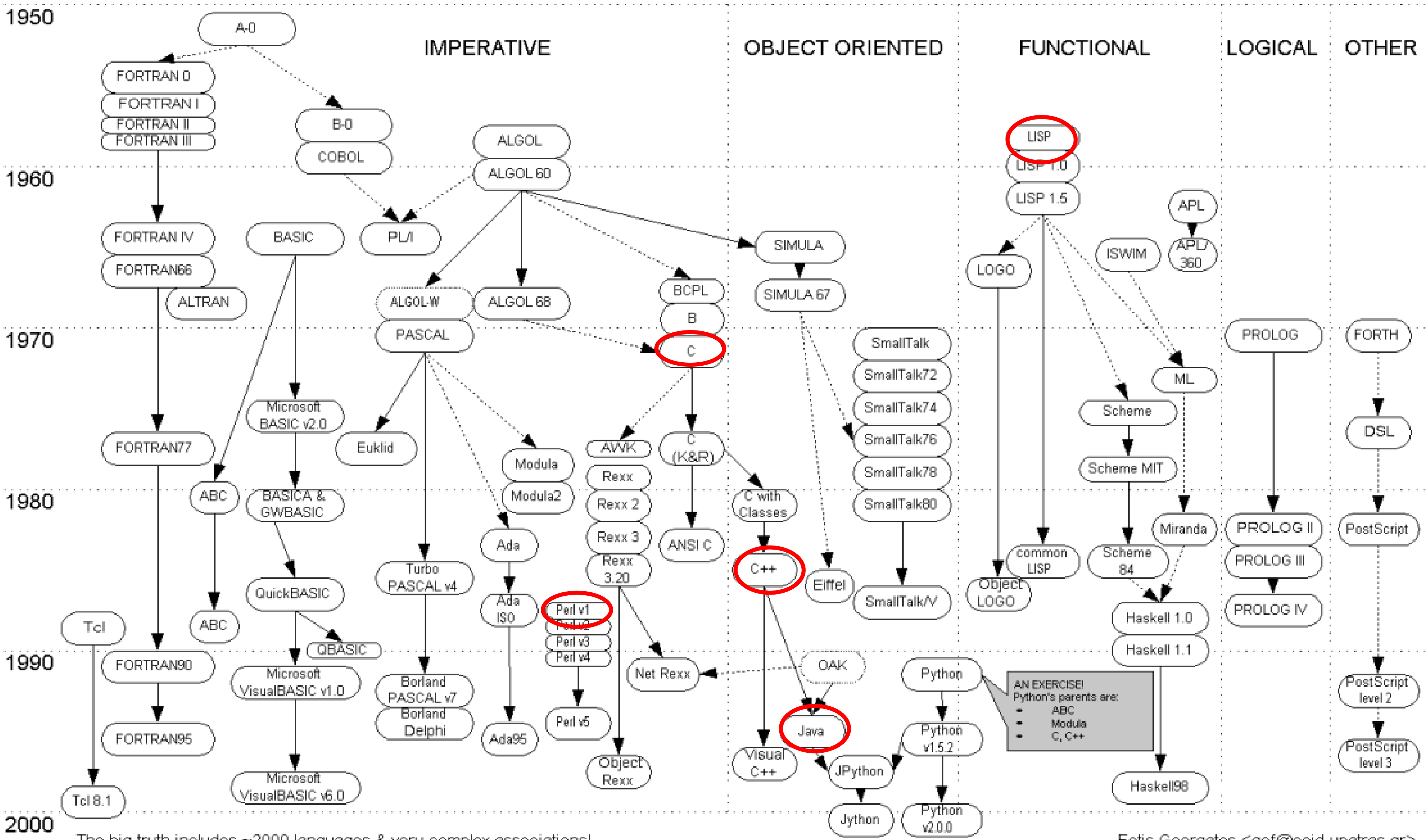
- (1998) open ISO-standardized language.
- compiled language (compiles directly to a machine's native code)
- strongly-typed unsafe language
- supports both manifest and inferred typing (explicitly defined or inferred variables’ types)
- supports both static and dynamic type checking (at compile-/run-time)
- offers many paradigm choices
- portable
- C-compatible
- huge library support

