Cross-platform object oriented C++

- Part 2 – C / C++ fundamentals
- version: 2012_03_05
- By Sergio Barbosa Villas-Boas
  - www.sbVB.com.br
  - sbvb@sbvb.com.br

Introduction

www.sbVB.com.br

Short history the sbVB Cross-platform Object Oriented C++ course

- Some course versions
  - 2001 – First version
  - 2003 – Separation of the course in the 7 parts
  - 2006 – added “Bouncing Shape” and “Bouncing Shape Explicit” example.
  - 2007 – added “functions” and “functions Explicit” example.
  - 2012 – new parts 8 (boost) and 9 (C++11)

www.sbVB.com.br

Some people that created electronic computing

some creators

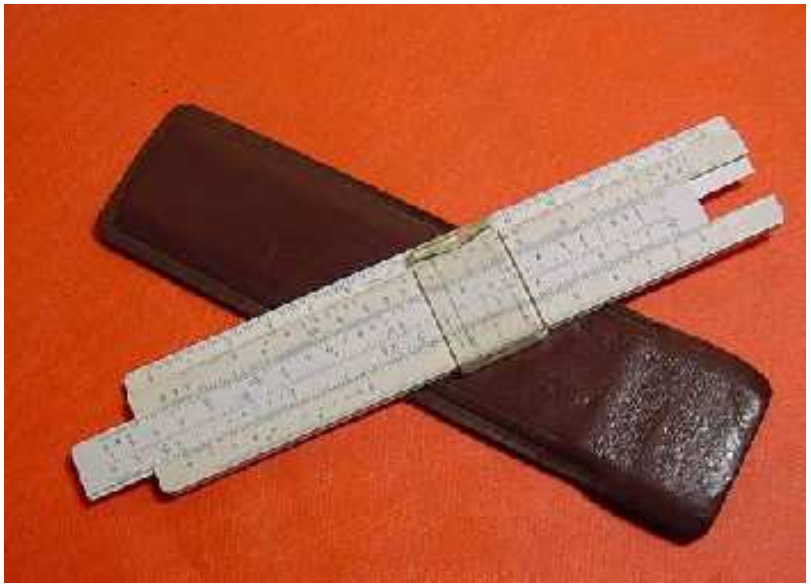
1550 – 1617
John Napier

- Scottish mathematician
- 1614: *Mirifici Logarithmorum Canonis Descriptio*
  - contained fifty-seven pages of explanatory matter and ninety pages of tables of natural logarithms

1574 – 1660
William Oughtred

www.sbVB.com.br

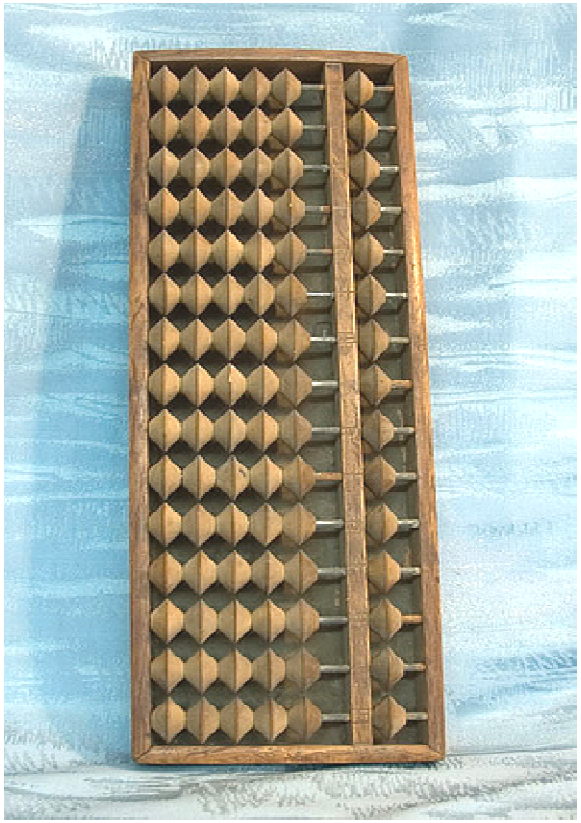
- English
- 1638: Slide rule



www.sbVB.com.br

16th century Japanese soroban

- On November 12, 1946, a contest was held in Tokyo between the Japanese soroban, used by Kiyoshi Matsuzaki, and an electric calculator, operated by US Army Private Thomas Nathan Wood.



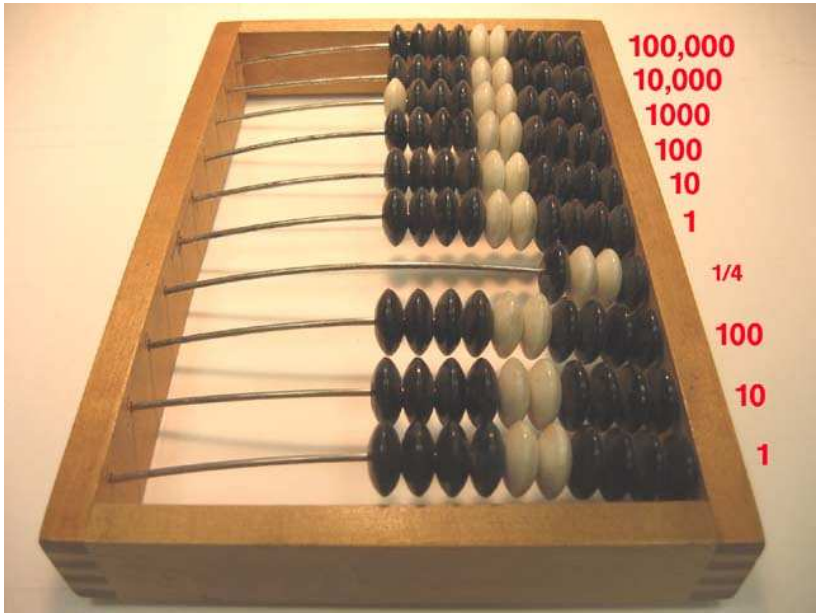
Japanese soroban

算盤

www.sbVB.com.br

17th century Schoty Russian abacus

- Little is known about how it came to be. The schoty is different from other abacuses in that it is not divided into decks. Also, the beads on a schoty move on horizontal rather than vertical wires. Each wire has ten beads and each bead is worth 1 in the ones column, 10 in the tens column, and so on.
- The schoty also has a wire for quarters of a ruble, the Russian currency. The two middle beads in each row are dark colored.
- The schoty shows 0 when all of the beads are moved to the right. Beads are moved from left to right to show numbers. The schoty is still used in modern Russia.

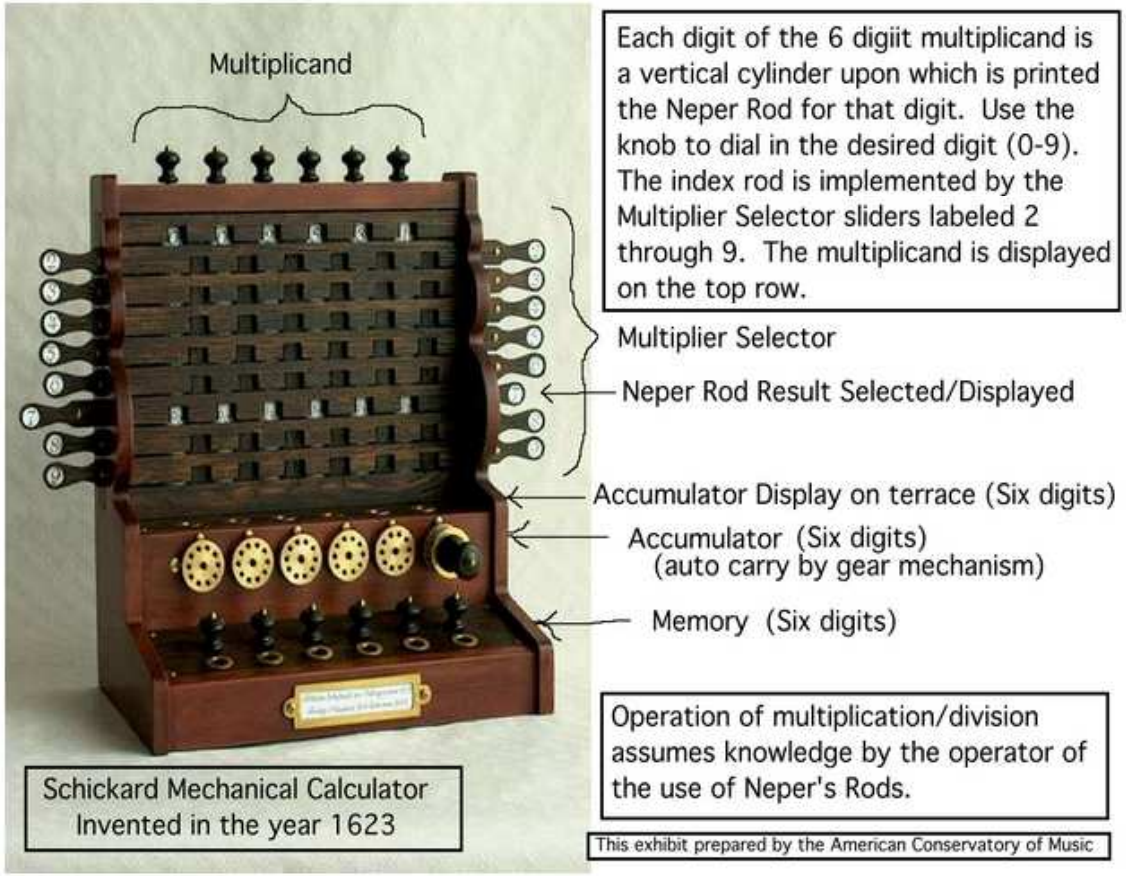


Russian abacus
Schoty

1592 – 1635
Wilhelm Schickard

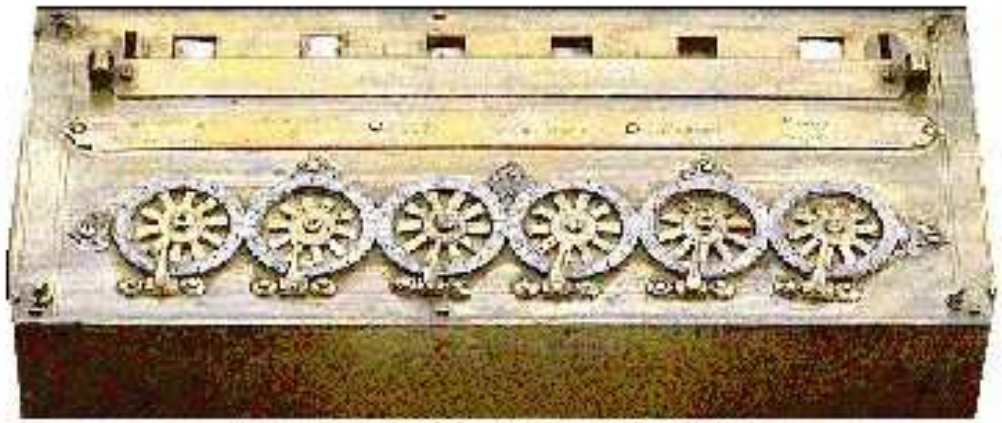
www.sbVB.com.br

- German
- 1623: Schickard mechanical calculator



1623 – 1662
Blaise Pascal

- French
- 1652: pascaline mechanical calculator



1646 – 1716
Gottfried Leibniz

- German
- 1671: Gottfried Leibniz invented a calculating machine which was a major advance in mechanical calculating.
- The Leibniz calculator incorporated a new mechanical feature, the stepped drum — a cylinder bearing nine teeth of different lengths which increase in equal amounts around the drum. Although the Leibniz calculator was not developed for commercial production, the stepped drum principle survived for 300 years and was used in many later calculating systems.

www.sbVB.com.br

1685 – 1731
Brook Taylor

- English mathematician, FRC (Fellow of the Royal Society)
- 1715: *Methodus Incrementorum Directa et Inversa;* It added a new branch to the mathematics, now designated the “calculus of finite differences”.
- This work contains the celebrated formula known as *Taylor's theorem*.

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f^{(3)}(x_0)}{3!}(x - x_0)^3 + \cdots$$

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

1707 – 1783
Leonhard Paul Euler

www.sbVB.com.br

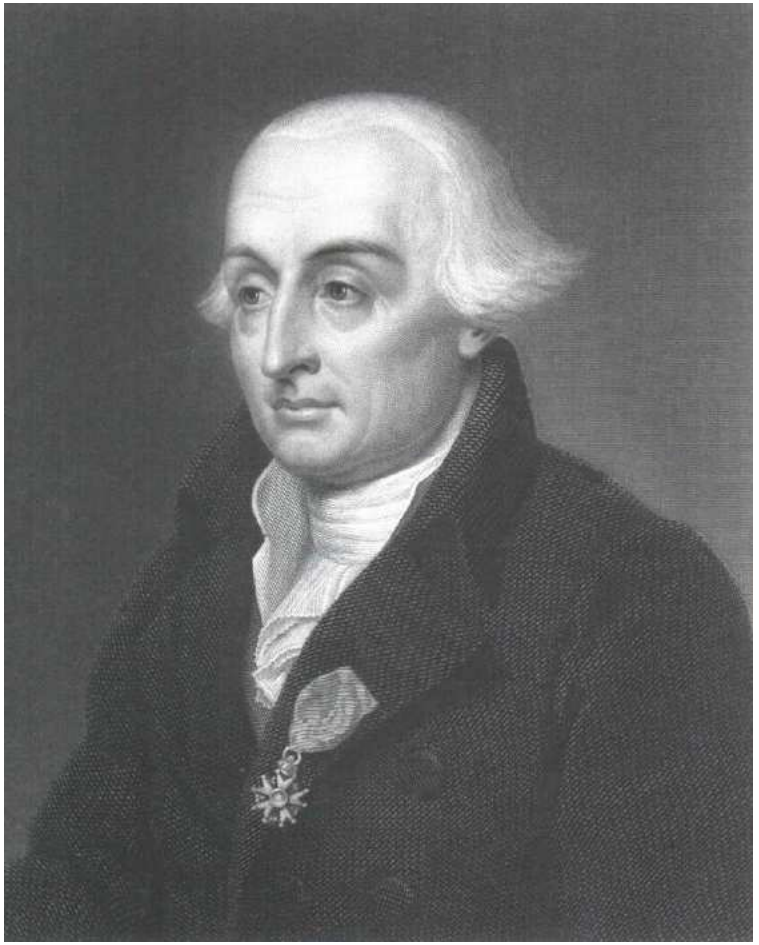
- Swiss mathematician and physicist
- Euler made important discoveries in fields as diverse as infinitesimal calculus and graph theory. He also introduced much of the modern mathematical terminology and notation, particularly for mathematical analysis, such as the notion of a mathematical function.
- He introduced the letter *e* for the base of the natural logarithm (now known as Euler's number), the Greek letter Σ for summations and the letter *i* to denote the imaginary unit sqrt(-1).

$$e^{i\theta} = \cos\theta + i\sin\theta$$

$$e^{i\pi} + 1 = 0$$

1736 – 1813
Joseph-Louis Lagrange

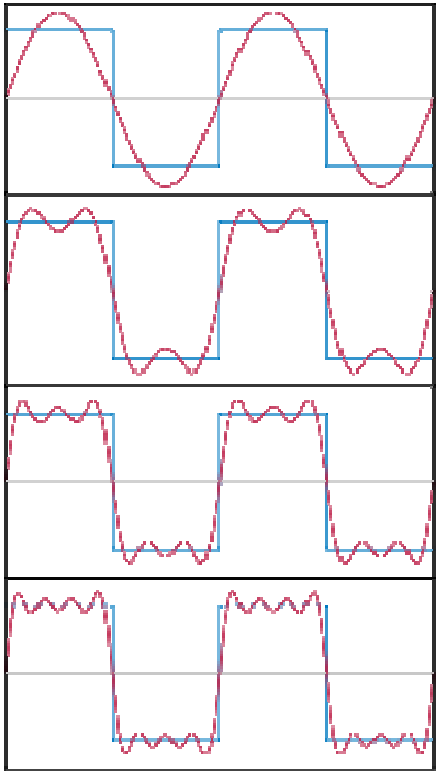
- Italian mathematician and astronomer
- Made significant contributions to all fields of analysis, number theory, classical and celestial mechanics.
- 1772: he realized the power of the Taylor's work and termed it "*le principal fondement du calcul différentiel*" ("the main foundation of differential calculus").



1768 – 1830
Joseph Fourier

www.sbVB.com.br

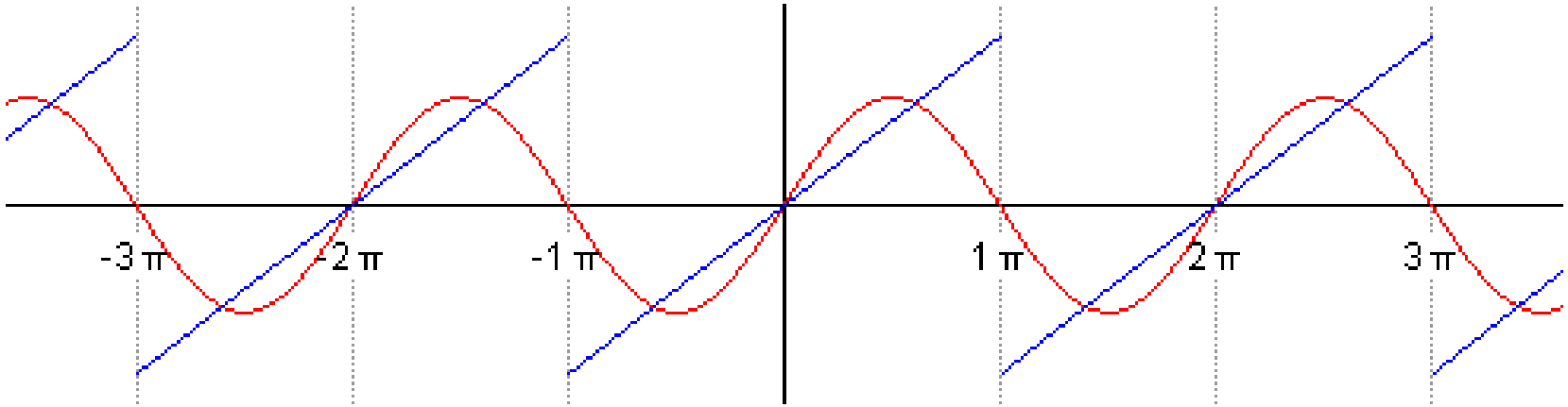
- French mathematician and physicist
- Proposed the Fourier series and Fourier transform.
- Fourier discovered the greenhouse effect.
- Fourier left an unfinished work on determinate equations which was edited by Claude-Louis Navier and published in 1831.



The first four Fourier series approximations for a square wave.

Fourier series example

www.sbVB.com.br

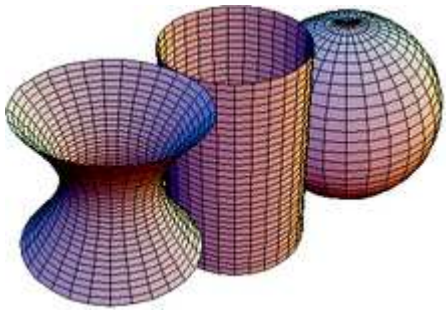


www.sbVB.com.br

1777 – 1855
Carl Friedrich Gauss

- German mathematician and scientist
- works on number theory, statistics, analysis, differential geometry, geodesy, geophysics, electrostatics, astronomy and optics.

- 1799: doctorate: A new proof of the theorem that every integral rational algebraic function of one variable can be resolved into real factors of the first or second degree,

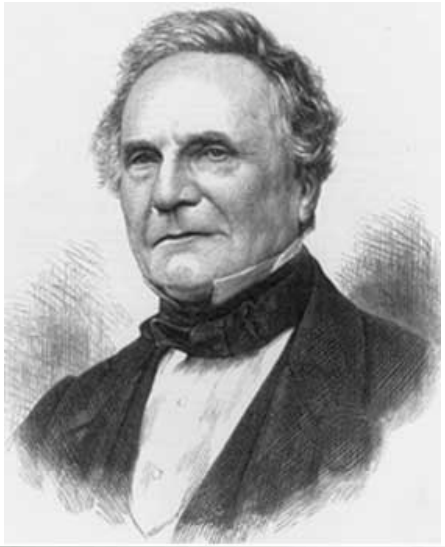


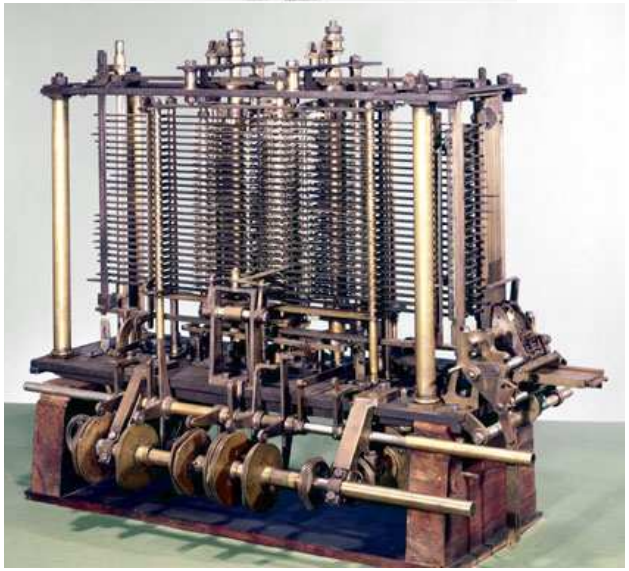
A 10 Deutsche Mark banknote from Germany 1993

1791 – 1871
Charles Babbage

www.sbVB.com.br

- English mathematician, philosopher, inventor
- 1837: Proposed the “analytical engine”, a concept of a programmable (mechanical) computer
- In 1991, a perfectly functioning difference engine was constructed from Babbage's original plans. Built to tolerances achievable in the 19th century, the success of the finished engine indicated that Babbage's machine would have worked.



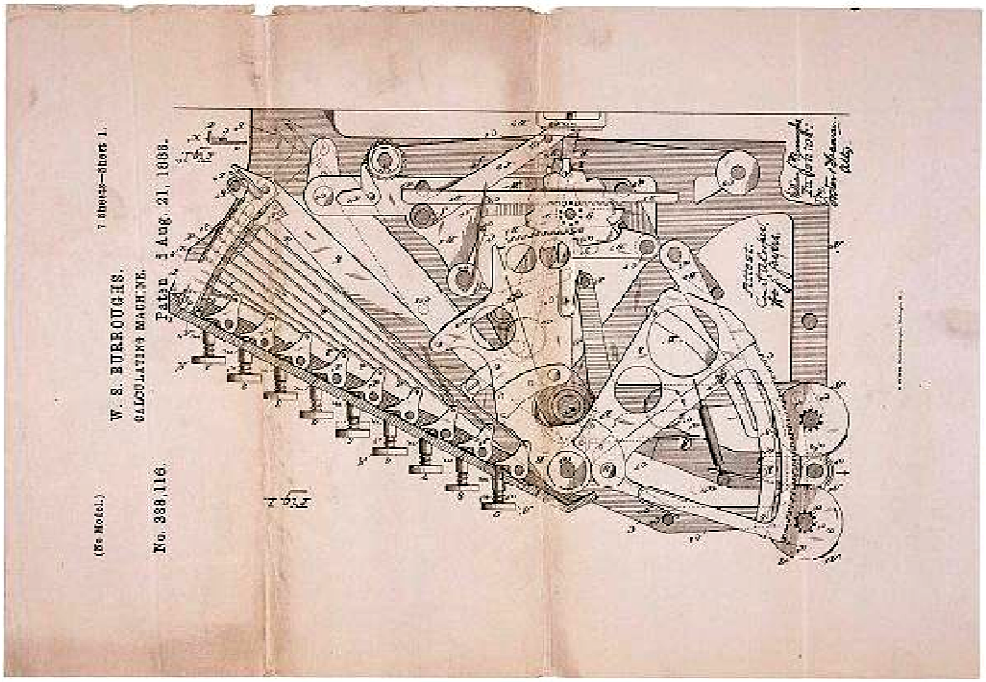


1815 – 1852
Ada Lovelace

- English
- 1842: wrote notes about the Babbage's "analytical engine". These notes were published in 1953.
- The Babbage's analytical engine is recognized as an early model for a computer. Lovelace's notes is a description of a computer and software.that is: **she is the world's first computer programmer**.
- The ADA programming language is named after her.

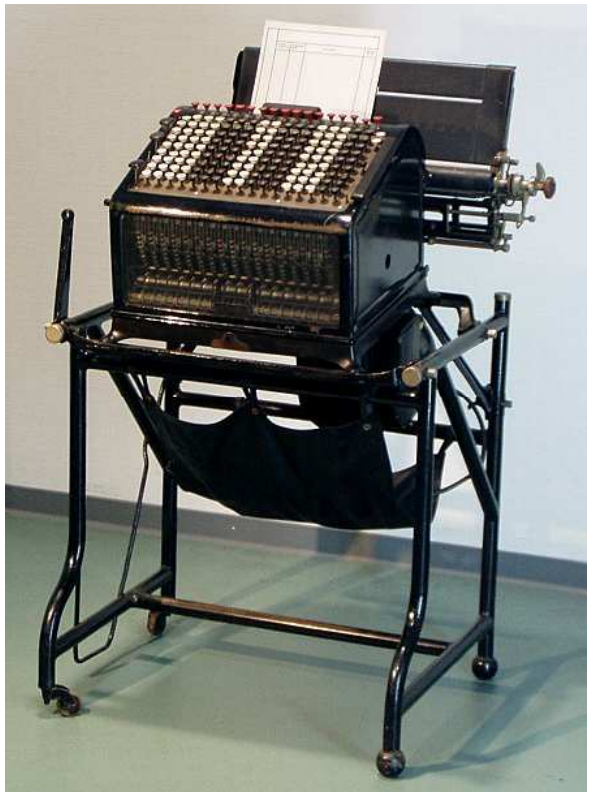
1857 –1898
William Seward Burroughs I

- US inventor and entrepreneur
- 1885: patent for calculating machine



Burroughs Mechanical calculators

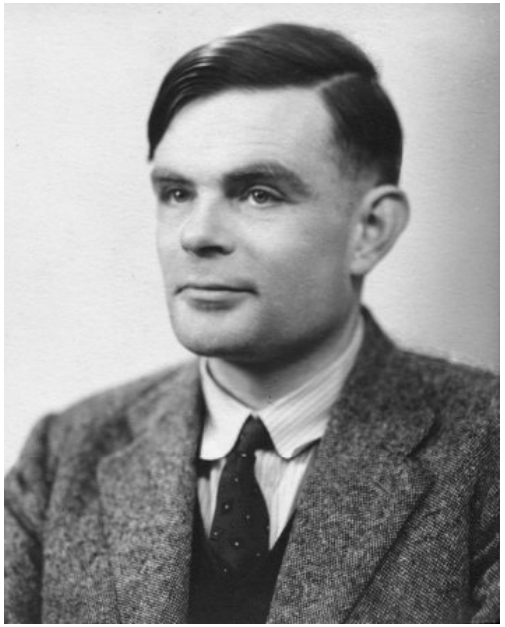
www.sbVB.com.br

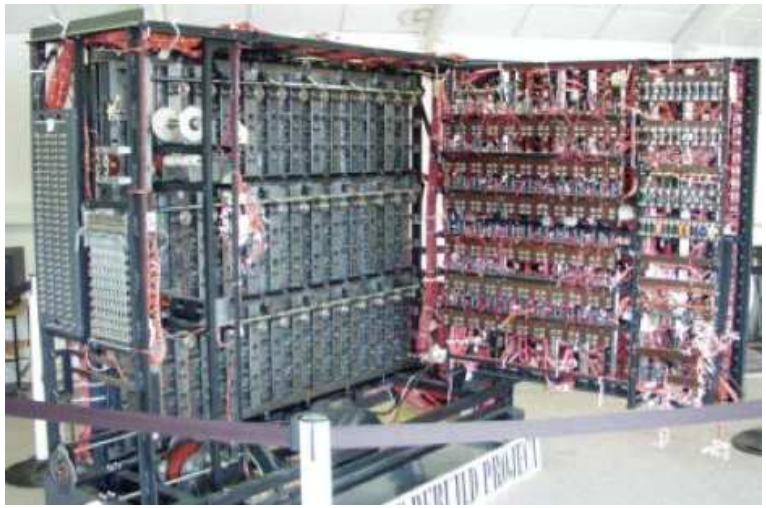


1912 – 1954
Alan Turing

- English mathematician, logician, cryptanalyst, and computer scientist.

- During the Second World War, Turing worked for the Government Code and Cypher School at Bletchley Park, Britain's code breaking centre.
- 1942: Hut 8, developed to break German naval cryptography.





Hut 8

Enigma machine

1933 – William M. Kahan

- Canadian
- Professor Emeritus of EECS, Berkeley

- ~1985: Kahan was the primary architect behind the IEEE 754-1985
- Developed the Kahan summation algorithm, an important algorithm for minimizing error introduced when adding a sequence of finite precision floating point numbers.

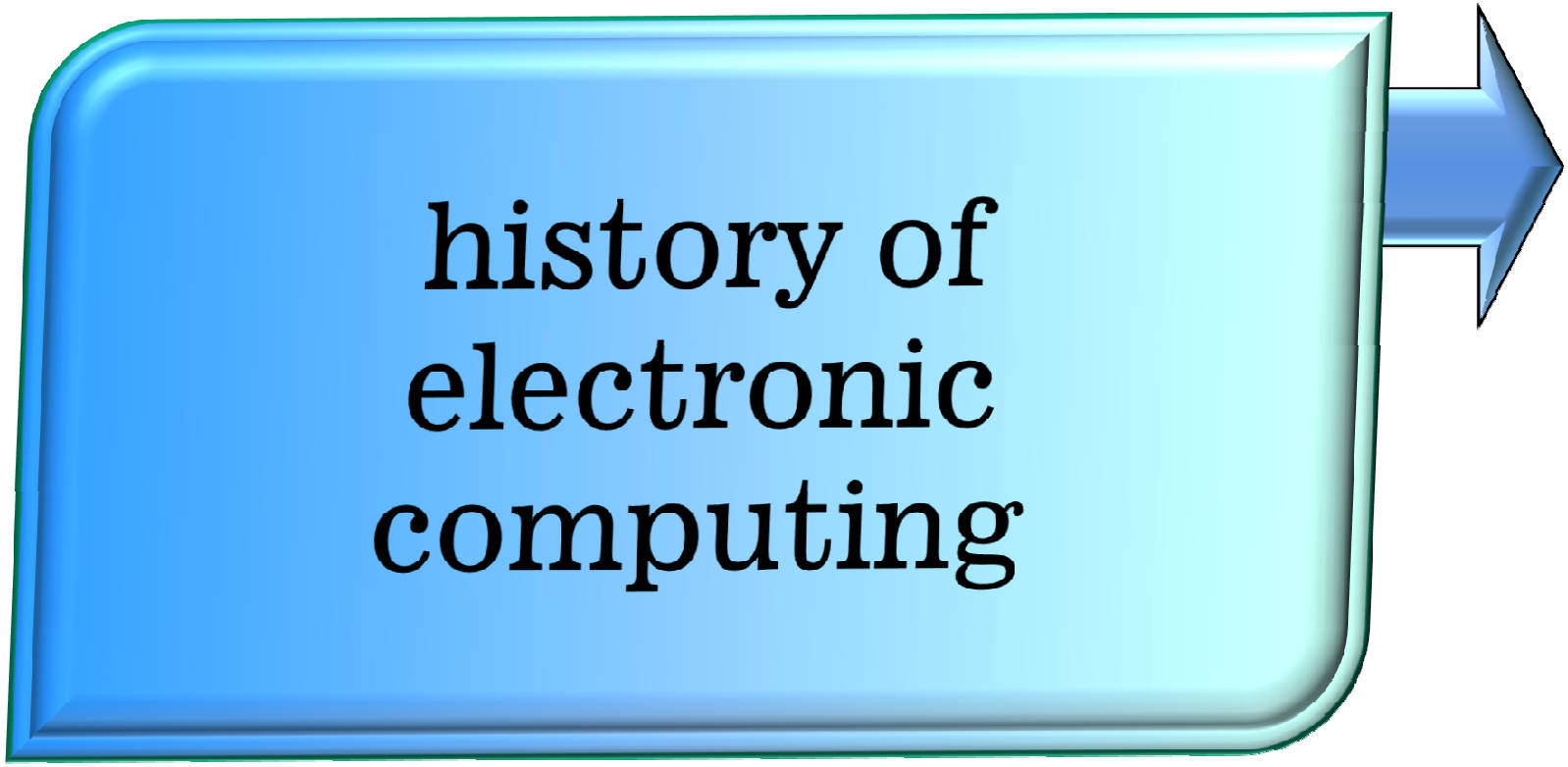
- http://www.eecs.berkeley.edu/Faculty/Homepages/kahan.html
- http://www.cs.berkeley.edu/~wkahan/

brief history of IEEE 754

- 1978: Chip Intel 8086 - CPU
- 1979: Chip Intel 8088 - CPU
- 1980: Chip Intel 8087, math coprocessor for 8088/86.
- 1981: IBM-PC (based on 8088 with socket for 8087)
- 1985: IEEE 754 – 1985
- 2008: IEEE 754 – 2008

- 8087 became an important hardware implementation for the IEEE 754 floating point standard. The 8087 did not implement all IEEE 754 details, as the standard wasn't finished until 1985, but the 80387 did.

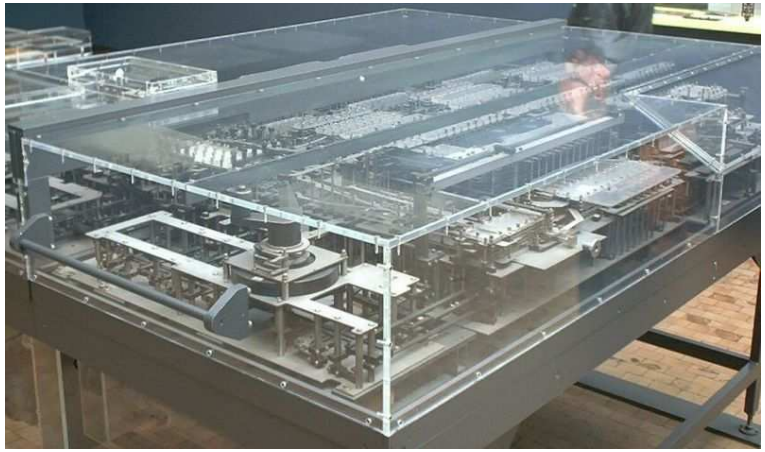
Some moments of history of electronic computing



1938, Z1, Konrad Zuse

- The original Z1 was destroyed by the war in 1943. Z1 was rebuilt for museum in 1989
- It was the first general-purpose digital computer, but it was electromechanical.

- Specifications
- Memory: 64 words of 22 bits
- Clock speed: 1 Hz
- Registers: Two floating-point registers of 22 bit each
- Arithmetic Unit: Four basic operations (add, subtract, multiply, divide) for binary floating point numbers
- Weight: 1,000 kilograms (2,200 lb)
- Average calculation speed: addition 5 seconds, multiplication 10 seconds



picture of museum of Berlin

1941, Z3, Konrad Zuse

- It was the world's first working programmable, fully automatic computing machine

- Specifications
- Frequency: 5.3 Hertz
- Arithmetic Unit: Floating point, 22 bit, add, subtract, multiply, divide, square root
- Average calculation Speed: Addition 0.8 seconds Multiplication 3 seconds
- Power Consumption: Around 4000 watts
- Weight: Around 1,000 kilograms (2,200 lb)
- Elements: Around 2,600 relays
- Memory: 64 words with a length of 22 bits

picture of museum of Berlin

1944, IBM's ASCC

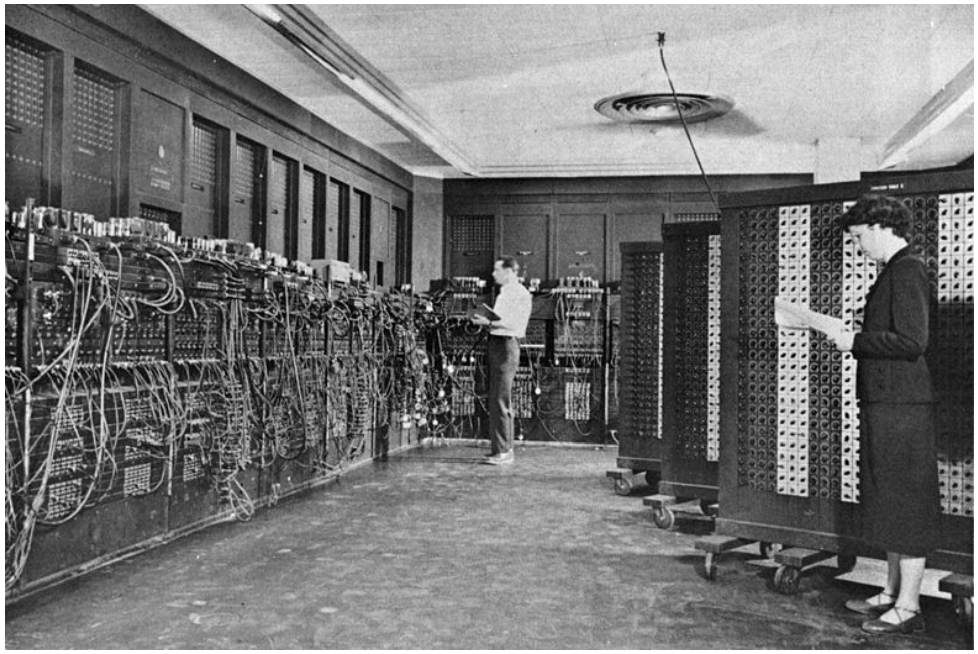
Automatic Sequence Controlled Calculator

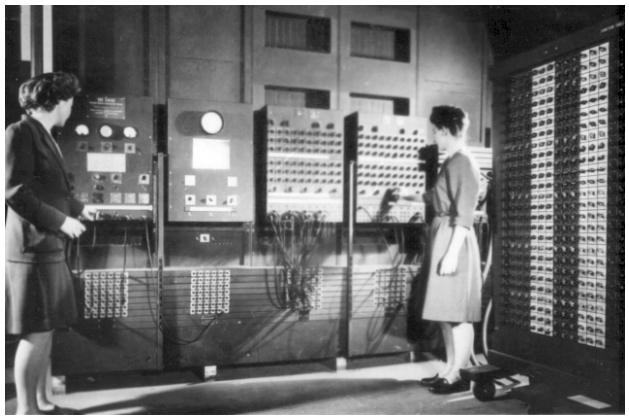
- It solved addition and multiplication problems in less than six seconds. The ASCC was operated by a system of thousands of vacuum tubes.

1946, Eniac

Electronic Numerical Integrator And Computer

- the first general-purpose electronic computer
- Clock speed: 5K Hz
- Programmed directly in binary machine language, using switches



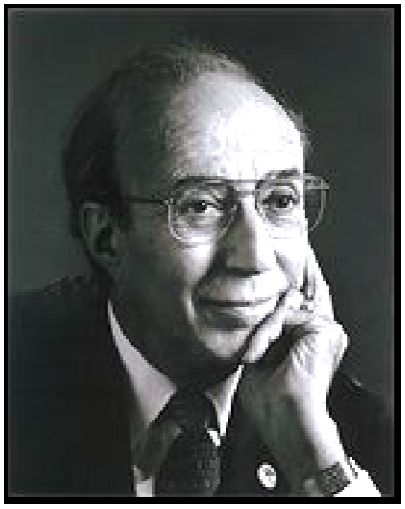


1947 - Numerical Inverting of Matrices of High Order

www.sbVB.com.br

- 1947 – paper by John von Neumann and Herman Goldstine, “Numerical Inverting of Matrices of High Order” . It is one of the first papers to study rounding error and include discussion of what today is called “scientific computing”.

John von Neumann



Herman Goldstine

Fortran moments

www.sbVB.com.br

- Fortran is the first 3GL computer language
  - 1GL is binary (machine level)
  - 2GL is assembly
- 1954 – draft specification for IBM mathematical FORmula TRANslating system
- 1957 – first FORTRAN compiler.
- 1958 - IBM's FORTRAN II
  - introduction of SUBROUTINE, FUNCTION, END, CALL, RETURN, COMMON
- 1959 - COBOL

IBM 704
mainframe (1954)

Fortran moments (2)

www.sbVB.com.br

- 1960 – versions of FORTRAN were available for the IBM 709, 650, 1620, and 7090 computers.
- 1961 – FORTRAN IV
- 1965 – Fortran IV was supposed to be the "standard" and in compliance with American Standards Association X3.4.3 FORTRAN Working Group

console of mainframe IBM 360 (1964)

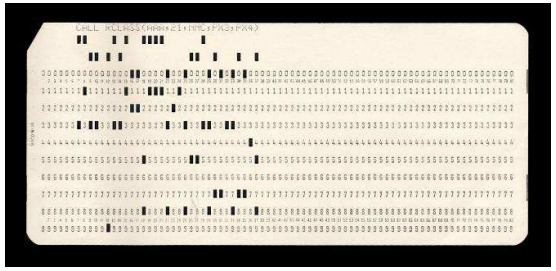
first family of computers
designed to cover the complete
range of applications,
from small to large,
both commercial and scientific

Fortran moments (3)

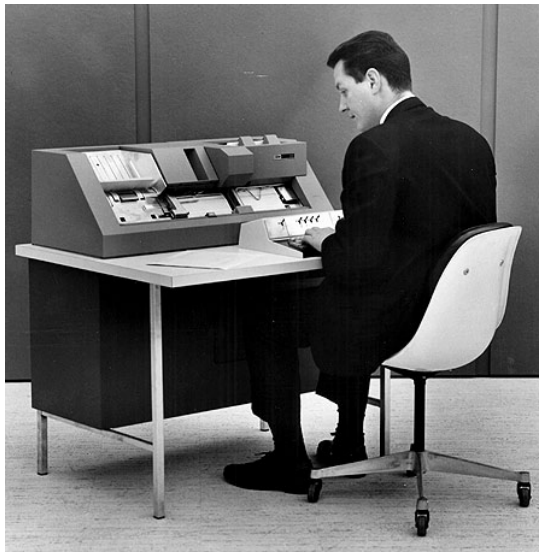
www.sbVB.com.br

- 1966 – FORTRAN 66
  - COMMON, DIMENSION, and EQUIVALENCE, INTEGER, REAL, DOUBLE PRECISION, COMPLEX, DO loops, READ, WRITE, BACKSPACE, REWIND, and ENDFILE statements for sequential I/O, FORMAT statement

- 1977 – FORTRAN 77
  - major upgrade
  - block IF and END IF statements, with optional ELSE and ELSE IF, OPEN, CLOSE, INQUIRE, IMPLICIT



punched card or
hollerith card



IBM 29
card punch machine

Fortran moments (4)

- 1990 – FORTRAN 90
  - Free-form source input, also with lowercase Fortran keywords
  - Identifiers up to 31 characters in length
  - Inline comments
  - whole, partial and masked array assignment statements and array expressions, such as X(1:N)=R(1:N)*COS(A(1:N))
- 1995 – FORTRAN 95
  - minor revision
- 2003 – FORTRAN 2003
  - OO support
  - unicode
- 2008 – FORTRAN 2008
  - minor revision

Some landmarks in history

1957 – first FORTRAN compiler

1960

1959 – first COBOL compiler

1964 – IBM 360

1966 – BCPL (Martin Richards)

1969 – UNICS (father of UNIX)

1970

1970 – B (Ken Thompson)

1972 – C (Dennis Ritchie e Ken Thompson)

1977 – Fortran77

1978 – C K&R (Brian Kernighan e Dennis Ritchie)

1980

1980 – “C with classes” (predecessor of C++), chip 8087

1982– first UFRJ computer class without punched card, using C & PDP/11

1983 – ANSI C, announcement of GNU, UNIX system V

1985 – first version of C++ made available to public by Bjarne Stroustrup
IEEE-754-1985

1990

1990 – Fortran90

tec

Eng.

M.Sc.

www.sbVB.com.br

Some landmarks in history (2)

M.Sc.

Ph.D.

MBA

1990 – Fortran90
1991 – Linux 0.01

1992 – compiler Borland C++ 3.1 (template already implemented)

1994 – STL included to the C++ standard library, Linux kernel 1.0.0

1995 – Begin of popularization of web, Apache http server 1.0

1996 – Linux kernel 2.0.0

1998 – Visual C++ 6.0 with STL

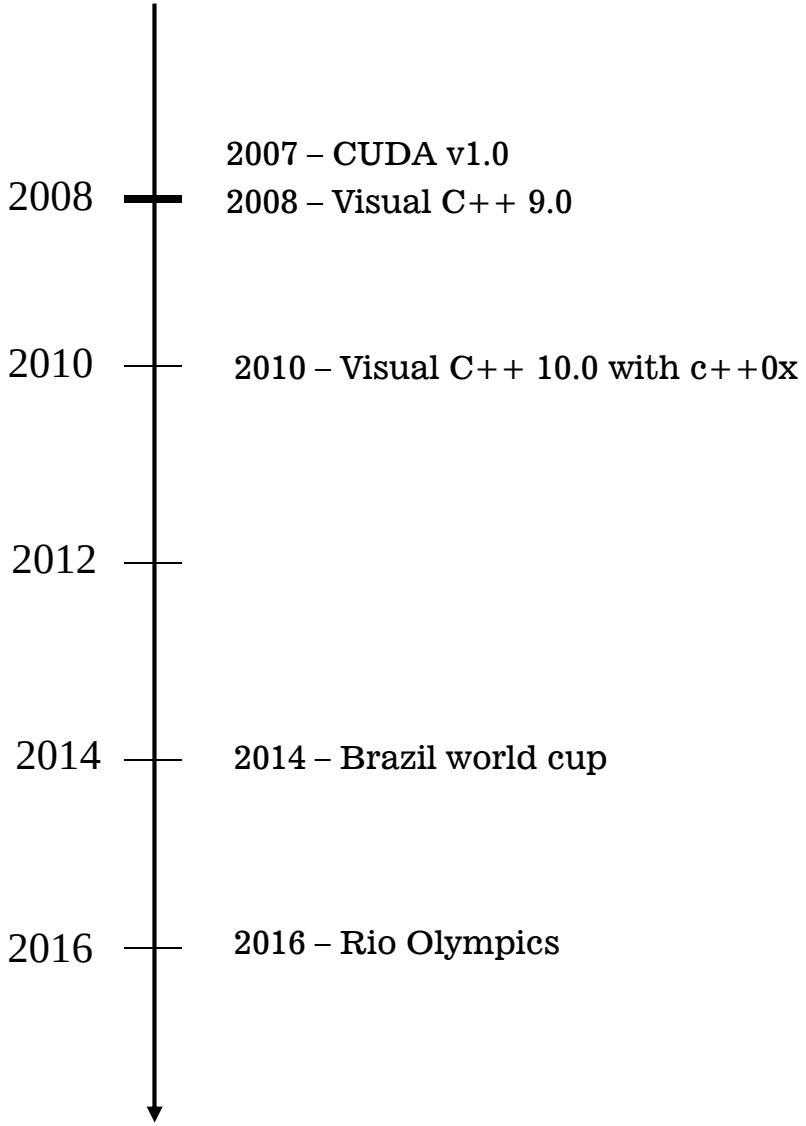
2000 – First version of VBMcgi library (by Sergio Villas-Boas and Alessandro Martins)

2001 – First version of sbVB cross-platform OO C++ course

2003 – Visual C++ 7.0

2005 – Visual C++ 8.0

Some landmarks in history (3)



Humor
learn C++ in 21 days

www.sbVB.com.br

Days 1 - 10
Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...