C++: essentials (3)

- Multi-paradigm:
  - generic -> algorithms are written in terms of types *to-be-specified-later* that are then *instantiated* when needed for specific types provided as parameters
  - imperative -> *how* (sequences of commands for the computer to perform)
  - object-oriented -> *objects* + attributes + procedures (methods)
  - functional -> computation as the evaluation of mathematical functions

COMPUTER LANGUAGES EVOLUTION

GENEALOGY TREE BETWEEN 1950-2000



The big truth includes ~2000 languages & very complex associations!
Read for more: *http://perso.wanadoo.fr/levenez/lang/history.html*

http://amstel.science.uva.nl/~fotisg/python

Fotis Georgatos <gef@ceid.upatras.gr>
Amstel Institute, Amsterdam, June 2002

(why) C++ plus+plus (1)

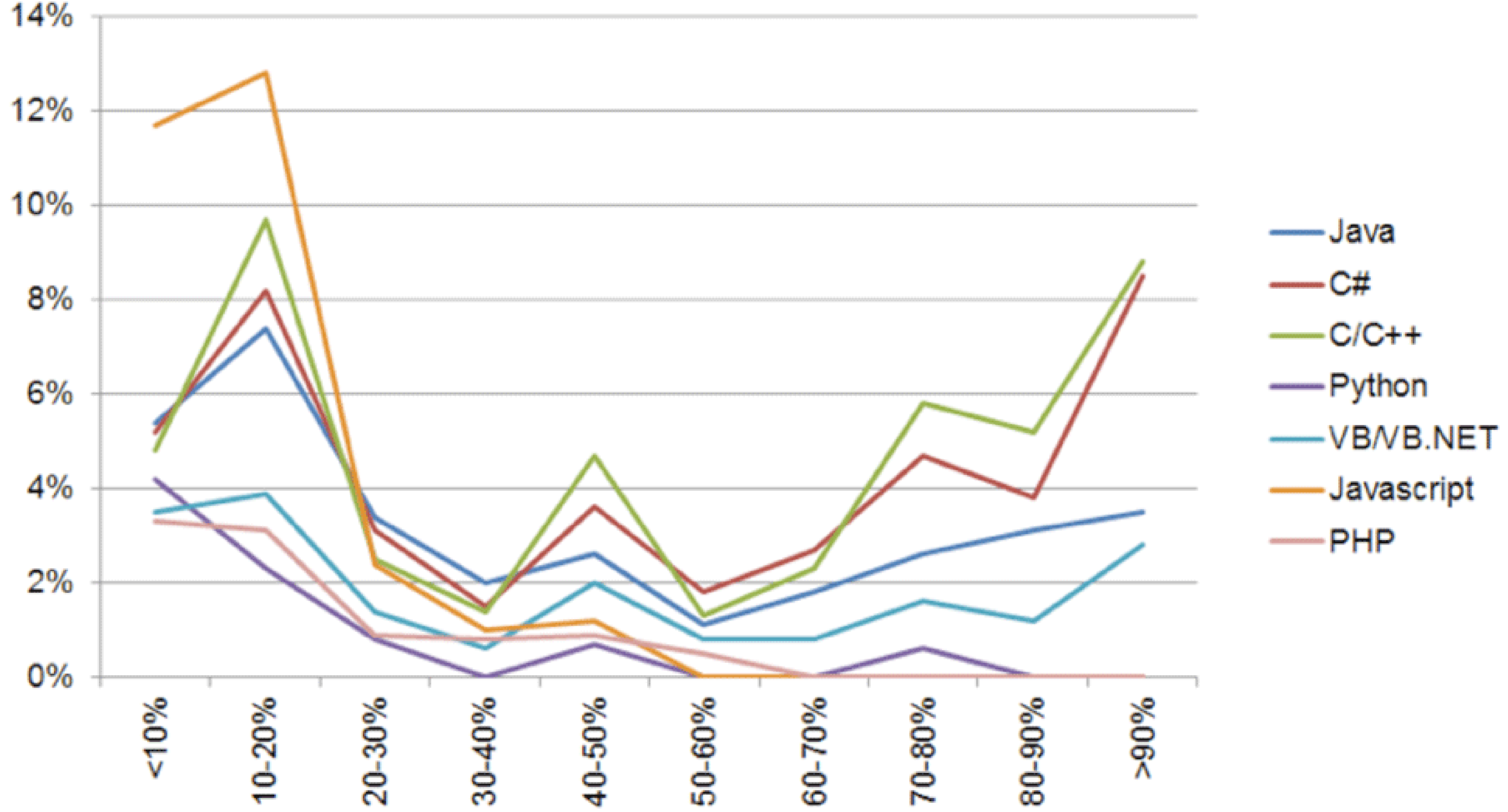
- ✓ portability (Windows, Apple, Linux, UNIX)
- ✓ structured
- ✓ maintainability
- ✓ conciseness
- ✓ less code, no run-time overhead, more safety
- ✓ faster (overall speed >= than that of other languages)
- ✓ templates and generic programming
- ✓ plethora of good libraries
- ✓ the preferred choice in programming for creating games
- ✓ starting point for other programming languages

(why) C++ plus+plus (2)

Oct 2014	Oct 2013	Change	Programming Language
1	1		C
2	2		Java
3	3		Objective-C
4	4		C++
5	6	↑	C#
6	7	↑	Basic
7	5	↓	PHP
8	8		Python
9	12	↑	Perl
10	9	↓	Transact-SQL
11	17	⇈	Delphi/Object Pascal
12	10	↓	JavaScript
13	11	↓	Visual Basic .NET
14	-	⇈	Visual Basic
15	21	⇈	R
16	13	↓	Ruby
17	81	⇈	Dart
18	24	⇈	F#
19	-	⇈	Swift
20	14	⇊	Pascal

Programming Language	2014	2009	2004	1999	1994	1989
C	1	2	2	1	1	1
Java	2	1	1	3	-	-
Objective-C	3	27	36	-	-	-
C++	4	3	3	2	2	2
C#	5	5	7	18	-	-
PHP	6	4	5	-	-	-
Python	7	6	6	23	21	-
JavaScript	8	8	9	15	-	-
Visual Basic .NET	9	-	-	-	-	-
Transact-SQL	10	28	-	-	-	-
Pascal	15	14	84	7	3	21
Lisp	17	17	13	14	5	3

(why) C++ plus+plus (3)



C++ IDE (Integrated Development Environment)

- VS - 1 (versus GCC - 2)

1. IDE: Microsoft Visual Studio:
   1. GUI based: more attractive & suggestive editor
   2. syntax checking
   3. debugger
2. IDE: GCC (the GNU Compiler Collection): command line compiler: Linux environment with g++ compiler
   1. faster, especially for short programs
   2. It compiles faster