Days 11 - 21
Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ...

Days 22 - 697
Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.

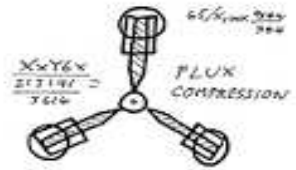
Days 698 - 3648
Interact with other programmers. Work on programming projects together. Learn from them.

Days 3649 - 7781
Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.

Days 7782 - 14611
Teach yourself biochemistry, molecular biology, genetics,...

Day 14611
Use knowledge of biology to make an age-reversing potion.

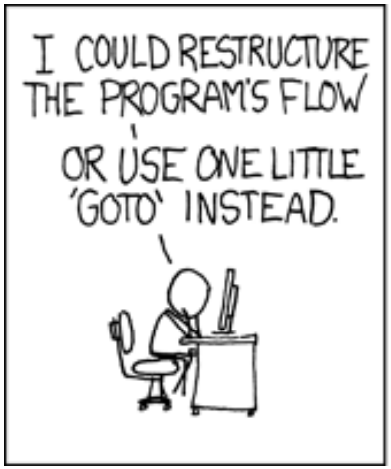
Day 14611
Use knowledge of physics to build flux capacitor and go back in time to day 21.

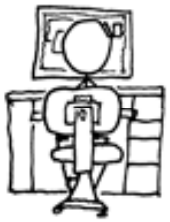


Day 21
Replace younger self.

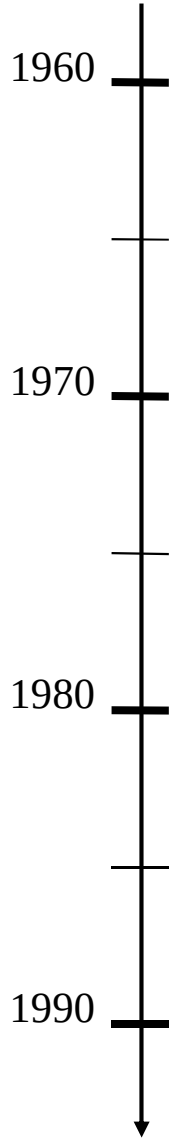
Humor
goto ok to use?

www.sbVB.com.br





Short history of C / C++



1966 – BCPL (Martin Richards)

1969 – UNICS (father of UNIX)

1970 – B (Ken Thompson)

1972 – C (Dennis Ritchie e Ken Thompson)

1978 – C K&R (Brian Kernighan e Dennis Ritchie)

1980 – “C with classes” (predecessor of C++)

1983 – ANSI C, announcement of GNU, UNIX system V

1985 – first version of C++ made available to public by Bjarne Stroustrup

www.sbVB.com.br

Short history of C / C++ (2)

1990

1991 – Linux 0.01

1992 – compiler Borland C++ 3.1 (template already implemented)

1994 – STL included to the C++ standard library, Linux kernel 1.0.0

1995 – Begin of popularization of web, Apache http server 1.0

1996 – Linux kernel 2.0.0

1998 – Visual C++ 6.0

2000

2000 – First version of VBMcgi library (by Sergio Villas-Boas and Alessandro Martins)

2001 – First version of sbVB cross-platform OO C++ course

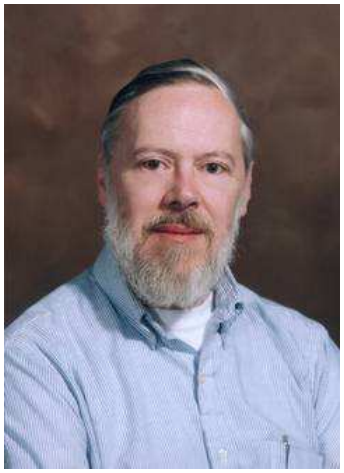
2003 – Visual C++ 7.0

2005 – Visual C++ 8.0

Gallery of creators

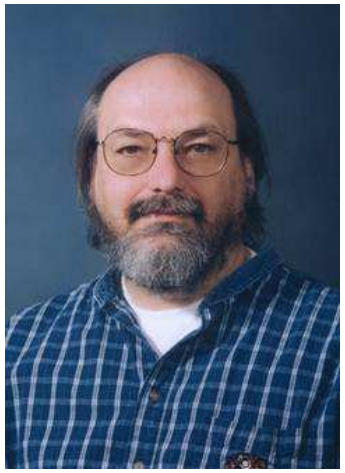
www.sbVB.com.br

Martin Richards



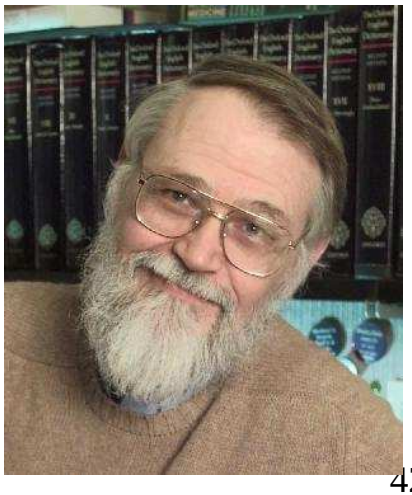
Dennis Ritchie

Richard Stallman



Ken Thompson

Bjarne Stroustrup



Brian Kernighan

www.sbVB.com.br

The interfaces of the applications

- These are the most used application interfaces
  - Console scroll (TTY)
  - Console full screen
  - GUI
  - Web / cgi (special case of client-server architecture)
  - web services (good for cloud computing)
  - Touch (for mobile devices)

www.sbVB.com.br

Console scroll (TTY)

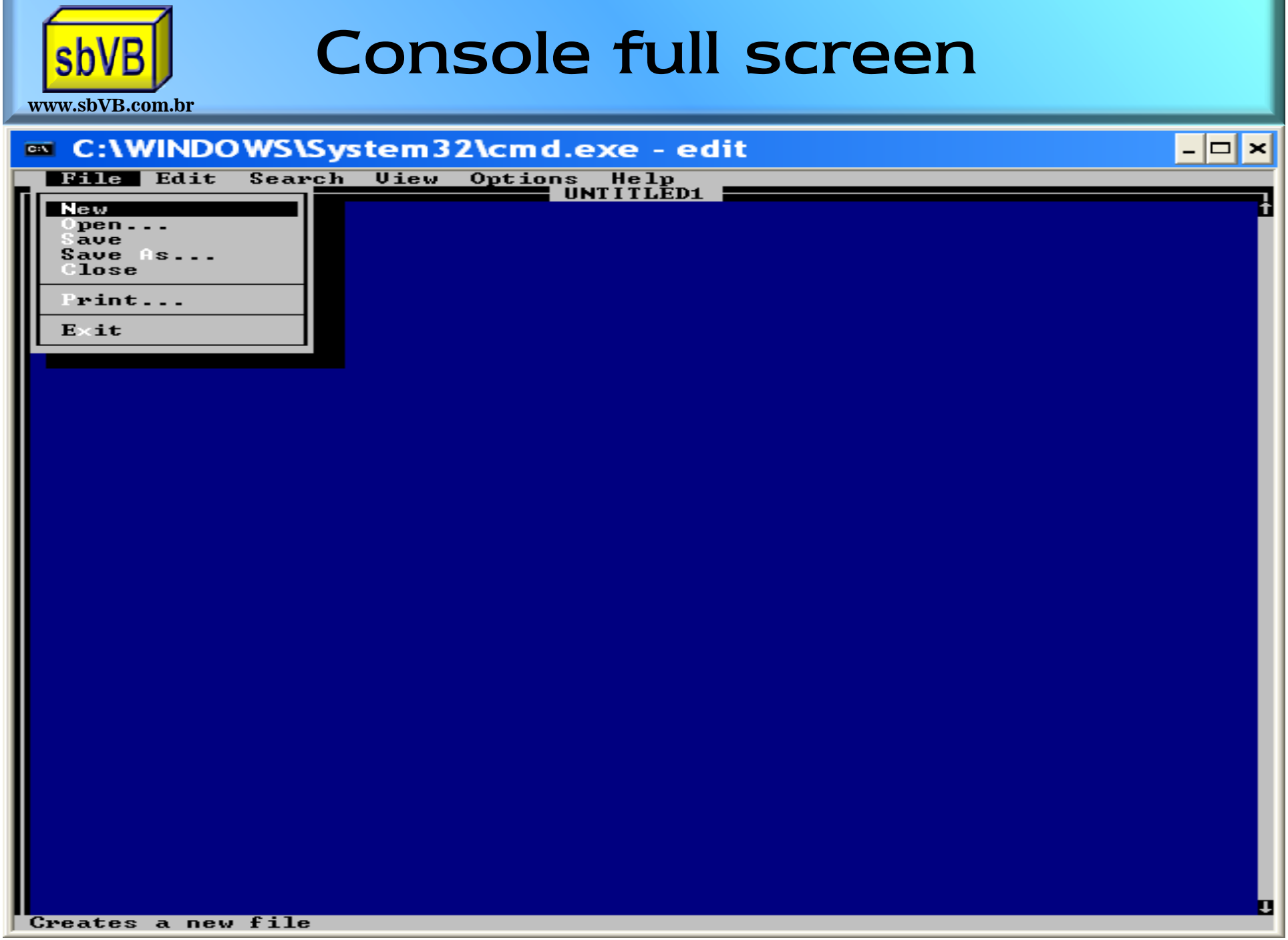
zeus.del.ufrj.br - default - SSH Secure Shell

File Edit View Window Help

Quick Connect Profiles

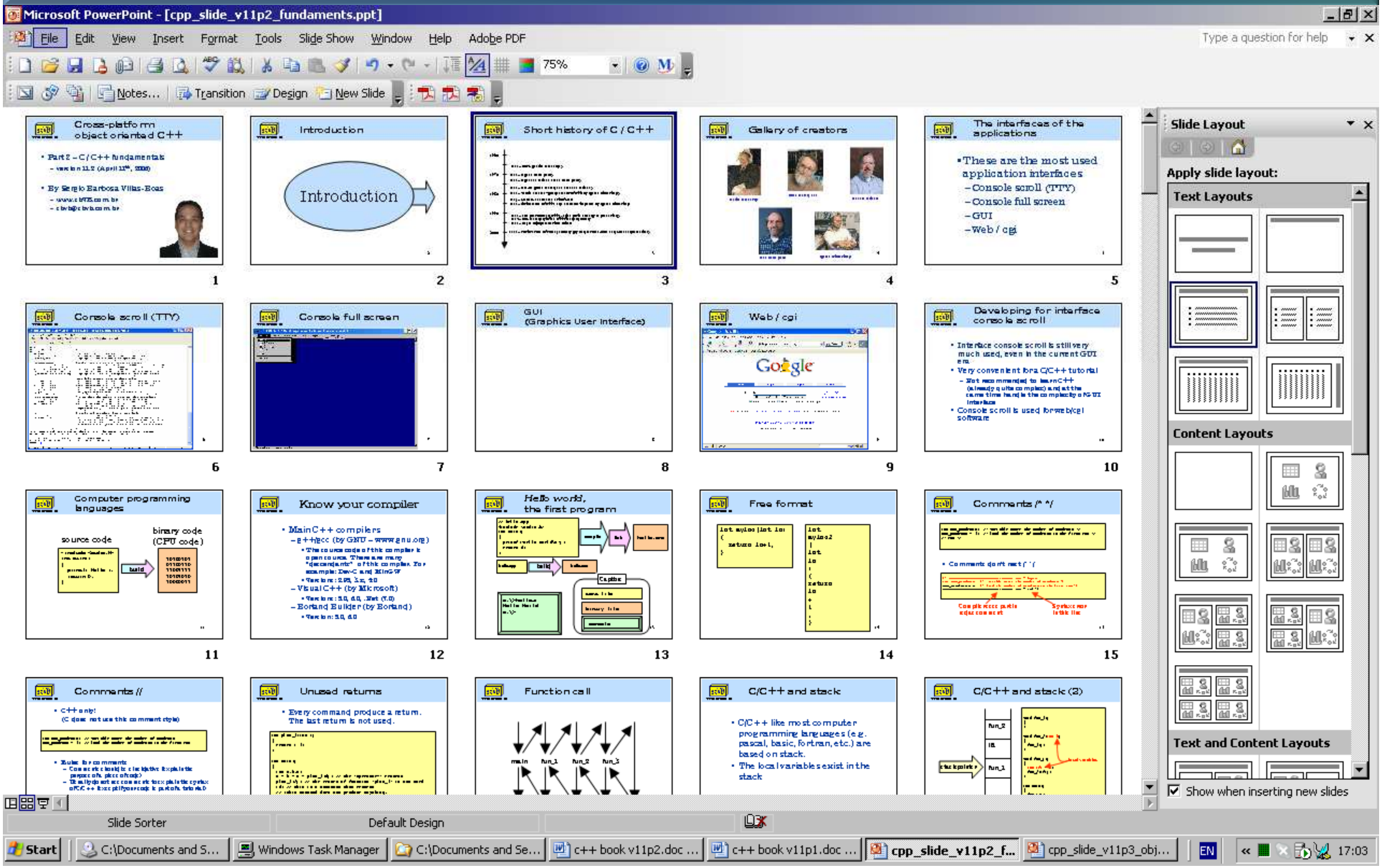
```
zeus% g++ --help
Usage: g++ [options] file...
Options:
  --help                    Display this information
  (Use '-v --help' to display command line options of sub-processes)
  -dumpspecs                Display all of the built in spec strings
  -dumpversion              Display the version of the compiler
  -dumpmachine              Display the compiler's target processor
  -print-search-dirs        Display the directories in the compiler's search path
  -print-libgcc-file-name   Display the name of the compiler's companion library
  -print-file-name=<lib>    Display the full path to library <lib>
  -print-prog-name=<prog>   Display the full path to compiler component <prog>
  -print-multi-directory    Display the root directory for versions of libgcc
  -print-multi-lib          Display the mapping between command line options and
                             multiple library search directories
  -Wa,<options>             Pass comma-separated <options> on to the assembler
  -Wp,<options>             Pass comma-separated <options> on to the preprocessor
  -Wl,<options>             Pass comma-separated <options> on to the linker
  -Xlinker <arg>            Pass <arg> on to the linker
  -save-temps               Do not delete intermediate files
  -pipe                     Use pipes rather than intermediate files
  -specs=<file>             Override builtin specs with the contents of <file>
  -std=<standard>           Assume that the input sources are for <standard>
  -B <directory>            Add <directory> to the compiler's search paths
  -b <machine>              Run gcc for target <machine>, if installed
  -V <version>              Run gcc version number <version>, if installed
  -v                        Display the programs invoked by the compiler
  -E                        Preprocess only; do not compile, assemble or link
  -S                        Compile only; do not assemble or link
  -c                        Compile and assemble, but do not link
  -o <file>                 Place the output into <file>
  -x <language>             Specify the language of the following input files
                             Permissable languages include: c c++ assembler none
                             'none' means revert to the default behaviour of
                             guessing the language based on the file's extension

Options starting with -g, -f, -m, -O or -W are automatically passed on to
the various sub-processes invoked by g++.  In order to pass other options
on to these processes the -W<letter> options must be used.

For bug reporting instructions, please see:
<URL:http://www.gnu.org/software/gcc/bugs.html>.
zeus%
```

Connected to zeus.del.ufrj.br | SSH2 - aes128-cbc - hmac-md5 | 92x43 | NUM


www.sbVB.com.br
Console full screen
C:\WINDOWS\System32\cmd.exe - edit
File Edit Search View Options Help
UNTITLED1
New
Open...
Save
Save As...
Close
Print...
Exit
Creates a new file

GUI (Graphics User Interface)

www.sbVB.com.br



Web / cgi

www.sbVB.com.br

www.sbVB.com.br

Developing for interface console

- Console interface is still very much used, even in the current GUI era
- Console interface is very convenient for a C/C++ tutorial
  - Not recommended to learn C++ (already quite complex) and at the same time handle the complexities of GUI interface
- Console is used for web/cgi software

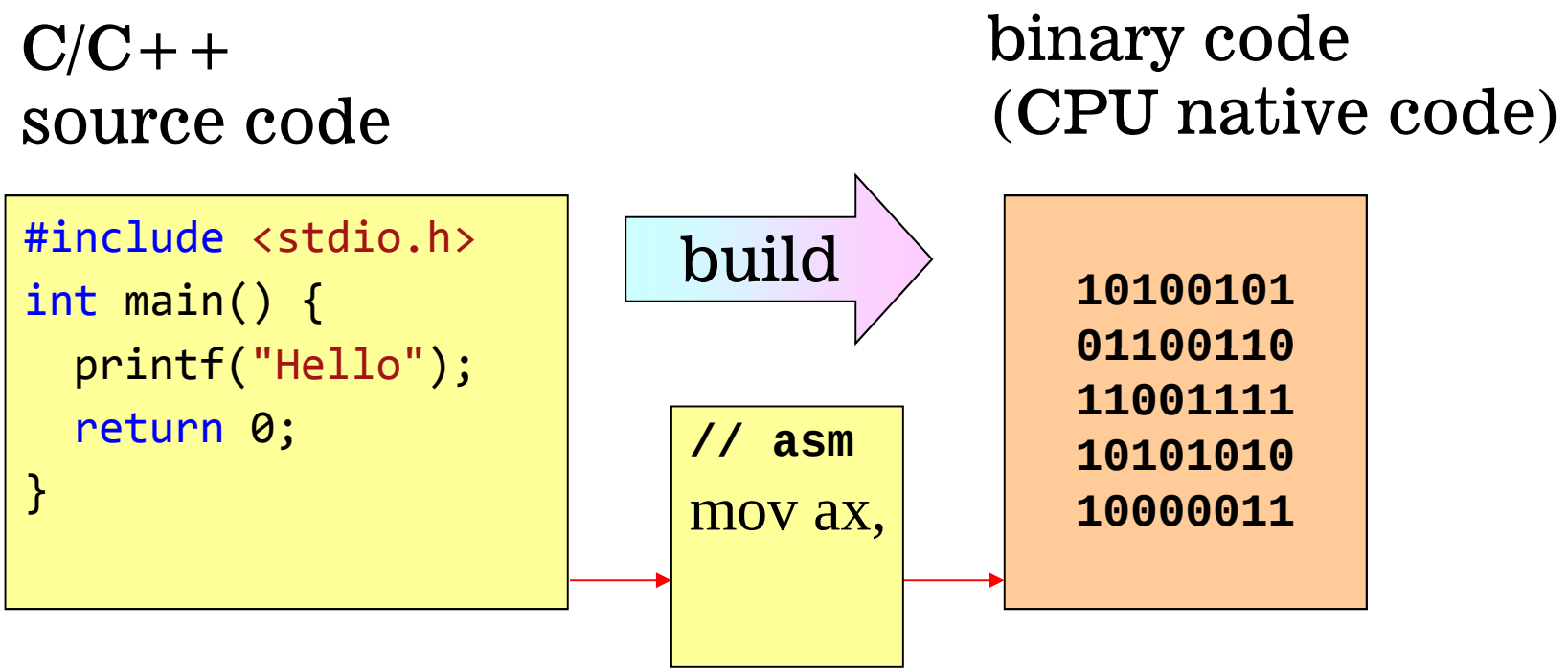
www.sbVB.com.br

Hello word in python

```
# hello.py
print "Hello World"
```

```
c:\>python hello.py↵
Hello World
c:\>
```

Computer programming languages, building native code



build = compile + link

www.sbVB.com.br

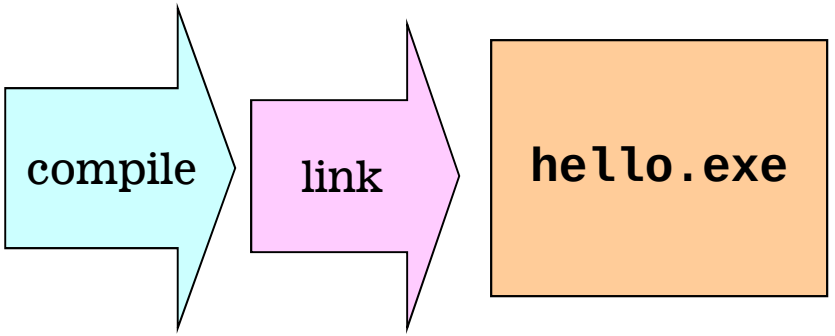
Some C++ compilers

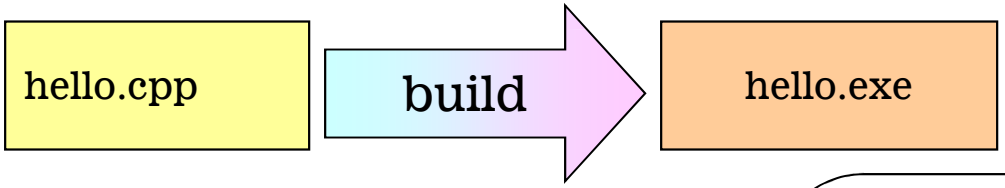
- g++/gcc (by GNU – www.gnu.org)
  - The source code of this compiler is open source. There are many "descendants" of this compiler. For example: Dev-C and MinGW
  - Old versions: 2.97, 3.x, 4.x
  - Latest 4.5.2
- Visual C++ (by Microsoft)
  - Old versions: 6 (1998), 7 (2003), 8 (2005), 9 (2008),
  - Latest: 10 (2010)
- Intel C++ (by Intel)
  - For linux and Windows

Hello world, the first program

www.sbVB.com.br

```c
// hello.c, C style, not recommended
#include <stdio.h>
int main() {
  printf("Hello World\n");
  return 0;
}
```






```
c:\>hello↵
Hello World
c:\>
```

Caption:

- text file
- binary file
- console

www.sbVB.com.br

Hello world, C++ style

```
// hello.cpp, C++ style, recommended
#include <iostream>
using namespace std;
int main()
{
  cout << "Hello World" << endl;
  return 0;
}
```

```
c:\>hello↵
Hello World
c:\>
```

Free format

```
int myinc(int in) {
  return in+1;
}
```

```
int
myinc2
(
int
in
)
{
return
in
+
1
;
}
```

Comments /* */

```
int num_students;  /* variable store the number of students */
num_students = 4;  /* load the number of students in the first row */
/* etc */
```

- Comments don't nest /* */

```
/* ======================== comment out - begin
int num_students;  /* variable store the number of students */
num_students = 4;  /* load the number of students in the first row */
============================ comment out - end */
```

Compiler sees part in green as comment

Syntax error in this line

www.sbVB.com.br

Comments //

- C++ only!
  (C does not use this comment style)

```
int num_students;  // variable store the number of students
num_students = 4;  // load the number of students in the first row
```

- Rules for comments
  - Comments should be elucidative (explain the purpose of a piece of code)
  - Usually do not use comments to explain the syntax of C/C++ (except if your code is part of a tutorial)

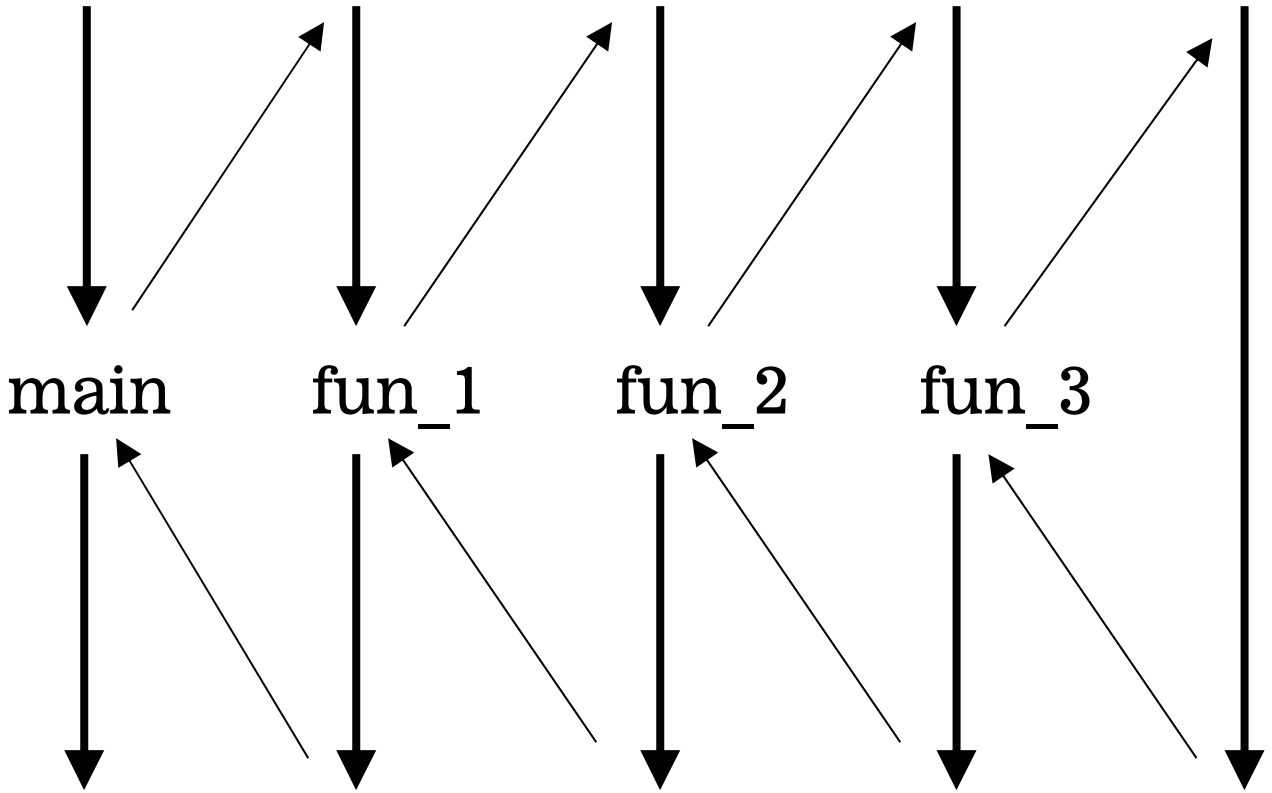
www.sbVB.com.br

Unused returns

- Every command produce a return.
- There's a LUR – Limbo of Unused Returns, that receives the return of every command.

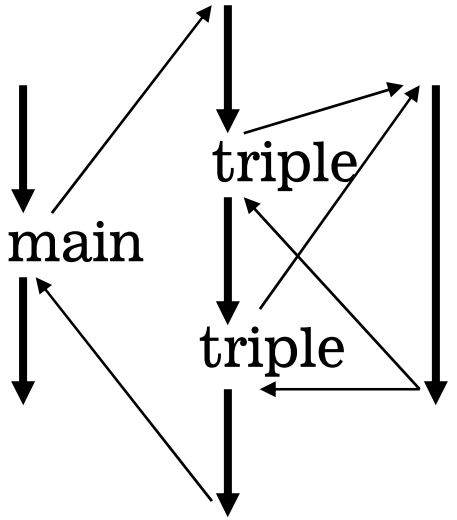
```
int plus_3(int i) {
  return i+3;
}

int main() {
  int a,b,c;
  a = b = c = plus_3(6); // the "operator=" returns
  plus_3(2); // the return of function "plus_3" is not used
  10; // this is a constant that returns
  // (this command does not produce anything,
  // but it is not a syntax error)
  return 0; // return of the main function
}
```

Functions



Functions (2)

```
double triple(double z) {
  return 3*z;
}

int main() {
  double d = 2.2;
  double dd = triple(d);
  triple(d);  // return not used
    // in this case, nothing is produced,
    // but this is legal
  return 0;
}
```



www.sbVB.com.br

Function examples

```
// return the square of the sin of input
double sin2(double x) {
  double ret = sin(x);
  ret *= ret; // ret = ret * ret;
  return ret;
}

// Usage example
int main() {
  double d = 0.4;
  double f = sin2(d);
  return 0;
}
```

www.sbVB.com.br

Function examples (2)

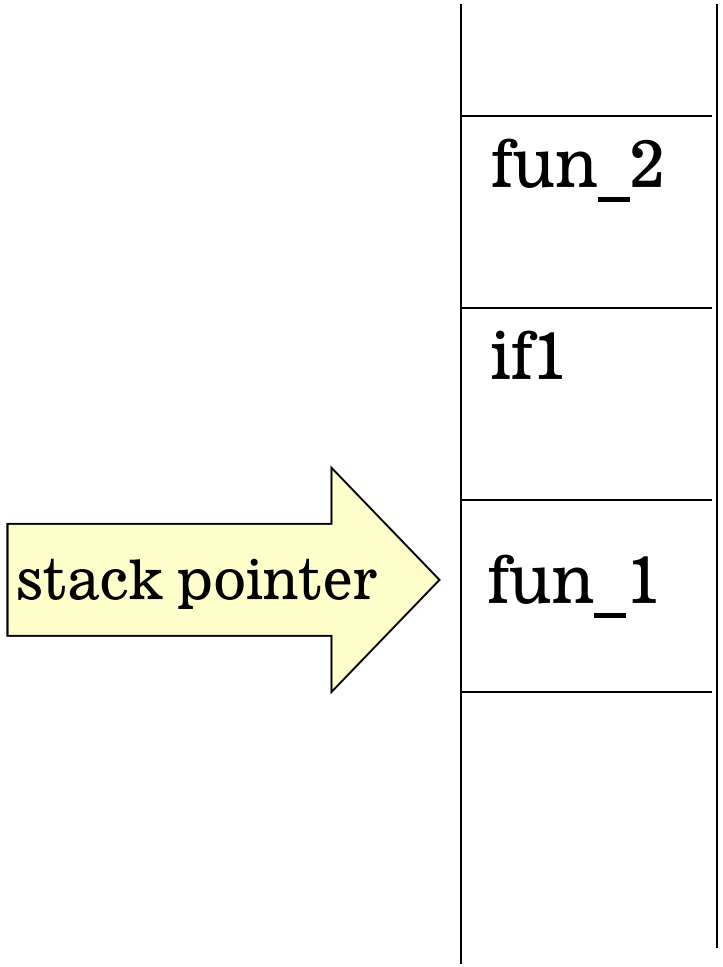
```
// return the mean of the 3 arguments
double mean3(double a, double b, double c)  {
  double ret = (a+b+c)/3;
  return ret;
}

// Usage example
int main() {
  double f = mean3(1.1, 2.2, 3.3);
  return 0;
}
```

www.sbVB.com.br

C/C++ and stack

- C/C++ like most computer programming languages (e.g. pascal, basic, fortran, etc.) are based on stack.
- The local variables exist in the stack

C/C++ and stack (2)



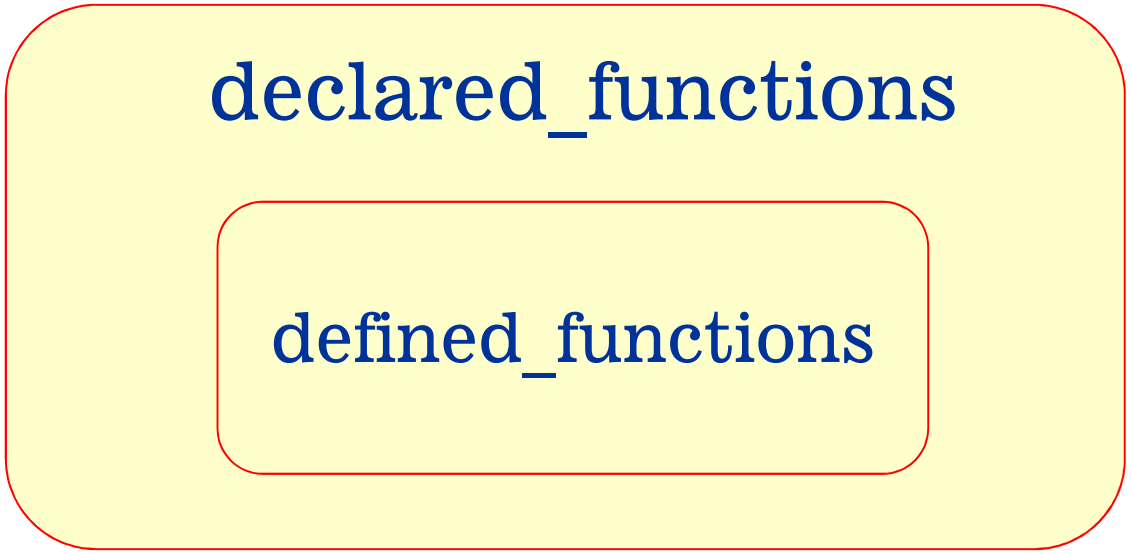

```
void fun_3()
{
}

void fun_2(int k)
{
  fun_3();
}

void fun_1()
{
  int if1 = 1;
  fun_2(if1);
}

int main()
{
  fun_1();
  return 0;
}
```

local variables

declaration × definition

- The definition declares
- The declaration does not define
- It is possible for a function to be declared and not defined, but not the opposite

$$declared\_functions \supset defined\_functions$$



declaration × definition (2)

```
// prototype of a function (declaration)
<returnType> funName(<par1, par2, etc>);

// description of a function (definition)
<returnType> funName (<par1, par2, etc>)
{
// code
}
```

```
// Declaration example
void printAge(int age);
// there is no return,
// that is, the return is "void"

// Definition example
void printAge(int age) {
  cout << "Your age = " << age << endl;
}
```

declaration × definition (3)

```
// declaration

int plus_3(int i);
```

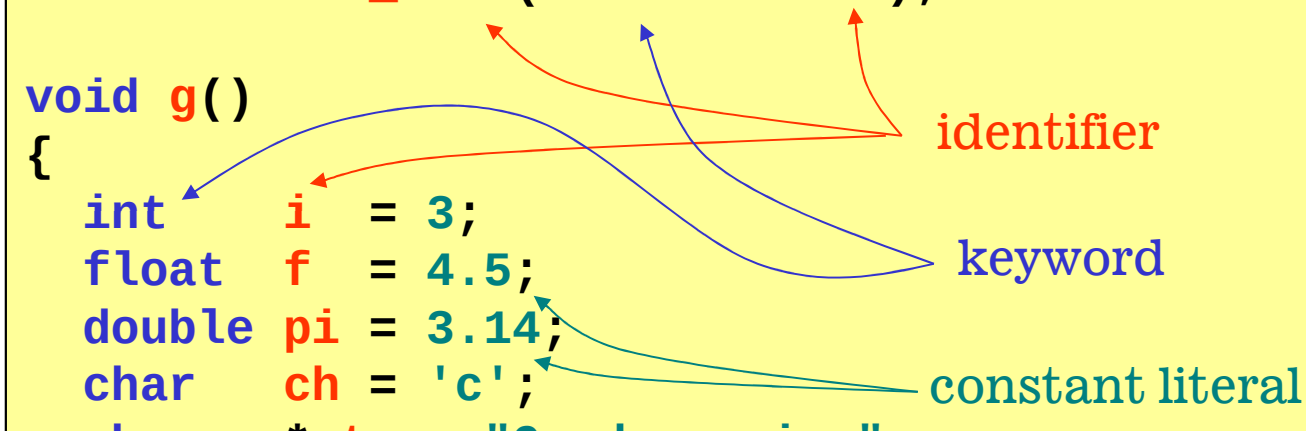
```
// definition
// (includes declaration)

int plus_3(int i)
{
  return i+3;
}
```

function prototype

the code that implements the function

Identifier, literal constant and keyword

```
float circle_area(float radius);

void g()
{
  int    i  = 3;
  float  f  = 4.5;
  double pi = 3.14;
  char   ch = 'c';
  char   *str = "Good morning";
  // a string constant returns a pointer to char
}
```



www.sbVB.com.br

C++ keywords

asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	else	inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

www.sbVB.com.br

C++ keywords (2)

```
void continue() // ERROR ! the function name is a keyword
{
  printf("continue");
  // this is no error, since the keyword
  // is inside a string;
  // thus it is the value of a literal constant
}
```

www.sbVB.com.br

Auxiliary letters

Letters used in operators

```
->    ++    --    .*    ->*    <<    >>    <=    >=    ==    !=
&&    ||    *=    /=    %=     +=    -=    <<=   >>=   &=    ^=
|=    ::
```

Letters used for punctuation

```
!     %     ^     &     *     (     )     -     +     =     {
}     |     ~     [     ]     ;     ’     :     "     <     >
?     ,     .     /
```

Standard data types (for 32bit cpu's)

Data type	# of bytes	Range
bool	1	true .. false
char	1	-128 .. 127
unsigned char	1	0 .. 255
short int	2	-32768 .. 32767
short unsigned	2	0 .. 65535
int, long int	4	-2 147 483 648 .. 2 147 483 647
unsigned, long unsigned	4	0 .. 4 294 967 295

$2^7 - 1$

$2^8 - 1$

$2^{15} - 1$

$2^{16} - 1$

$2^{31} - 1$

$2^{32} - 1$

-2^7 -2^{15} -2^{31}

www.sbVB.com.br

Standard data types (for 64bit cpu's)

Data type	# of bytes	Range
bool	1	true .. false
char	1	-128 .. 127
unsigned char	1	0 .. 255
short int	2	-32768 .. 32767
short unsigned	2	0 .. 65535
int	4	-2 147 483 648 .. 2 147 483 647
unsigned	4	0 .. 4 294 967 295
long int	8	-1 152 921 504 606 846 977 ..
long unsigned	8	0 .. 18 446 744 073 709 551 615

www.sbVB.com.br

Summary of floating point

precision	in C	in fortran	Bytes	CPU
half binary16			2	?
normal binary32	float	real(kind=4)	4	x86
double binary64	double	real(kind=8)	8	x86
extended	long double (2)	real(kind=10)	10 (1)	x86
quadruple binary128		real(kind=16)	16	sparc

(1) usually with alignment to 12 or to 16
(2) in Visual C++ “long double” is implemented as “double”

Variable instantiation

```
<data_type> <list_of_user_identificators>;
```

```
int i; // global variable, defined outside a function (global scope)

void fun_1()
{
  char ch; // local variable of fun_1
  double ch; // error, ch already defined
  float f1, f2, f3; // variables of type float
}

void fun_2()
{
  double ch; // local variable of fun_2 (don't confuse with ch of fun_1)
}
```

Scope

```
int i; // global variable, defined outside a function (global scope)

void f1()
{
  int i; // declaration of variable "i"
  i = 3; // set a constant to a variable

  {  // open inner scope
    int i; // variables in the inner scope are different from
           // variables with the same name outside the scope

    i = 5;   // set a constant to the inner "i"
  } // close inner scope

  // what happened in the inner scope does not matter to the "i"
  // outside the scope, that still has the value 3
}
```

scope of function f1

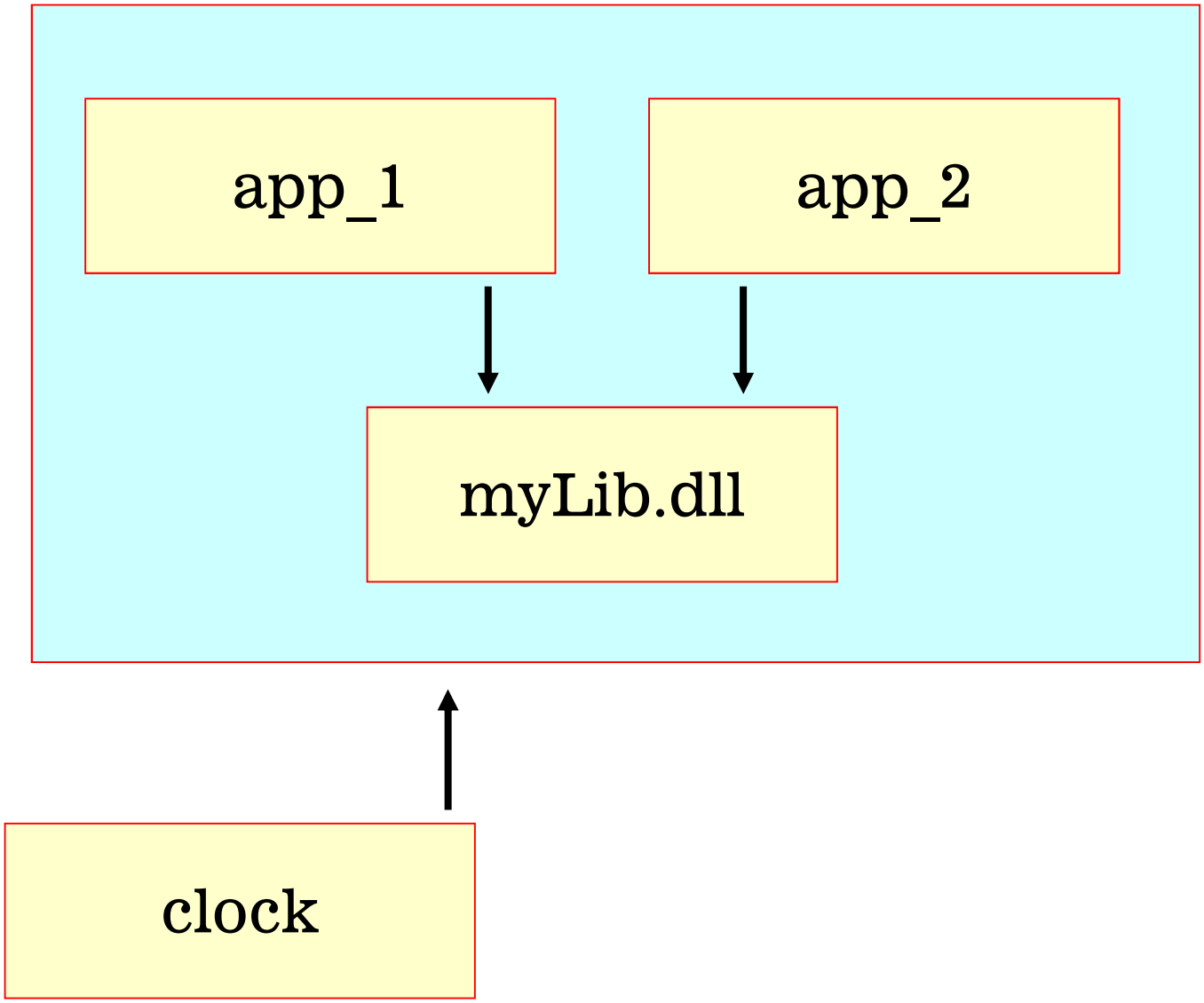
inner scope

www.sbVB.com.br

Problem with global variables

- global variables are the enemy when developing for switched context

Avoid using global variables



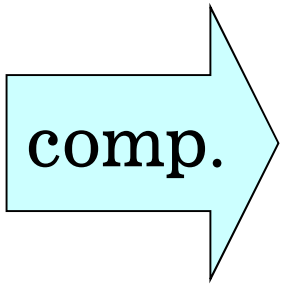
Projects

C/C++ projects

- To produce binary output from source files is a process called “build”, that has 2 phases – compilation and linkage.
- A C/C++ project is the complete set of information that the compiler needs to build (compile and link) the output, from one or more source files (*.c and/or *.cpp).
- Different compilers have different formats to store the project information (even if the source files are the same).
  - That is, the project file from compiler A can not be used with compiler B
- Specially needed if there are more than one source files (*.c or *.cpp).