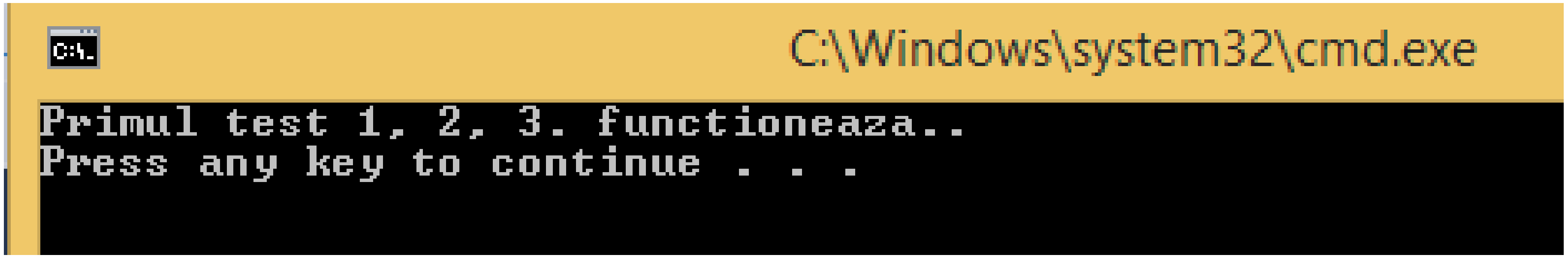
First program (1)

```
/*
 * first program in C++
 */

#include <iostream>  // #include<stdio.h>

int main ()
{
std::cout << "Primul test 1, 2, 3. "; //printf("");
std::cout << "functioneaza.. \n";
return 0;
}
```

First program (output)



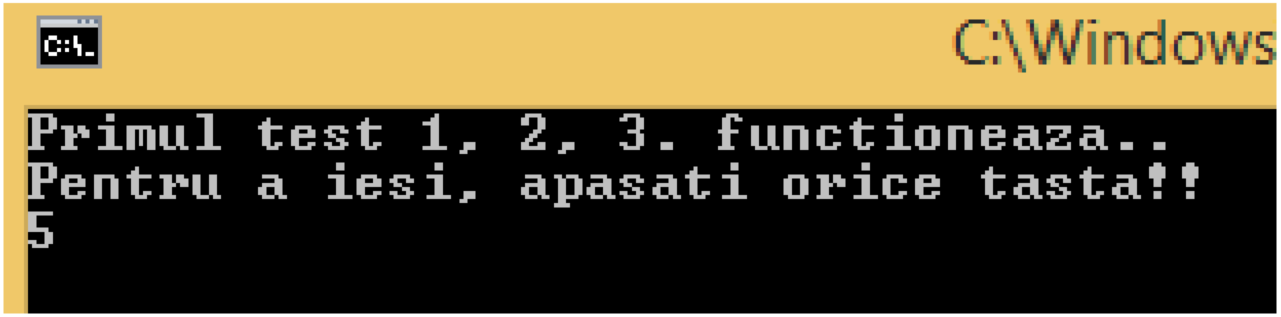
```
Primul test 1, 2, 3. functioneaza..
Press any key to continue . . .
```

First_program++

```cpp
/*
 * first program in C++
 */

#include <iostream>
using namespace std;

int main ()
{
cout << "Primul test 1, 2, 3. ";
cout << "functioneaza.. " << endl;
char c;
cout << "Pentru a iesi, apasati orice tasta!!\n";
cin >> c;
return 0;
}
```



```
Primul test 1, 2, 3. functioneaza..
Pentru a iesi, apasati orice tasta!!
5
```

General form of a C++ program

/*comments, ignored by the compiler */
// preprocessor directives
`#include <`*`header_file`*`>` (Input/output, math, strings, …)
// definition of CONSTANTS **`#define`**
//declaration of global variables (user-defined), functions **`type name`**`;`

```
returnType main(arguments from command line)
 {
 declaration of local variables used by main function;
 body of the main function;
 }
```

// user-defined functions

```
functionReturnType user_function(argument list)
 {
 declaration of local variables for user_function;
 body of the user_function;
 }
```

C++ header files

`<iostream>` (`<cstdio>`)	several standard stream objects
`<iomanip>`	Helper functions to control the format or input and output
`<cctype>`	functions to classify and transform individual characters
`<climits>`	limits of integral types
`<cmath>`	Common mathematical functions
`<complex>`	Complex number type
`<string>`	various narrow character string handling functions
`<exception>`	Exception handling utilities