C/C++ projects (2)

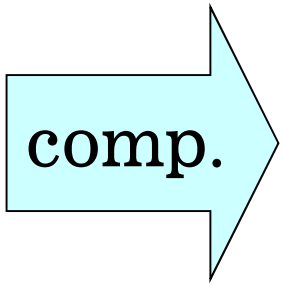
```
// f1.cpp
void f1_1() { /* ... */ }
void f1_2() { /* ... */ }
void f1_3() { /* ... */ }
```



f1.obj

- f1_1
- f1_2
- f1_3

```
// f2.cpp
void f2_1() { /* ... */ }
void f2_2() { /* ... */ }
void f2_3() { /* ... */ }
```

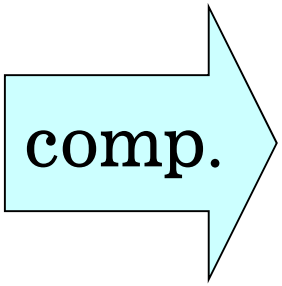


f2.obj

- f2_1
- f2_2
- f2_3

...

```
// fn.cpp
```

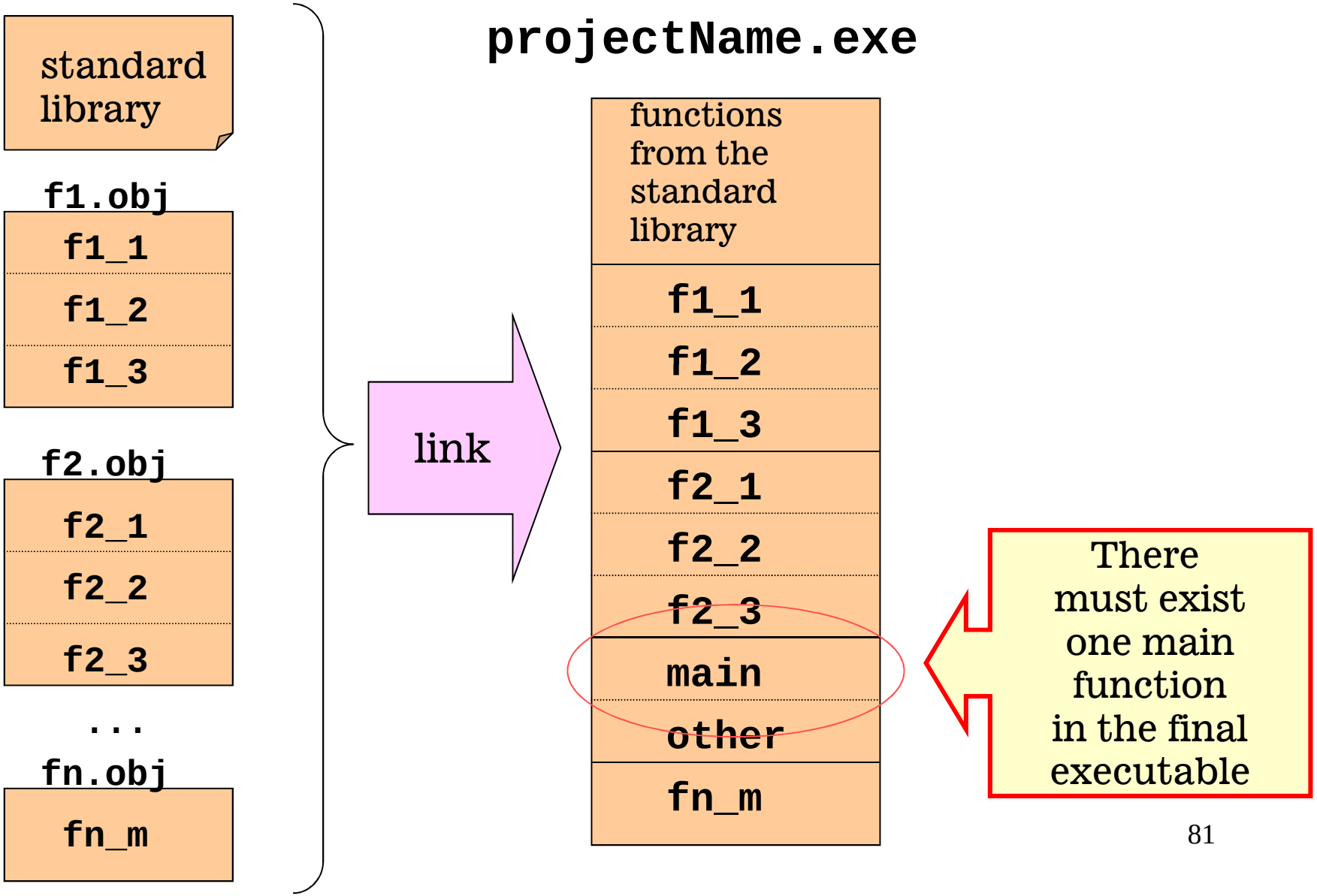


fn.obj

- fn_m

C/C++ projects (3)

www.sbVB.com.br



www.sbVB.com.br

Compilation

- Each source file (*.c or *.cpp) is compiled generating a *.obj (in unix *.o).
- Each source file is compiled as if the other source files in the project didn't exist.
- In the beginning of the compilation of each source file the compiler knows only the keywords and the C/C++ syntax.
- The compiler must succeed to compile the sources in only one step, otherwise a compilation error will be produced.

www.sbVB.com.br

Compilation (2)

- When parsing the source code, the compiler does on of the 2 things:
  - Generate binary code
  - Learn the meaning of identifiers, which could be used to generate code later.
- The compiler can not generate binary code using identifiers that have not been previously declared. Attempt to do so lead to compilation error.
- A source file can belong to more than one project.
- Header files (*.h) shouldn't belong directly to the project.

www.sbVB.com.br

Compilation examples

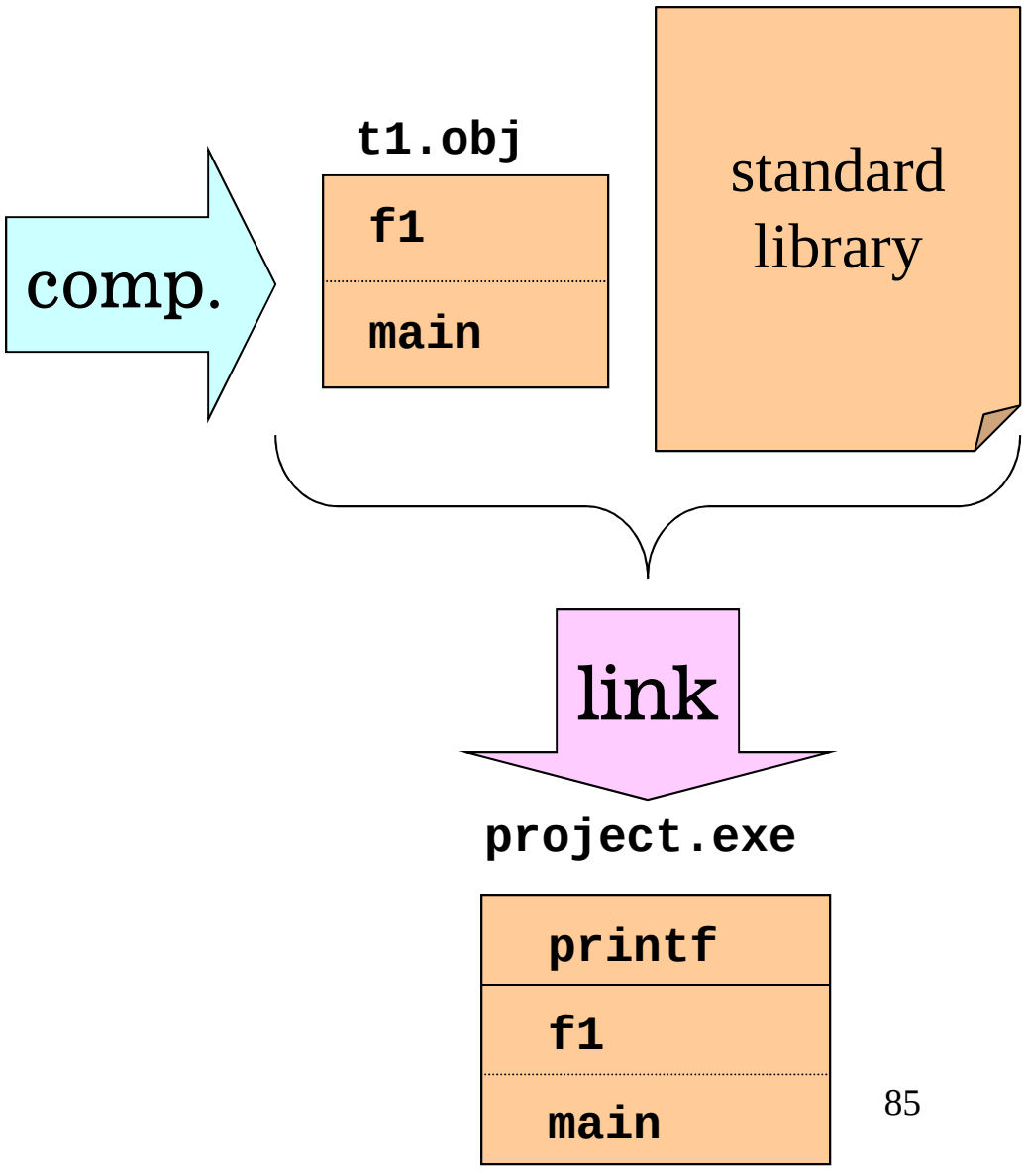
```c
// t1.cpp
#include <stdio.h>
// define (and declare) function “f1”
// (introduce the identificator “f1”)
void f1()
{
  // call function “printf”
  printf("Hello\n");
}

int main()
{
  f1(); // call function “f1”
/* the calling of “f1”
is compatible with prototype
previously declared of “f1” */
 return 0;
}
```

```c
// t2.cpp
#include <stdio.h>
/* declare function “f1”
(introduce the identificator “f1”)
this is the prototype of
function “f1”. This line does not
generate code, but makes the compiler
understand the meaning of “f1” */
void f1();

int main()
{
  f1(); // call function “f1”
  return 0;
}
/* define function “f1”, compatible
with the prototype previously
declared */

void f1()
{
  // call function “printf”
  printf("Hello\n");
}
```

Linking

www.sbVB.com.br

```
// t1.cpp
#include <stdio.h>
void f1() {
  printf("hello\n");
}
int main() {
  f1();
  return 0;
}
```



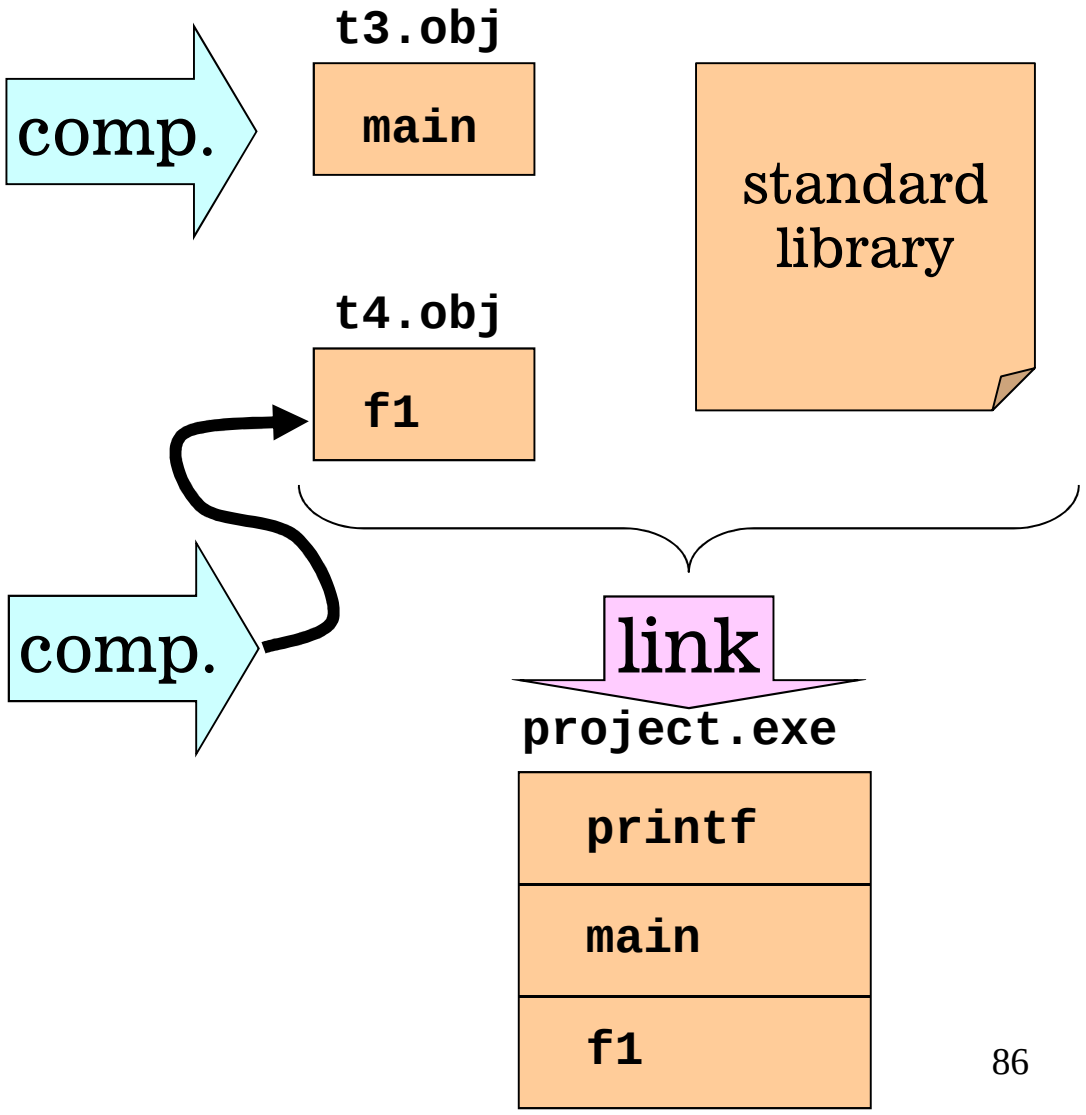

```
c:\>project↵
hello
c:\>
```

Linking (2)

www.sbVB.com.br

```cpp
// t3.cpp
void f1();
int main() {
  f1();
  return 0;
}
```

```cpp
// t4.cpp
#include <stdio.h>
void f1() {
  printf("Hello\n");
}
```



Linking (3)

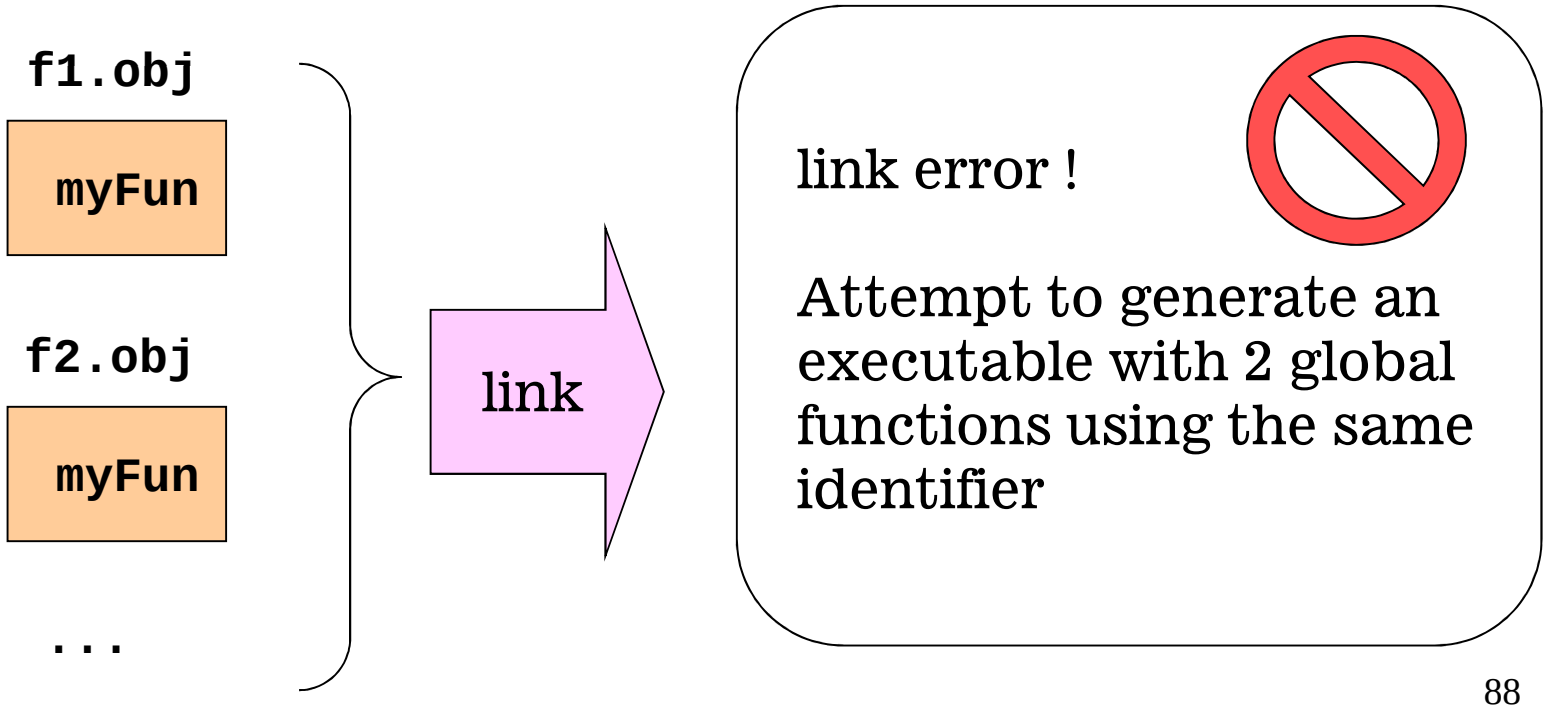
- The link phase starts only after *all* source files in the project have been compiled.
- There are possible error in the link phase
  - There can not have a executable program without a global function “main” (in Windows, “WinMain”).
  - A function declared and used, but not defined

```
// t.cpp

// declaration of
// functions hello and bye
void hello();
void bye();

int main()
{
  hello();
  bye();
  return 0;
}
```

Link error.
Where is the definition of functions hello and bye ?

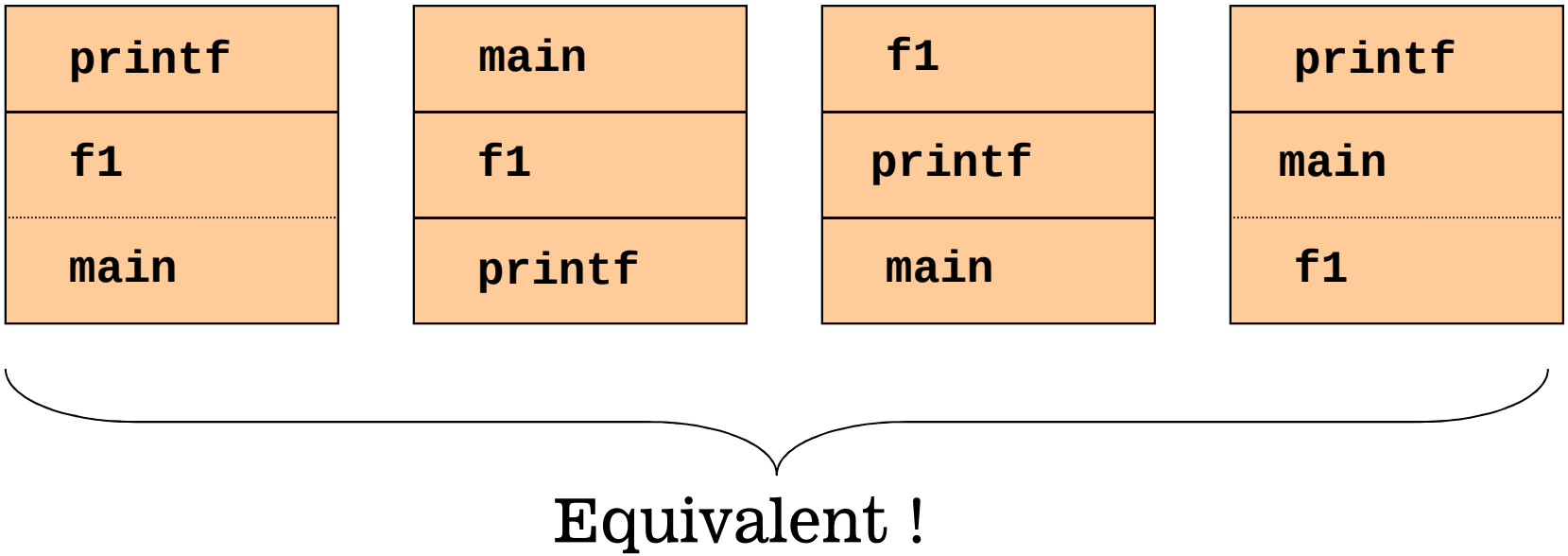
Linking (4)

- Other possible link error
  - In a project, there can not exist more than one entity (functions, objects, etc.) with the same identifier, in the same namespace



Linking (5)

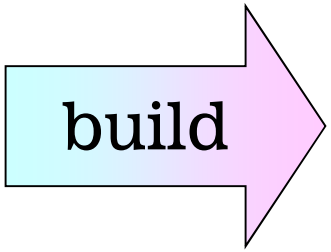
- The order (in the binary output) that the blocks will be placed is irrelevant.

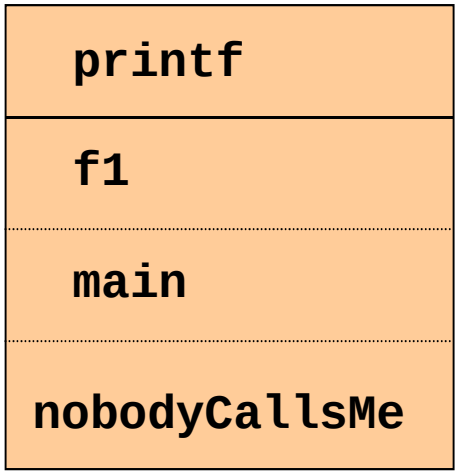


Linking (6)

- Should there be code that is not used, that increases the binary output, but does not affect the program

```
// t5.cpp
#include <stdio.h>
void f1()
{
  printf("Hello");
}
int main()
{
  f1();
  return 0;
}
void nobodyCallsMe()
{
  printf("do something");
}
```





Linking (7)

- The identifiers used (names of functions, names of variables, etc.) do not happen in the binary output. The linker take them off (if the compiler is set to release; if it is set to debug, the identifiers are not taken off).
  - The identifiers must exist in the debug version of a binary output, so that the compiler will use them for the debug process.

www.sbVB.com.br

Header files

- Header files are source files (*.h, *., *.hxx, *.hpp).
- The compiler does not produce object files (*.obj, *.o) from the header files, only directly compilable source files (*.c, *.cpp) make the compiler generate object files.
- The purpose of header files is to be included in directly compilable source files (*.c, *.cpp)
- Typically the header files have function prototypes and other information needed for compilation, but that produce no code.