C/++ fundamental elements

- **Expressions** – consisting of
  - **Data** represented by
    - variables
    - constants

    characterised by
    - type
    - name
    - value
    - memory
  - **Operators**
- **Expressions** – a sequence of *operators* and their *operands*, that specifies a computation
- expression evaluation may produce a result (calculation) and may generate side-effects (a function call)

Data types

- A **type** defines a set of values (**domain** for that type) and a set of operations that can be applied on those values, with a specific amount of memory for its storage.
- Categories of data types:
  - <u>Standard</u> types (Void, Boolean, Character, Integer, Floating-point)
  - <u>Complex</u> data types (String, Array, Pointer)
  - <u>High-level</u> data types (data structures)
    - The operations are implemented through user-defined algorithms

Fundamental data types

- `void`
- `null_pointer`
- Arithmetic types
  - Floating-point types (`float`, `double`, `long double`)
  - Integral types
    - The type `bool`
    - Character types (`char`, `un/signed char`)
  - Signed integer types (`short, int, long, long long`)
  - Unsigned integer types (`unsigned short, unsigned int, unsigned long, unsigned long long`)

Data types

Name	size	domain
unsigned char	8	0..255
char	8	-128..127
signed char	8	-128..127
unsigned int	16	0..65535
short int, signed int	16	-32768..32767
int	16	-32768..32767
unsigned long	32	0..4.294.967.295
long, (signed) long int	32	-2.147.483.648..2.147.483.647
float	32	accurate to 6 decimals
double	64	accurate to 10 decimals
long double	80	accurate to 15 decimals

Equivalences between data types

`signed short int`	≡	`short`
`unsigned short int`	≡	`unsigned short`
`signed int`	≡	`int`
`unsigned int`	≡	`unsigned`
`signed long int`	≡	`long`
`unsigned long int`	≡	`unsigned long`

Compound data types

- Reference
- Pointer
- Array
- Function
- Enumeration
- Class types

Characters

- **Letters:**

  `A B C D … X Y Z`

  `a b c d … x y z`

- **Digits:**

  `0 1 2 3 4 5 6 7 8 9`

- **Other characters:**

  `+ -* / ^ \ () [] {} = != <>`

  `‘ “ $ , ; : % ! & ? _ # <= >= @`

- **Space characters:** backspace, horizontal tab, vertical tab, form feed, carriage return

C++ tokens

Token type	Description/Purpose	Examples
Keywords	Words with special meaning to the compiler	`int, char, while, auto`
Identifiers	Names of things that are not built into the language `bun, _bun, bun1` VS `.rau, 1rau, rau!`	`cin, std, ics, aFunction`
Literals	Basic constant values with values specified directly in the source code	`"functie", 3.14, 0, ’a’`
Operators	assignment, mathematical, logical operations	`+, -, &&, %, <<`
Punctuation / Separators	Punctuation defining the structure of a program	`{ } ( ) , ;`
Whitespace	Spaces of various sorts; ignored by the compiler	Spaces, tabs, newlines, comments

C++ keywords

auto	const	double	float	int	short	struct	unsigned	
break	continue		else	for	long	signed	switch	void
case	default	enum	goto	register	sizeof	typedef	volatile	C
char	do	extern	if	return	static	union	while	

asm	dynamic_cast		namespace		reinterpret_cast			try
bool	explicit	new	static_cast		typeid			
catch	false		operator		template		typename	
class	friend		private		this		using	C++
const_cast			inline		public		throw	virtual
delete	mutable		protected		true		wchar_t	

and		bitand	compl	not_eq		or_eq	xor_eq	
and_eq		bitor		not		or	xor	

http://en.cppreference.com/w/cpp/keyword

Special characters

Escape sequence	Character
\b	Backspace
\t	horizontal tab
\v	vertical tab
\n	line feed - new line
\f	formfeed – new page
\r	carriage return
\"	double quote
\'	single quote
\\	backslash
\?	question mark
\a	audible bell