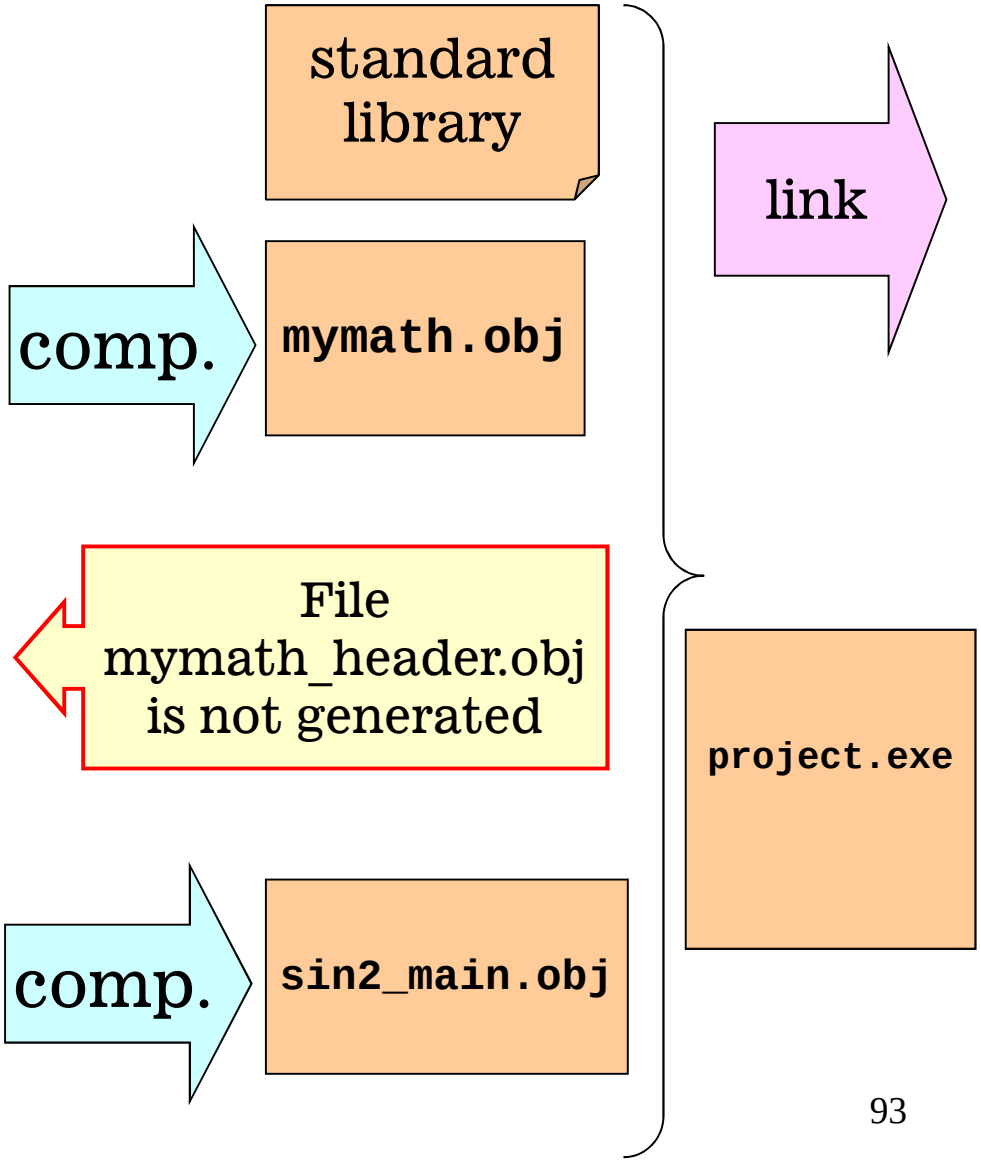
Header files (2)

www.sbVB.com.br

```cpp
// mymath.cpp
#include<math.h>
double sin2(double x)
{
  double ret = sin(x);
  ret *= ret;
  return ret;
}
```

```cpp
// mymath_header.h
double sin2(double x);
```

```cpp
// sin2_main.cpp
#include "mymath_header.h"
int main() {
  double f = sin2(0.4);
  return 0;
}
```



Header files (3)

- The compiler provides many header files, called “standard header files”.
- These files have (among other information) the prototypes of functions of the standard library. E.g.: printf.
- The standard header files are included using the directive “#include” with <, >. E.g.: #include <stdio.h>
- The header files created by the developer are included using the directive “#include” with ", ". E.g.: #include "mymath_header.h"

Header files (4)

- There could be a single standard header. Its name should be “c.h”. But this file would be too big. So, the compiler has many standard headers.
- According to the function of the standard library you want to use, a related standard header should be included for the prototype. To know which header is to be included for which function is not important to memorize, because there are helps available for that. E.g.: the function “printf” needs header stdio.h.

www.sbVB.com.br

Libraries

- A library can be understood as a group of object files (*.obj or *.o), that are packed in a single file to ease the handling.
- A library is a binary block that is not an executable. It is useful to be included in a project that generates an executable (or some other library).
- The only difference from the standard library and some other library developed by someone is that the standard library is automatically included to any project.

www.sbVB.com.br

Elements of a Library

Element	Example in standard library
Global variables (global objects)	cin, cout
Global functions	sin, cos, atoi
Global classes	ifstream, ofstream

www.sbVB.com.br

Header ≠ Library

- Header
  - Is a text file (*.h, *.hpp, *.hxx, *.), produced by a text editor (such as notepad or vi).
  - Is used to be include to a directly compilable source files (*.c, *.cpp) (E.g.: #include<myheader.h>).
  - To include or not to include a header file to a directly compilable source file (*.c, *.cpp) influences the compilation of that source file.
  - A project defines a “header path”
- Library
  - Is a binary file (*.a, *.lib, *.dll, *.so), produced as output of a build (compilation followed by a link).
  - Is used to be included in a project (remark that each compiler has a different interface to allow inclusion of files to a project).
  - To include or not to include a library to a project influences the link phase of a build.
  - A project defines a “library path”

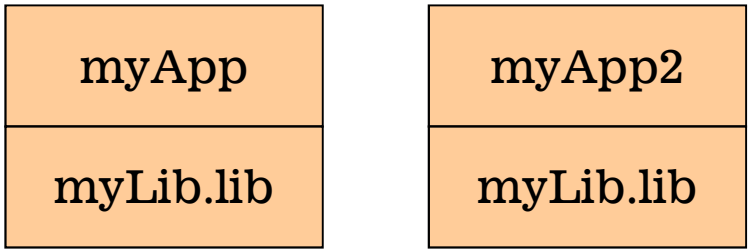
www.sbVB.com.br

The 3 ways to link libraries

- There are 3 ways to link a library to a project (an application, another library, etc.)
  1. *static link*: (*.lib in Win, *.a in unix).
  2. *implicit dynamic link:* links ***at time*** the application is being launched by the operating system (*.dll in Win, *.so in unix).
     - **No** extra code is required to use the entities of the library. For the developer, the use is like the static link.
  3. *explicit dynamic link*: links ***after*** the application was launched by the operating system (*.dll in Win, *.so in unix).
     - Extra code is required to use the entities of the library.

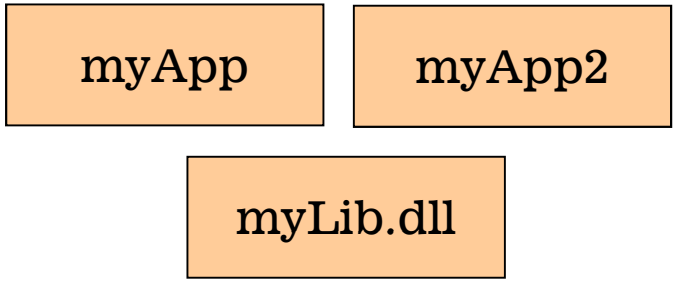
Static link × implicit dynamic link

static link



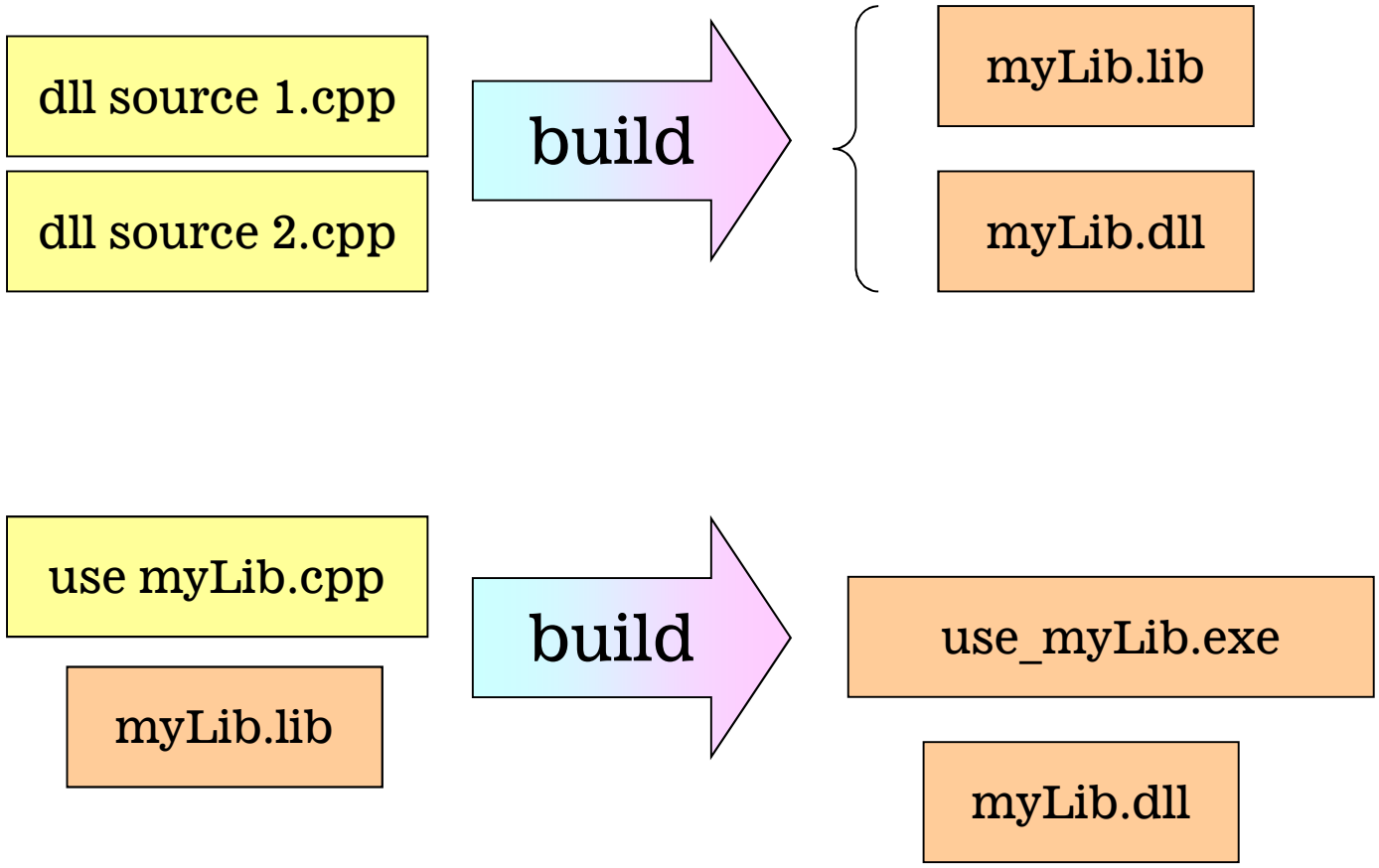
- The executable has all code needed (easier to install).
- Executable is bigger (because the code of library is added to the executable).
- For *n* executable running, there are *n* copies of the library in the memory.

implicit dynamic link



- The executable can only run if the dll is in the path (harder to install).
- Executable is smaller (because the code of library is not added to the executable)
- For *n* executable running, there is only 1 copy of the library in the memory.

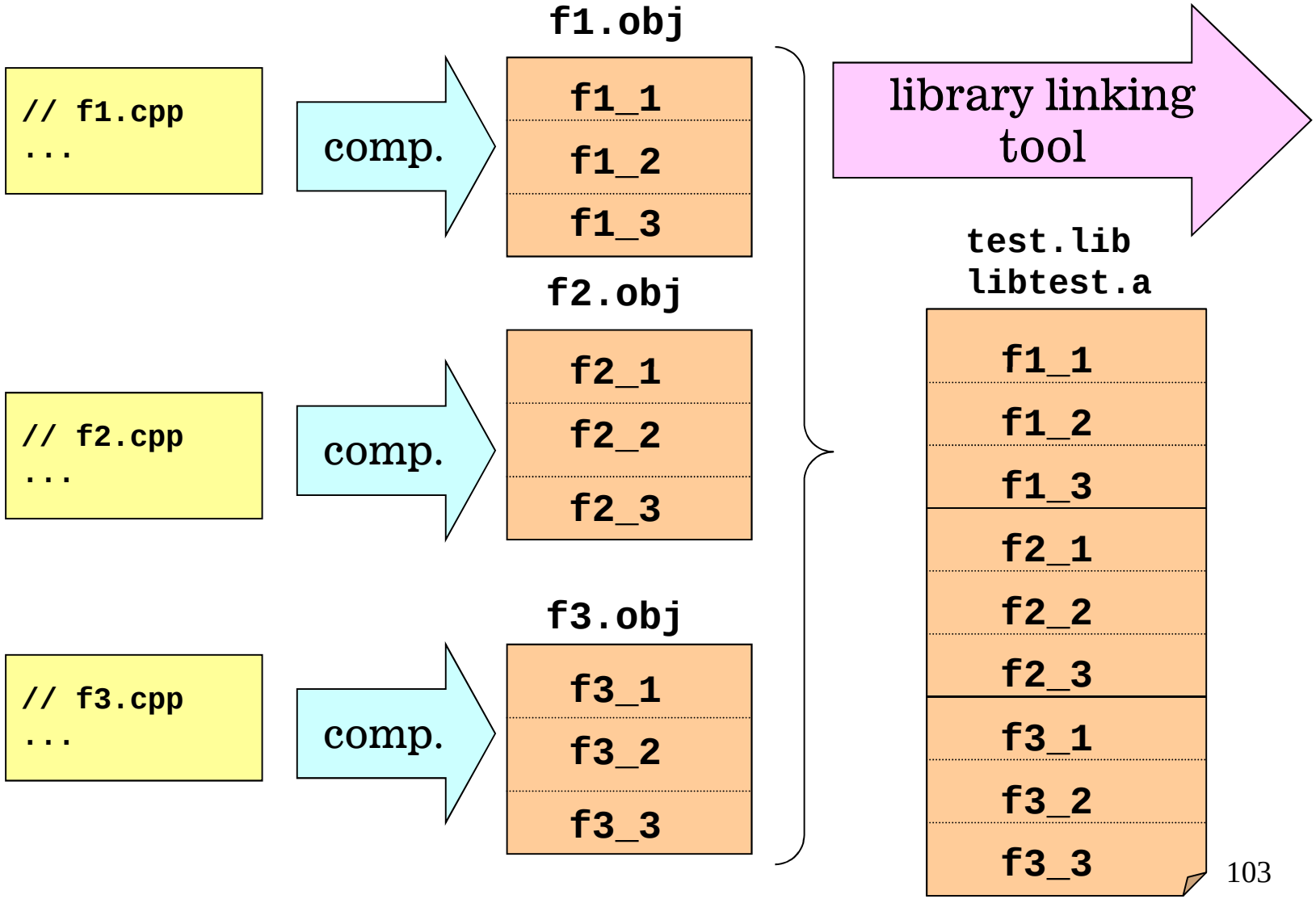
Generating dll with VC



Dynamic libraries

- *.dll in Windows and *.so in Unix.
- The code of the library stays outside the application file. If more than one application uses the same dynamic library, only one copy of it goes to memory.
  - That is good for developing “famous libraries”, e.g.: wxWidgets, Xerces, VBLib, SQLAPI, etc.
- Maintainability.
- If using explicit link, change in the behavior without restarting the program.
  - Applying a skin to a program
  - Installing a printer and using with your old text editor
  - Plugins in general

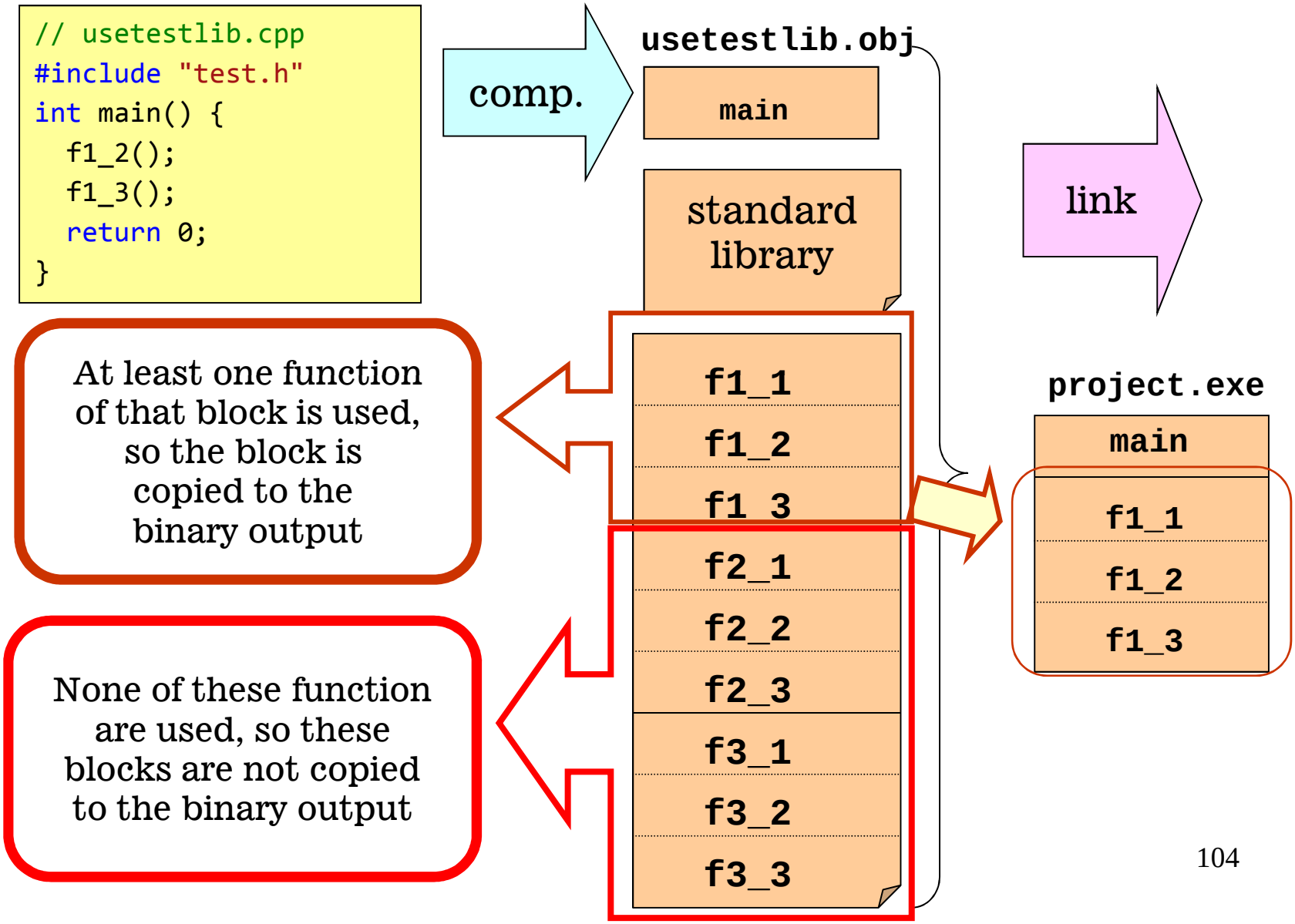
Building a static library



Using a static library

www.sbVB.com.br

```
// usetestlib.cpp
#include "test.h"
int main() {
  f1_2();
  f1_3();
  return 0;
}
```



Example library: plus_n

```cpp
// plus_1_file.cpp
int plus_1 (int in)
{ return in + 1; }
```

```cpp
// plus_2_file.cpp
int plus_2 (int in)
{ return in + 2; }
```

```cpp
// plus_3_file.cpp
int plus_3 (int in)
{ return in + 3; }
```

```cpp
// plus_4_file.cpp
int plus_4 (int in)
{ return in + 4; }
```

```cpp
// plus_5_file.cpp
int plus_5 (int in)
{ return in + 5; }
```

plus_n.lib
libplus_n.a

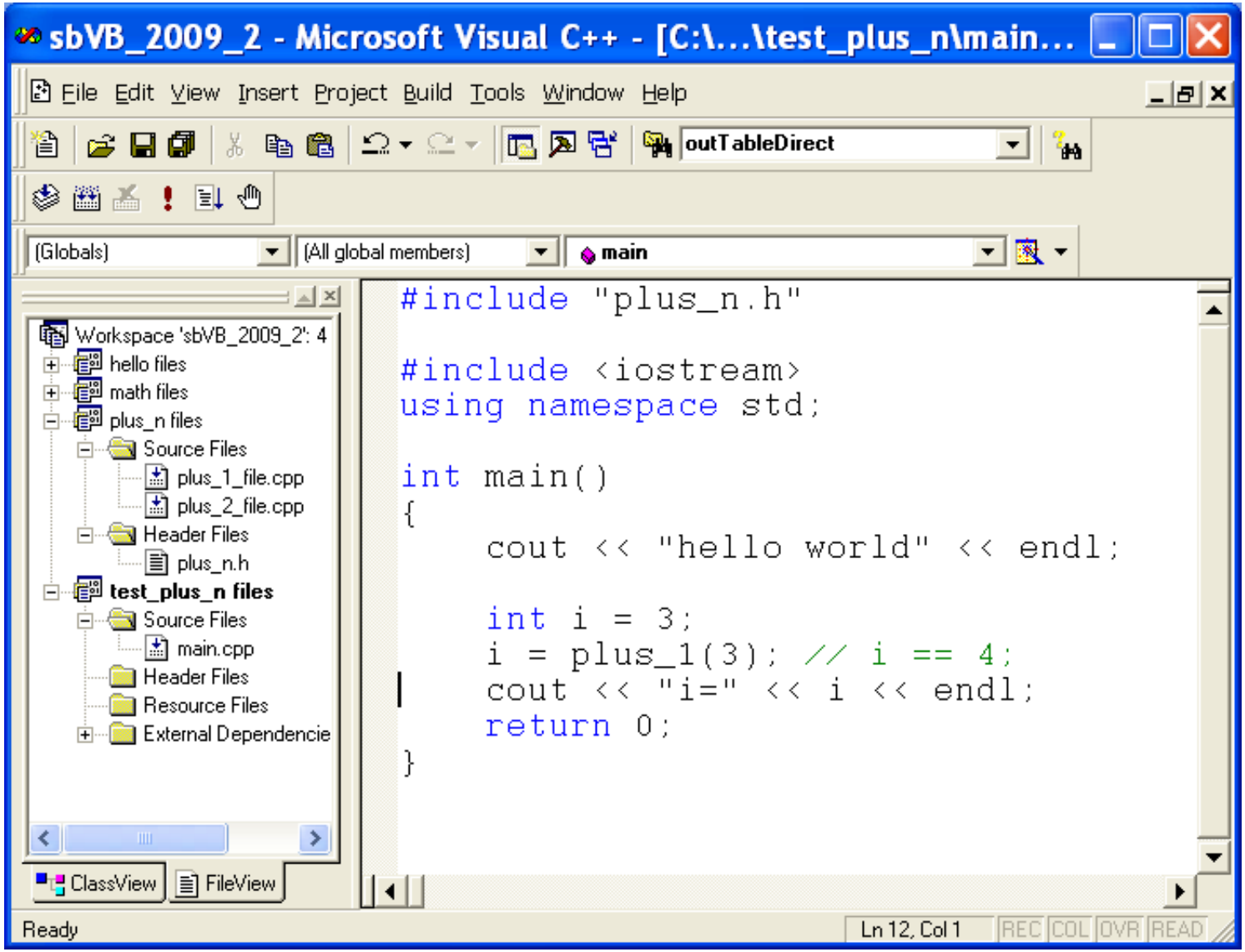
```cpp
// plus_n.h
int plus_1 (int in);
int plus_2 (int in);
int plus_3 (int in);
int plus_4 (int in);
int plus_5 (int in);
```

```
k = 10
k+3 = 13
```

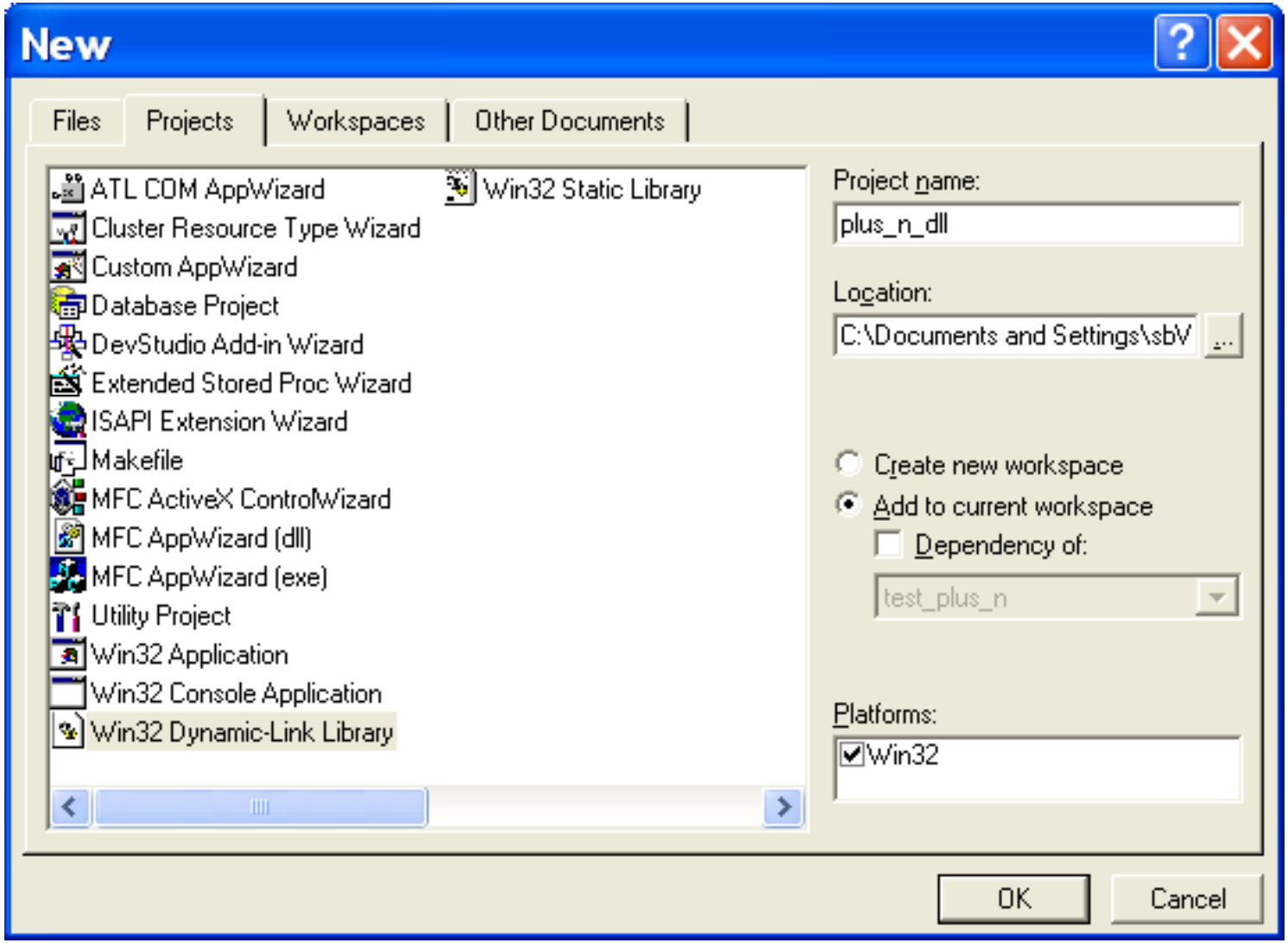
```cpp
// test_plus_n.cpp
#include <iostream>
using namespace std;
#include "plus_n.h"
int main() {
  int k = 10;
  cout << "k = " << k << endl;
  cout << "k+3 = " << plus_3(k) << endl;
  return 0;
}
```

www.sbVB.com.br

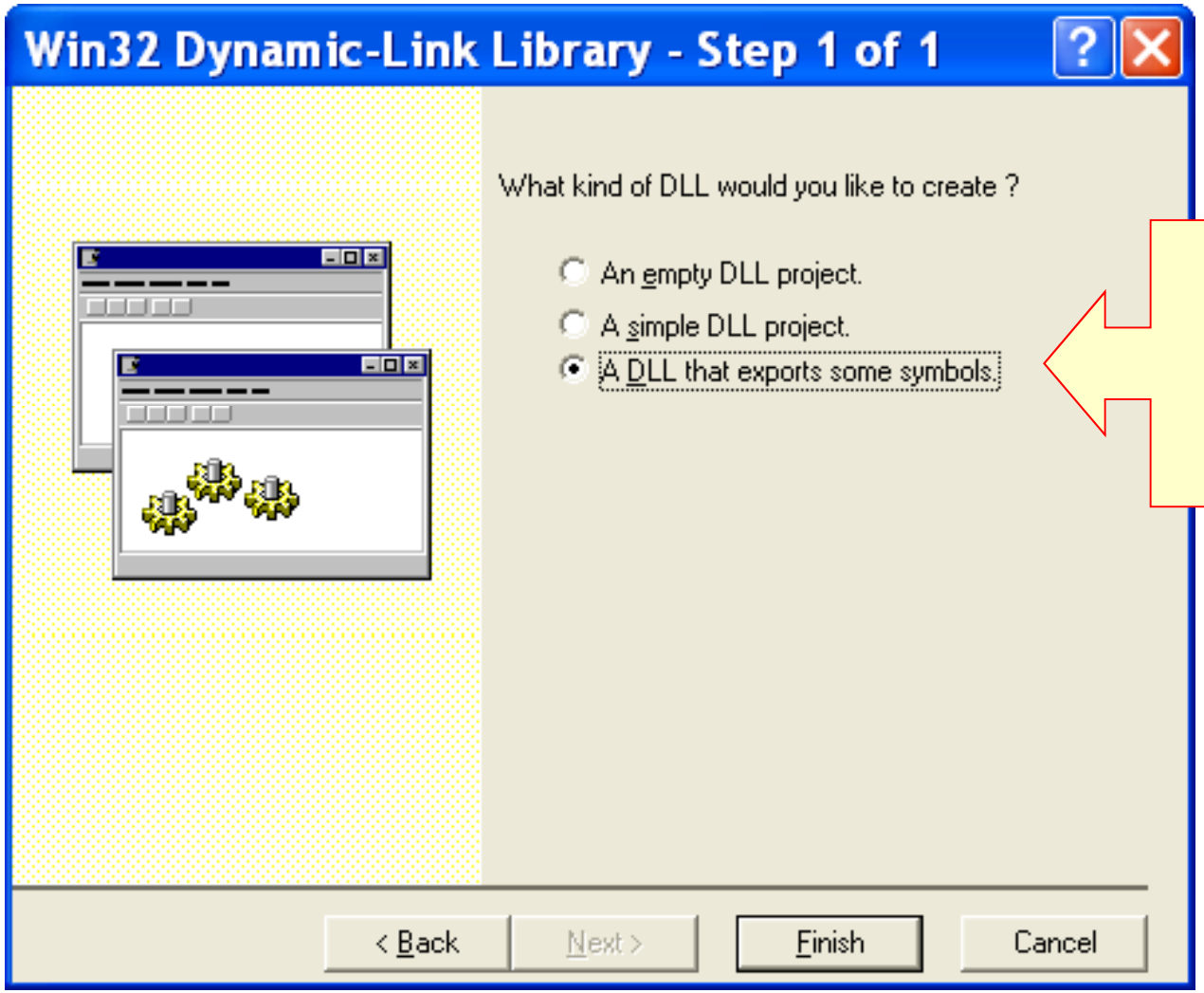
plus_n static with VC 6



plus_n dynamic with VC 6



plus_n dynamic with VC 6 (2)



choose this option !

www.sbVB.com.br

plus_n dynamic with VC 6 (3)

```
// plus_n_dll.h
#ifdef PLUS_N_DLL_EXPORTS
#define PLUS_N_DLL_API __declspec(dllexport)
#else
#define PLUS_N_DLL_API __declspec(dllimport)
#endif

// This class is exported from the plus_n_dll.dll
class PLUS_N_DLL_API CPlus_n_dll {
public:
	CPlus_n_dll(void);
	// TODO: add your methods here.
};

extern PLUS_N_DLL_API int nPlus_n_dll;


PLUS_N_DLL_API int fnPlus_n_dll(void);
```

code to control export of elements

exported global class

exported global object

exported global function

plus_n dynamic with VC (4)

```
// plus_n_dll.cpp
#include "stdafx.h"
#include "plus_n_dll.h"
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                     )
{
    switch (ul_reason_for_call)
       {
              case DLL_PROCESS_ATTACH:
              case DLL_THREAD_ATTACH:
              case DLL_THREAD_DETACH:
              case DLL_PROCESS_DETACH:
                     break;
    }
    return TRUE;
}
```

code to control dll

www.sbVB.com.br

plus_n dynamic with VC (5)

```
// This is an example of an exported variable
PLUS_N_DLL_API int nPlus_n_dll=0;

// This is an example of an exported function.
PLUS_N_DLL_API int fnPlus_n_dll(void)
{
      return 42;
}

// This is the constructor of a class that has been exported.
// see plus_n_dll.h for the class definition
CPlus_n_dll::CPlus_n_dll()
{
      return;
}
```

exported global object

exported global function

exported global class

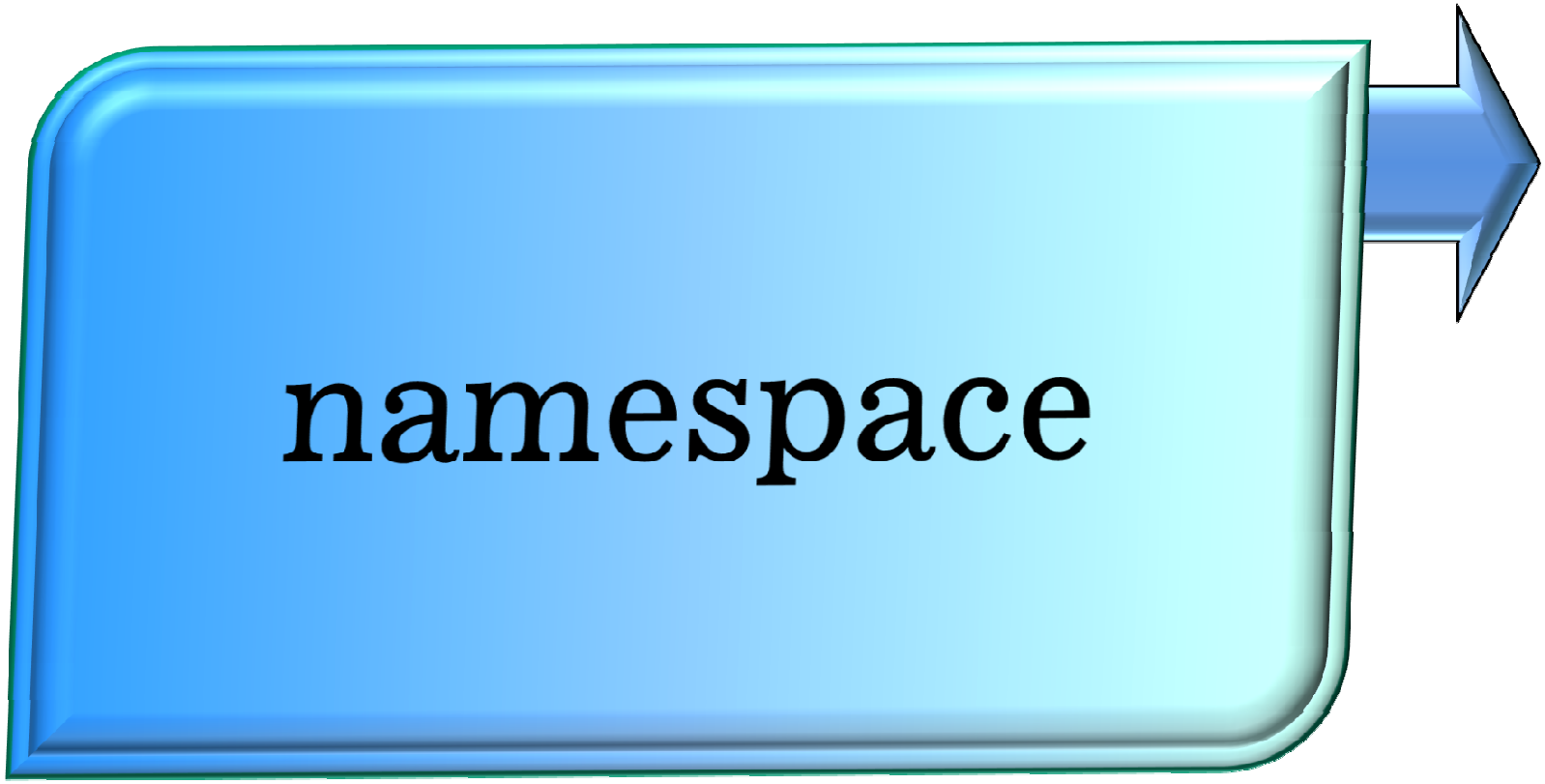
Building and testing plus_n in linux/g++ (static link)

```
#mymake.sh

##########################################
# static link version
##########################################

# compile source files
g++ -c plus_1_file.cpp plus_2_file.cpp plus_3_file.cpp plus_4_file.cpp plus_5_file.cpp

# link and produce library
ar r libplus_n.a plus_1_file.o plus_2_file.o plus_3_file.o plus_4_file.o plus_5_file.o

# compile and link test of library
g++ test_plus_n.cpp -L. -lplus_n -ot_plus_n

# run test
./t_plus_n
```

Building and testing plus_n in linux/g++ (dynamic link)

```
#mymake.sh

##########################################
# dynamic link version
##########################################

# compile and link dynamic library
g++ -shared plus_1_file.cpp plus_2_file.cpp plus_3_file.cpp plus_4_file.cpp plus_5_file.cpp
  -o plus_n.so

# compile and link test of library
# g++ test_plus_n.cpp -ldl ./plus_n.so -ot_plus_n_dyn
g++ test_plus_n.cpp ./plus_n.so -ot_plus_n_dyn


# run test
./t_plus_n_dyn
```

Namespace



www.sbVB.com.br

Namespace

- Feature of the C++ language to isolate names.
- This feature is not an “object oriented” feature.
- C++ has a “before namespace era”, while Java was invented already with this feature, that in Java is known as “package”.

Namespace (2)

```
#include <iostream.h>
namespace OneNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of oneNamespace"
      << endl;
  }
}
namespace OtherNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of otherNamespace" << endl;
  }
}
int main()
{
  ::OneNamespace::errorHandler();
  ::OtherNamespace::errorHandler();
  return 0;
}
```

```
This is the error handler of oneNamespace
This is the error handler of otherNamespace
```

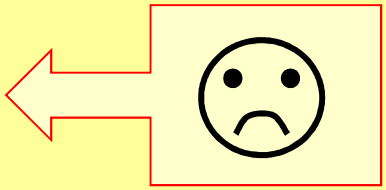
Namespace (3)

```
This is the error handler of oneNamespace
This is the error handler of otherNamespace
```

```
#include <iostream.h>
namespace OneNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of oneNamespace" << endl;
  }
}
namespace OtherNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of otherNamespace" << endl;
  }
}
int main()
{
  using namespace OneNamespace;
  errorHandler();  // uses namespace oneNamespace by default
  ::OtherNamespace::errorHandler();
  return 0;
}
```

Namespace (4)

```
#include <iostream.h>
namespace OneNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of oneNamespace" << endl;
  }
}
namespace OtherNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of otherNamespace" << endl;
  }
}
int main()
{
  using namespace OneNamespace;
  using namespace OtherNamespace;
  errorHandler();  // error C2668: 'errorHandler' :
    // ambiguous call to overloaded function
  return 0;
}
```



www.sbVB.com.br

Namespace (5)

```
#include <iostream.h>
namespace OneNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of oneNamespace" << endl;
  }
}
namespace OtherNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of otherNamespace" << endl;
  }
}
int main()
{
  using namespace OneNamespace;
  using namespace OtherNamespace;
  ::OneNamespace::errorHandler();
  ::OtherNamespace::errorHandler();
  return 0;
}
```

Namespace is open

```
#include <iostream.h>
namespace OneNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of oneNamespace" << endl;
  }
}
namespace OtherNamespace
{
  void errorHandler()
  {
    cout << "This is the error handler of otherNamespace" << endl;
  }
}
namespace OneNamespace // continuation of oneNamespace
{
  void otherFun()
  {
    cout << "otherFun of oneNamespace" << endl;
  }
}
```

www.sbVB.com.br

Namespace can be nested

```
namespace br {
  namespace com {
    namespace sbVB {
      typedef int MyInt;

      // declaring a function inside the
      // nested namespace
      MyInt fun();

      // defining a function inside
      // the nested namespace
      int plus_1(int in) {return in+1;}
    }
  }
}

// defining the function outside the namespace
::br::com::sbVB::MyInt br::com::sbVB::fun() {return 3;}


int main()
{
  int k = ::br::com::sbVB::plus_1(3); // k=4
  return 0;
}
```

The nested namespace should represent the url of the author (person or entity)

www.sbVB.com.br

The C++ standard library uses the namespace std

- Use standard headers without the ".h" extension (when they exist).
- Many standard headers were rewritten using namespace std.
- Do not mix old-style standard headers (with .h extension) with new style headers (that use std namespace, without .h extension).
- Don't add global "using namespace xyz" (e.g.: using namespace std) statements in user header files, because this kills the namespace isolation.

www.sbVB.com.br

The C++ standard library uses the namespace std (2)

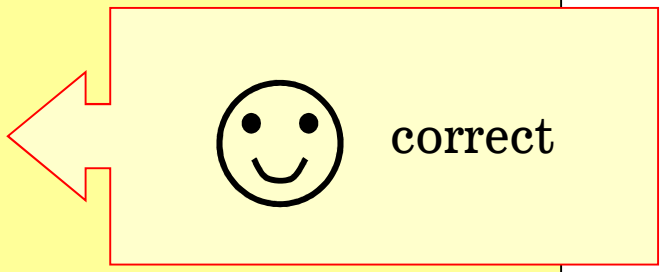
```
#include <iostream>
#include <string>
int main()
{
  ::std::string s = "abc";
  s += "def";
  ::std::cout << s << ::std::endl;
  return 0;
}
```

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string s = "abc";
  s += "def";
  cout << s << endl;
  return 0;
}
```

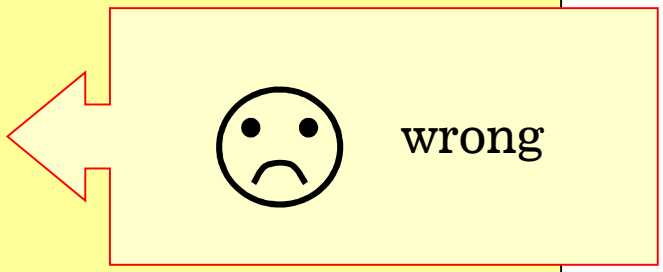
```
abcdef
```

Don't mix styles (with and without namespace)

```
// correct, namespace style only
#include<iostream>
#include<string>
using namespace std;
// ...
```




```
// wrong, mixing namespace style (without .h) and
// non namespace style (with .h)
#include<iostream>
#include<string.h>
using namespace std;
// ...
```



New headers for namespace std

www.sbVB.com.br

wrong

iostream.h
fstream.h
stdio.h
math.h

```
// using old style
// C functions
#include <stdio.h>
int main()
{
  int i=3;
  printf("i=%d\n",i);
  return 0;
}
```

iostream
fstream
cstdio
cmath

correct

```
// using C functions with
// new style C++
#include <cstdio>
int main()
{
  int i=3;
  ::printf("i=%d\n",i);
  ::std::printf("i=%d\n",i);
  return 0;
}
```

Testing software

www.sbVB.com.br

Software engineering

- Software engineering is more than just “coding,” it is applying “a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software”

  (IEEE 1990)

www.sbVB.com.br

Why to test software?

- In normal conditions, the hardware rarely fails. So, why should software be tested?
  - Because there can exist bugs (defects in the software).
- There's a lot of theory on software engineering that deals on how to analyze, specify, develop, test, deploy, control versions, estimate costs, etc. The computer language is one of the aspects of the software engineering.
  - Tests are divided in automatic tests and manual tests.
- Part of the specification work can be done through automatic tests.
  - This is test-driven software development
- If (when) a bug happens, it is very useful to have automatic tests to help track down the bug.

www.sbVB.com.br

Why to test software?(2)

- The automatic test of a software is itself a software. Therefore the automatic tests should be tested as well.
  - One should produce a intentional bug, to see if the automatic test will detect it.
  - If an automatic test is not itself tested, a buggy software can be approved by the automatic test and let the bug go ahead undetected.
- To do a lot of automatic tests is good. However there's the issue that to develop cost resources. So there's a "trade-off" defined by the management that defines how much of automatic tests will be developed.
- The existence of automatic tests represent a level of excellence in the software development process.
  - The automatic tests represent assurance that the libraries and other components will not degrade through time, as the components keep being maintained (changed), and new versions are being released.

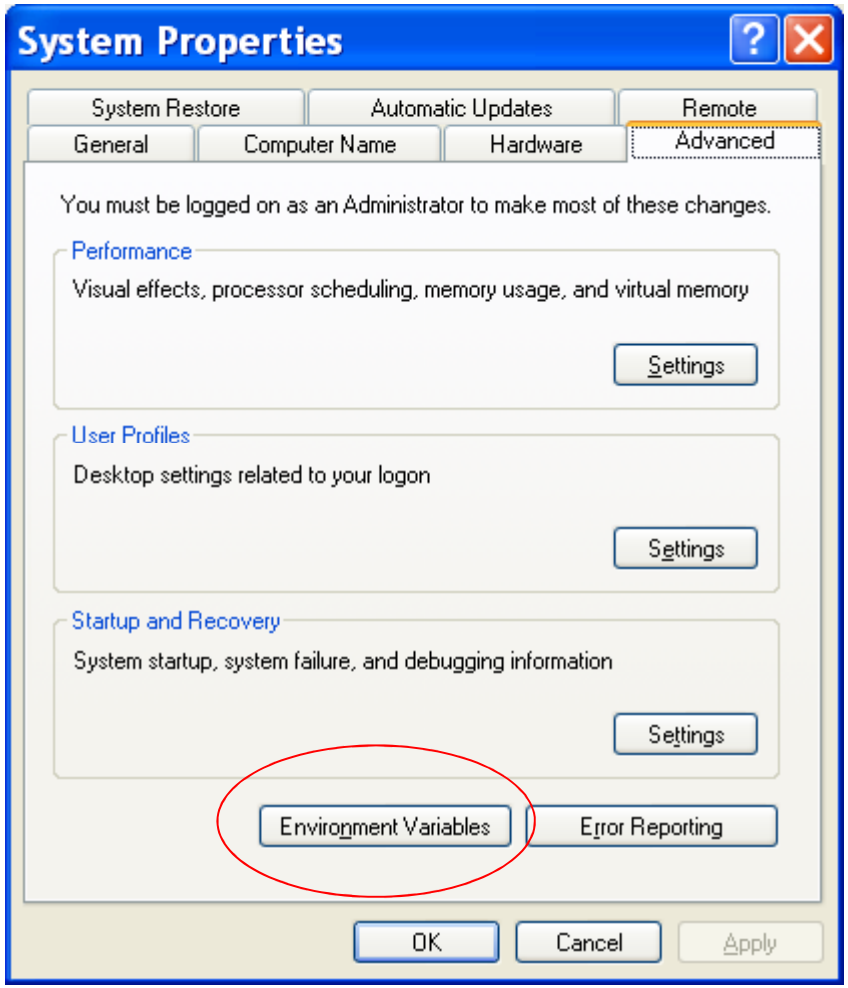
www.sbVB.com.br

Using library VBLib

- download VBLib from google code
  - http://code.google.com/p/sbvblibs/
- If using Windows, download tortoise svn client.
- Use svn client and point it to
  - http://sbvblibs.googlecode.com/svn/trunk/
- Create a folder do receive sbvblibs;
  - for example ~My Documents_googleCode\sbvblibs
- Create an environment variable named "sbvb_home" and assign it to the folder you used to receive sbvblibs
- Add to path the folder below
  - ~My Documents_googleCode\sbvblibs\AddMeToPath
- Rebuild VBLib using your C++ compiler
- Create a "HelloVBLib" project to test usage of VBLib

Windows system path

www.sbVB.com.br

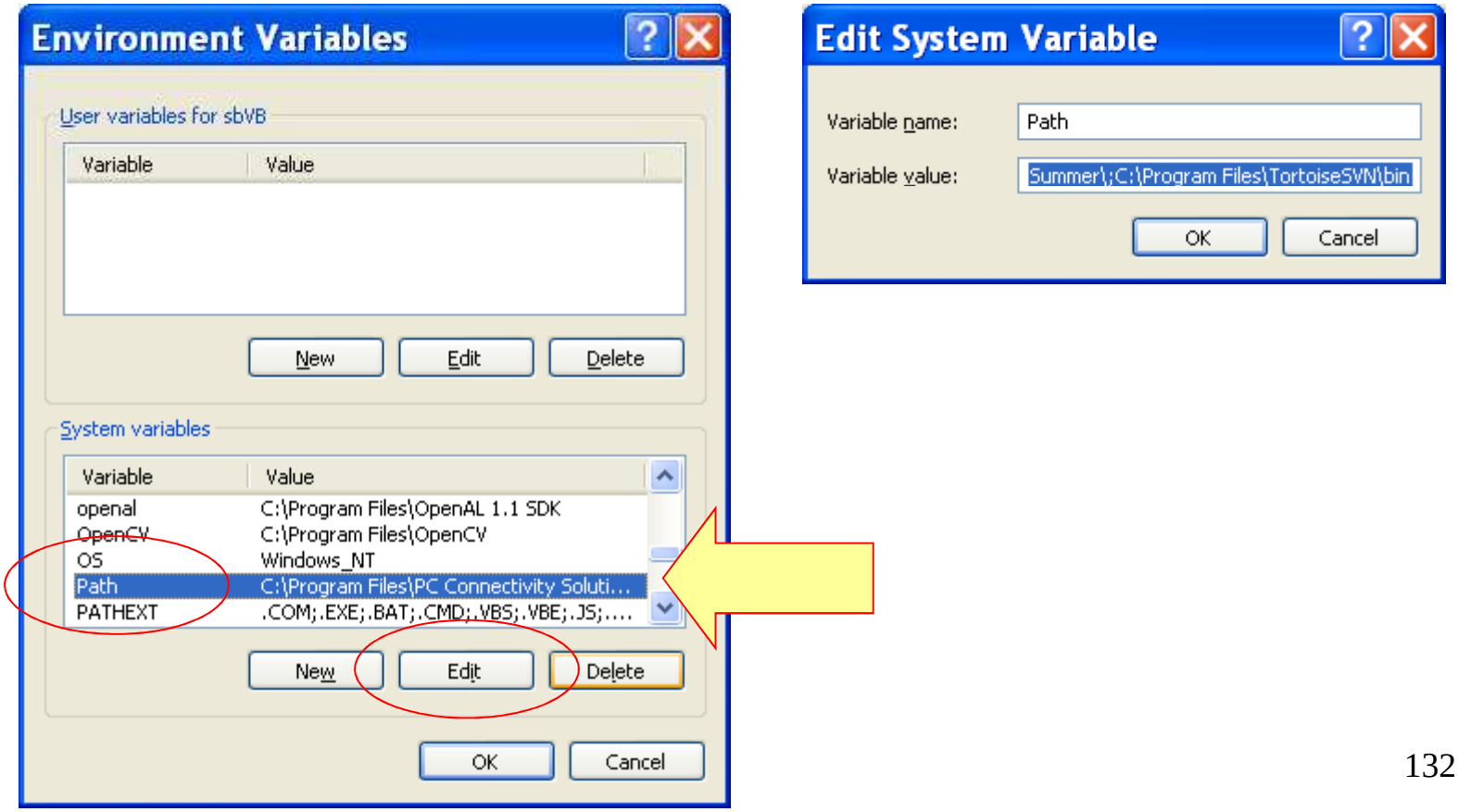


- My Computer-> properties, “advanced” tab, Environment variables button

Windows system path

www.sbVB.com.br

- edit “path” environment variable, and add folder below to it
- ~My Documents_googleCode\sbvblibs\AddMeToPath



Test VBLib