Variables

- A named location in the memory, used to store data; must be declared before use
- Declaration:

type variabila;

type var1, var2, …, varn;

type variabila = expresie_constanta;

```
char c;
signed char sc;
```

```
int a,b;
a = b = 5;
```

```
int i;
int suma = 0;
long j;
```

```
float x1, x2, x3;
float pi = 3.14;
double y;
```

Variable ’ s scope

- *Scope:* who can see it and how long it lives for
- *Global variable*: to be declared at the beginning of the program, <u>program-scope</u>
- *Local variable*: to be declared inside a block (function) where it will be used – <u>block-scope</u>
- Avoid global variables:
  - they increase the program’s complexity (search)
  - their values can be changed by any function that is called

Assignment

- in a declaration statement
- At execution time:
  - Lvalues (left-side) & Rvalues (right-side)
    - Lvalues -> an object that occupies some identifiable location in memory (i.e. has an address) ->variables
      - can appear on the left side of an assignment statement
      - constants are **not** lvalues
      - the result of an arithmetic expression is **not** an lvalue
    - Rvalues -> expressions
  - Compatibility!
  - Conversions!

```
5 = var; // ERROR!
(var + 1) = 5; // ERROR!
```

Integer constants

- Octal: prefix 0 (zero)

  032 = 26 077 = 63
- Hexadecimal: prefix 0x OR 0X

  0x32 = 50 0x3F = 63
- “long” integers: sufix l OR L

  2147483647L 0xaf9Fl = 44959
- “unsigned” integers: sufix u OR U

  345u 0xffffu = 65535
- Characters between single quotes: ‘A’, ‘+’, ‘n’
- Characters in decimal: 65, 42
- Characters in octal: ’\101’, ‘\52’
- Characters in hexadecimal: ‘\x41’, ‘\x2A’
- Special characters – escape sequences

Functions and operators for `int` types

- Operators :

  `+    -    *    /    %    ==    !=    <    <=    >`
  `>=    ++    --`

- Functions:
  - Those from `floating` type
  - Those included in **<cctype> (ctype.h)** library: **tolower, toupper, isalpha, isalnum, iscntrl, isdigit, isxdigit, islower, isupper, isgraph, isprint, ispunct, isspace**

Operators for `int` types

Type of operator	Operator	Associativitaty
Unary (postfix)	- (unary) ++ --	left to right
Unary (PREfix)	+ - (unary) ++ --	right to left
Arithmetic: scaling	* / %	left to right
Arithmetic: addition	+ -	left to right
Relational	< <= > >=	left to right
equality relational	== !=	left to right
Assignment & Shorthand arithmetic assignment	= *= /= %= += -=	right to left

Operators: ++ and --

- Can be applied only to an l-value expression:

`Expression:`	`++i`	`i++`	`--i`	`i--`
`Value`	`i+1`	`i`	`i-1`	`i`
`i after evaluation`	`i+1`	`i+1`	`i-1`	`i-1`

`++5`   `--(k+1)`   `++i++`   NO sens

`(++i)++`   `ok`

Floating-point types

- **float**
  - Real numbers with simple precision
  - **sizeof(float)** = 4
  - $10^{-37} \le \text{abs}(f) \le 10^{38}$
  - 6 significant digits
- **double**
  - Real numbers with double precision
  - **sizeof(double)** = 8
  - $10^{-307} \le \text{abs}(f) \le 10^{308}$
  - 15 significant digits

Floating-point types

- **long double**
  - Real numbers with extra double precision
  - **sizeof(long double)** = 12
  - $10^{-4931} \leq \text{abs}(f) \leq 10^{4932}$
  - 18 significant digits
- The limits – in <float.h>
- Operaţii: `+ - * / == != < <= > >=`

Floating-point constants

- Implicitly - **double**

  125.435   1.12E2   123E-2   .45e+6   13.   .56

- In order to be **float** the f or F suffix is needed

  .56f   23e4f   45.54E-1F

- For **long double** the l or L suffix is needed

  123.456e78L

Functions (in <cmath>)

sin	**cos**	**tan**
asin	**acos**	**atan**
sinh	**cosh**	**tanh**
exp	**log**	**log10**
pow	**sqrt**	**ceil**
floor	**fabs**	
ldexp	**frexp**	
modf	**fmod**	

Boolean type (logical)

- Type: **bool**
- range: {**false**, **true**}
- **false** = 0
- **true** = 1 and any other non=zero integer
- Operations: **!** **&&** **||** **==** **!=**
- assignments

```
bool x = 7;     // x becomes “true”
int y = true;   // y becomes 1
```

Logical expressions

Relational_expression ::=

expr1 < expr2 | expr 1 > expr2

| expr1 <= expr2 | expr1 >= expr2

| expr1 == expr2 | expr1 != expr2

Logical_expression *::=*

! *expr*

| expr1 || expr2

| expr1 **&&** *expr2*

Value of relational expressions

`a-b`	`a<b`	`a>b`	`a<=b`	`a>=b`	`a==b`	`a!=b`
Positive	0	1	0	1	0	1
Zero	0	0	1	1	1	0
negative	1	0	1	0	0	1

Value of logical expression ||

exp1	**exp2**	**exp1 || exp2**
<> 0	**Not** evaluated	**1**
= 0	Evaluated	1, if exp2 <> 0 0, if exp2 = 0

Value of logical expression &&

exp1	**exp2**	**exp1 && exp2**
= 0	**Not** evaluated	0
<> 0	Evaluated	1, if exp2 <> 0 0, if exp2 = 0

Examples

- The condition a ≤ x ≤ b is equivalent in C++ with :

```
(x >= a) && (x <= b)
```

```
(a <= x) && (x <= b)
```

- A condition like (a > x or x > b) is equivalent in C++ with :

```
(x < a) || (x > b)
```

```
!(x >= a && x <= b)
```

`void` data type

- the `void` type serves as a unit type, not as a zero or bottom type
- comprises an empty set of values
- type for the result of a function that returns normally, but does not provide a result value to its caller
- the sole argument of a function prototype to indicate that the function takes no arguments
- Conversion of an expression to `void` means that its value is ignored
- when used as a pointer, then it does not specify which data type it is pointing to.

```
void* vague_pointer;
```

- Incomplete type, that cannot be completed

Glossary

- high-level / low-level language
- interpret / compile a program
- source / object code
- parsing
- executable
- token
- header file (library)
- declaration / definition
- syntax / semantics
- compilation / run-time / semantic error
- bug / debugging

Glossary (2)

- data type/s
- expression
- l-/r-value
- keywords
- identifiers
- declaration
- statement
- assignment
- operator
- precedence
- priority
- associativity

If programming languages were vehicles..

C was the great all-arounder: compact, powerful, goes everywhere, and reliable in situations where your life depends on it.

If programming languages were vehicles..

C++ is the new C — twice the power, twice the size, works in hostile environments, and if you try to use it without care and special training you will probably crash.

If programming languages were vehicles..

C# is C++ with more safety features so that ordinary civilians can use it. It looks kind of silly but it has most of the same power so long as you stay near gas pumps and auto shops and the comforts of civilization.

If programming languages were vehicles..

Java is another attempt to improve on C. It sort of gets the job done, but it's way slower, bulkier, spews pollution everywhere, and people will think you're a redneck.

References

- www.cplusplus.com
- www.tiobe.com/
- en.cppreference.com/
- www.greenteapress.com/thinkcpp/index.html
- ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-096-introduction-to-c-january-iap-2011/
- kickassinfographics.com/history/the-evolution-of-computer-languages-infographic/
- crashworks.org/if_programming_languages_were_vehicles/