```
#include "VBLib/VBLib.h"
using namespace std;
using namespace br::com::sbVB::VBLib;
int main()
{
  VBString s = "hello VBLib";
  cout << s << endl;
  return 0;
}
```

```
hello VBLib
```

Test VBLib (2)

- Don’t forget to
  - add header path
    $(sbvb_home)/VBLib_6_2/include
  - add VBLib.lib to project
  - add library path
    $(sbvb_home)/VBLib_6_2/lib/vc6_dll
  - add folder below to sytem path
    - ~My Documents_googleCode\sbvblibs\AddMeToPath

www.sbVB.com.br

An automatic test, when succeeds, does nothing

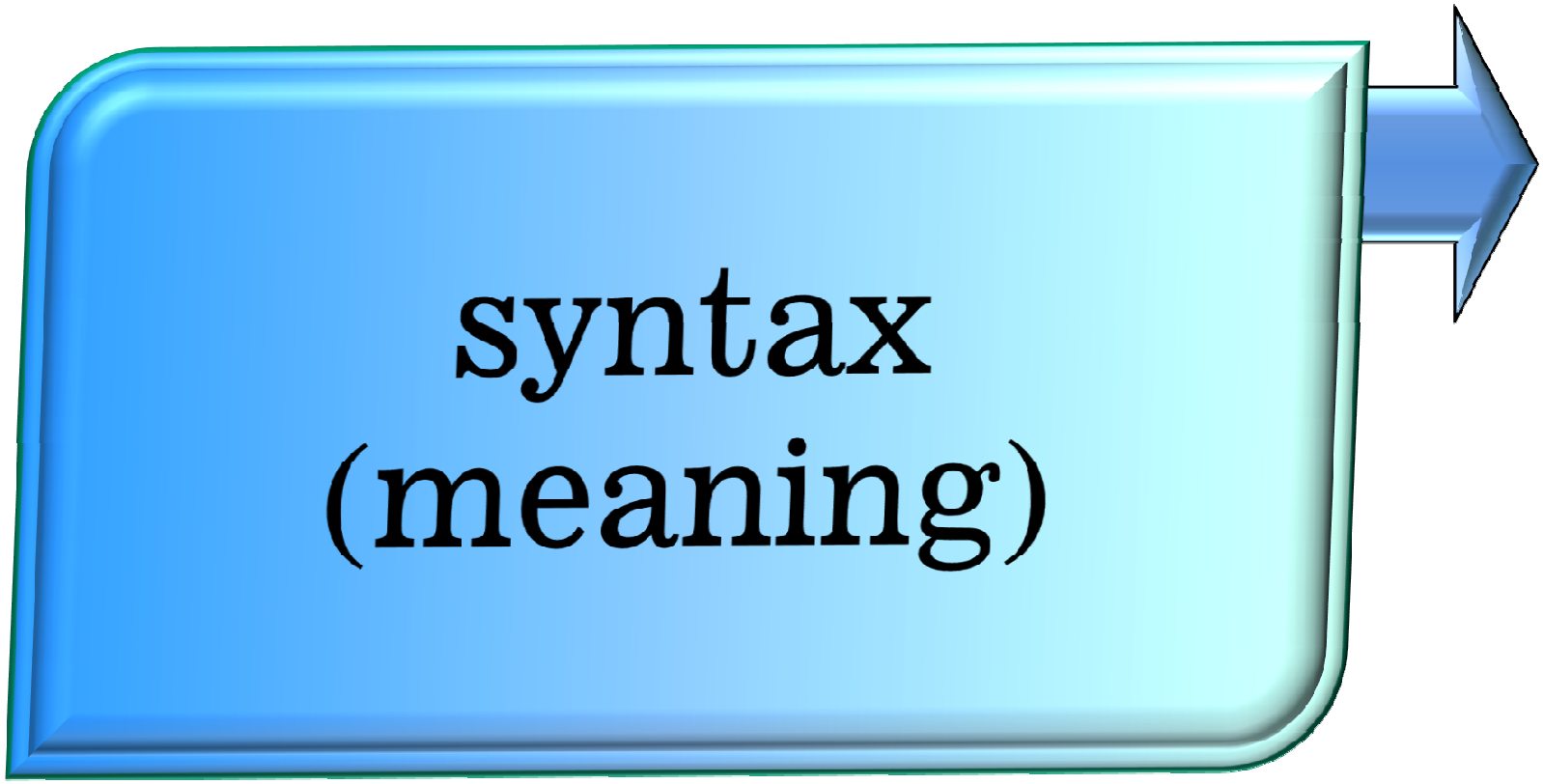
```
#include "VBLib/VBLib.h"
using namespace std;
using namespace br::com::sbVB::VBLib;
int main()
{
  VBAssert(true);
  VBAssert(false);
  VBAssert(false,"message");
  return 0;
}
```

```
VBAssert assertion failed:
VBAssert assertion failed:message
```

```
#include "VBLib/VBLib.h"
using namespace std;
using namespace br::com::sbVB::VBLib;

int plus_2(int i)
{
  return i + 2;
}

int main()
{
  VBAssert(plus_2(3)==5,"error in plus_2");
  VBAssert(plus_2(-3)==-1,"error in plus_2");
  return 0;
}
```

This program does nothing, that is, it passes the test. Everything inside the asserts must be true for the test to succeed.

C++ syntax



typedef

- typedef creates (defines) a user type.
- One usefulness of typedef is to let you not use standard types directly, thus improving maintainability

```
typedef int MyInt;

int main()
{
  MyInt i = 3;
  return 0;
}
```

Conditional execution and block statement

```
if (<bool>)
  <statement>
else              // optional
  <statement>; // optional
```

```
#include <iostream>
using namespace std;
int main()
{
  float a, b, c;
  a = 2;
  b = 1.5;
  if (b == 0)
    cout << "c is undefined" << endl;
  else
  {
    c = a / b;
    cout << "c = " << c << endl;
  }
  return 0;
}
```

```
c = 1.33333
```

block statement

Loop control (while)

- Loops
  - while
  - do .. while
  - for

```
#include <iostream>
using namespace std;
int main()
{
  int i = 0;
  while (i < 5)
  {
    cout << "i=" << i << endl;
    i++; // i = i + 1;
  }
  return 0;
}
```

```
while (<bool>)
  <statement>;
```

```
i=0
i=1
i=2
i=3
i=4
```

www.sbVB.com.br

Loop control (do.. while)

- Loops
  - while
  - do .. while
  - for

```
#include <iostream>
using namespace std;
int main()
{
  int i = 0;
  do
  {
    cout << "i=" << i << endl;
    i++; // i = i + 1;
  } while (i < 5);
  return 0;
}
```

```
do
  <statement>;
while (<bool>);
```

```
i=0
i=1
i=2
i=3
i=4
```

Loop control (for)

www.sbVB.com.br

- Loops
  - while
  - do .. while
  - for

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

<A> <B> <C> <D>

```
i=0
i=1
i=2
i=3
i=4
```

```
for (<A> ; <B> ; <C>)
  <D>;
```

```
{
  <A>;
  while (<B>)
  {
    <D>;
    <C>;
  }
}
```

Visual C++ 6.0 implements the for in a non standard way. Newer versions fixed this

Loop control (for) (2)

```
#include<iostream>
using namespace std;
int main ()
{
  int i;
  int i; // 'int i' : redefinition

  return 0;
}
```

Same identifier.
Compile error.

```
#include<iostream>
using namespace std;
int main ()
{
  for (int i=0; i<5 ; i++)
    cout << "i=" << i << endl;
  for (int i=0; i<5 ; i++)
    cout << "i=" << i << endl;

  return 0;
}
```

Whether compile or not depends on compiler.

VC6 won't compile;
modern compilers do compile.

Loop control: break

```
#include <iostream>
using namespace std;
int main()
{
  for (int i = 0; i < 2; i++)
  {
    for (int j = 5; j>0; j--)
    {
      if (j == 3)
        break;
      cout << "i = " << i << ", j = " << j << endl;
    }
  }
  return 0;
}
```

The break command produces interruption of the inner loop, going to the outer loop (if exist)

```
i = 0, j = 5
i = 0, j = 4
i = 1, j = 5
i = 1, j = 4
```

Loop control: continue

```
#include <iostream>
using namespace std;
int main()
{
  for (int i = 0; i < 2; i++)
  {
    for (int j = 5; j>0; j--)
    {
      if (j == 3)
        continue;
      cout << "i = " << i << ", j = " << j << endl;
    }
  }
  return 0;
}
```

The continue command produces a goto the end of current loop

```
i = 0, j = 5
i = 0, j = 4
i = 0, j = 2
i = 0, j = 1
i = 1, j = 5
i = 1, j = 4
i = 1, j = 2
i = 1, j = 1
```

www.sbVB.com.br

Flow control: switch–case

```
switch (<expr>)
{
  case <item>:
    <statements>;
  case <item>:
    <statements>;
  default:
    <statements>;
}
```

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

int fun(int i)
{
  int ret;
  switch (i)
  {
    case 1:
      ret = 10;
      break;
    case 2:
      ret = 20;
      break;
    default:
      ret = 30;
      break;
  }
  return ret;
}

int main()
{
  VBAssert(fun(1)==10);
  VBAssert(fun(2)==20);
  VBAssert(fun(3)==30);
  VBAssert(fun(4)==30);
  return 0;
}
```

www.sbVB.com.br

Flow control: switch–case (2)

```
#include "VBLib/VBLib.h"
using namespace std;
using namespace br::com::sbVB::VBLib;

VBString fun(int grade) {
  VBString ret;
  switch (grade)
  {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
      ret = "terrible";
      break;
    case 5:
    case 6:
    case 7:
    case 8:
      ret = "good";
      break;
    case 10:
      ret = "outstanding & ";
    case 9:
      ret += "very good";
             break;
    default:
      ret = "undefined";
      break;
  }
  return ret;
}
```

```
int main()
{
  VBAssert(fun(0)=="terrible");
  VBAssert(fun(1)=="terrible");
  VBAssert(fun(2)=="terrible");
  VBAssert(fun(3)=="terrible");
  VBAssert(fun(4)=="terrible");
  VBAssert(fun(5)=="good");
  VBAssert(fun(6)=="good");
  VBAssert(fun(7)=="good");
  VBAssert(fun(8)=="good");
  VBAssert(fun(9)=="very good");
  VBAssert(fun(10)=="outstanding & very good");
  VBAssert(fun(11)=="undefined");
  return 0;
}
```

www.sbVB.com.br

Boolean expressions

```
/*****************************************************
A == B // return true if A equal to B
A != B // return true if A not equal to B
A > B  // return true if A greater than B
A >= B // return true if A greater or equal to B
A < B  // return true if A less than B
A <= B // return true if A less or equal to B

A || B // return A or  B (logic or)
A && B // return A and B (logic and)
!A (logic not)
*****************************************************/

#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

int main()
{
  int i = 3, j = 4, k = 5;
  VBAssert(i < j || j > k);
  VBAssert(!(i < j && j > k && k > i));
  VBAssert(i < j && j > k || k > i);
  return 0;
}
```

www.sbVB.com.br

Bitwise Boolean expressions

Operator	Description	Example
&	AND bitwise	C = A & B
|	OR bitwise	C = A | B
^	XOR bitwise	C = A ^ B
<<	Shift bits left	C = A << B
>>	Shift bits right	C = A >> B
~	Complement bitwise	C = ~A

www.sbVB.com.br

Bitwise Boolean expressions (2)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

#include <iostream>
using namespace std;
int main()
{
  int A, B;
  A = 0xf3; // 1111 0011
  B = 0x8f; // 1000 1111
  VBAssert((A | B) == 0xff); // A or bitwise B,  1111 1111 (ff)
  VBAssert((A & B) == 0x83); // A and bitwise B, 1000 0011 (83)
  VBAssert((A ^ B) == 0x7c); // A xor bitwise B, 0111 1100 (7c)
  VBAssert((A << 3) == 0x798); // A 3 shift left, 0111 1001 1000 (798)
  VBAssert((A >> 3) == 0x1e);  // A 3 shift right, 0001 1110 (1e)
  VBAssert((~A) == 0xffffff0c); // complement of A,
    // 1111 1111 1111 1111 1111 1111 0000 1100 (ffffff0c)
  VBAssert(((~A) << 3) == 0xfffff860);
    // 1111 1111 1111 1111 1111 1000 0110 0000 (fffff860)

  return 0;
}
```

www.sbVB.com.br

Enumerate, enum

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

enum Colors {red, green, blue}; // red = 0
enum {JAN=1, FEB, MAR, APR, MAY, JUN, JUL,
  AUG, SEP, OCT, NOV, DEC}; // anonymous enumerate

int main()
{
  Colors c;
  c = blue;
  VBAssert(c==2);
  VBAssert(c==blue);

  VBAssert(MAR==3);
  return 0;
}
```

enum and switch

```
#include <iostream>
using namespace std;
enum WeekDays {sun, mon, tue, wed, thu, fri, sat};
void whatToDo(WeekDays day)
{
  switch (day)
  {
    case mon:
    case tue:
    case wed:
    case thu:
    case fri:
      cout << "go to work" << endl;
      break;
    case sat:
      cout << "clean the yard and ";
    case sun:
      cout << "relax" << endl;
  }
}
int main()
{
  whatToDo(fri);
  whatToDo(sat);
  whatToDo(sun);
  return 0;
}
```

```
go to work
clean the yard and relax
relax
```

Math functions of the standard library

acos	asin	atan	atan2	sin	tan
cosh	sinh	tanh	exp	frexp	ldexp
log	log10	modf	pow	sqrt	ceil
fabs	floor	fmod			

www.sbVB.com.br

Math functions of the standard library

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
using namespace std;

/* inside VBLib
bool nearEqual(double x, double y, double eps)
{
  return fabs(x-y) < eps;
} */

int main()
{
  double eps=1e-8;
  double d;
  d = 2/5; // dividing ints
  VBAssert(d==0); // would you expect that???
  d = (double)2/(double)5; // type cast assures float division
  VBAssert(nearEqual(d,0.4,eps));
  d = 2.0/5.0;
  VBAssert(nearEqual(d,0.4,eps));
  return 0;
}
```

www.sbVB.com.br

Math functions
frexp

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
#include <cmath>
using namespace std;

int main () {
 double eps = 1e-8;
  double param2, param, result;
  int n;
  param = 8.0;
  result = frexp (param , &n); // [ result, n ] = fresp (param)
  // cout << result << " * 2 ^ " << n << " = " << param << endl;
  VBAssert(nearEqual(result,0.5,eps));
  VBAssert(nearEqual(n,4.0,eps));
  param2 = result * pow(2.0,n);
  VBAssert(nearEqual(param,param2,eps));
```

Math functions
frexp (2)

```
  param = 10.0;
  result = frexp (param , &n);
  VBAssert(nearEqual(result,0.625,eps));
  VBAssert(nearEqual(n,4.0,eps));

  param = 12.5;
  result = frexp (param , &n);
  VBAssert(nearEqual(result,0.78125,eps));
  VBAssert(nearEqual(n,4.0,eps));
  return 0;
}
```

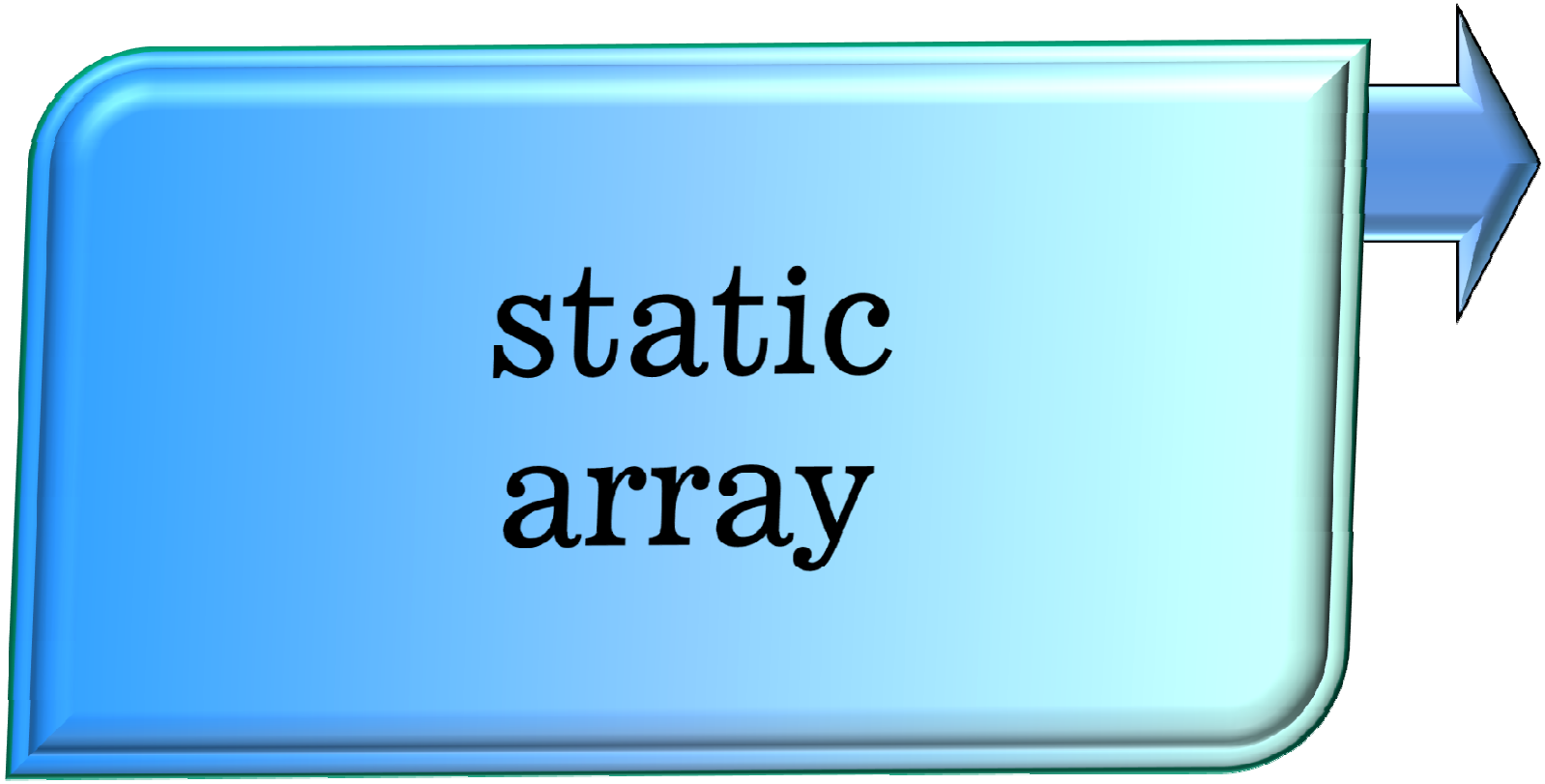
www.sbVB.com.br

ceil, floor

```
#include <cmath>
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

int main()
{
  VBAssert(::floor(3.5)==3);
  VBAssert(::ceil(3.5)==4);

  VBAssert(::floor(-3.5)==-4);
  VBAssert(::ceil(-3.5)==-3);

  VBAssert((int)(3.5)==3);
  VBAssert((int)(-3.5)==-3);

  return 0;
}
```

double converted to int

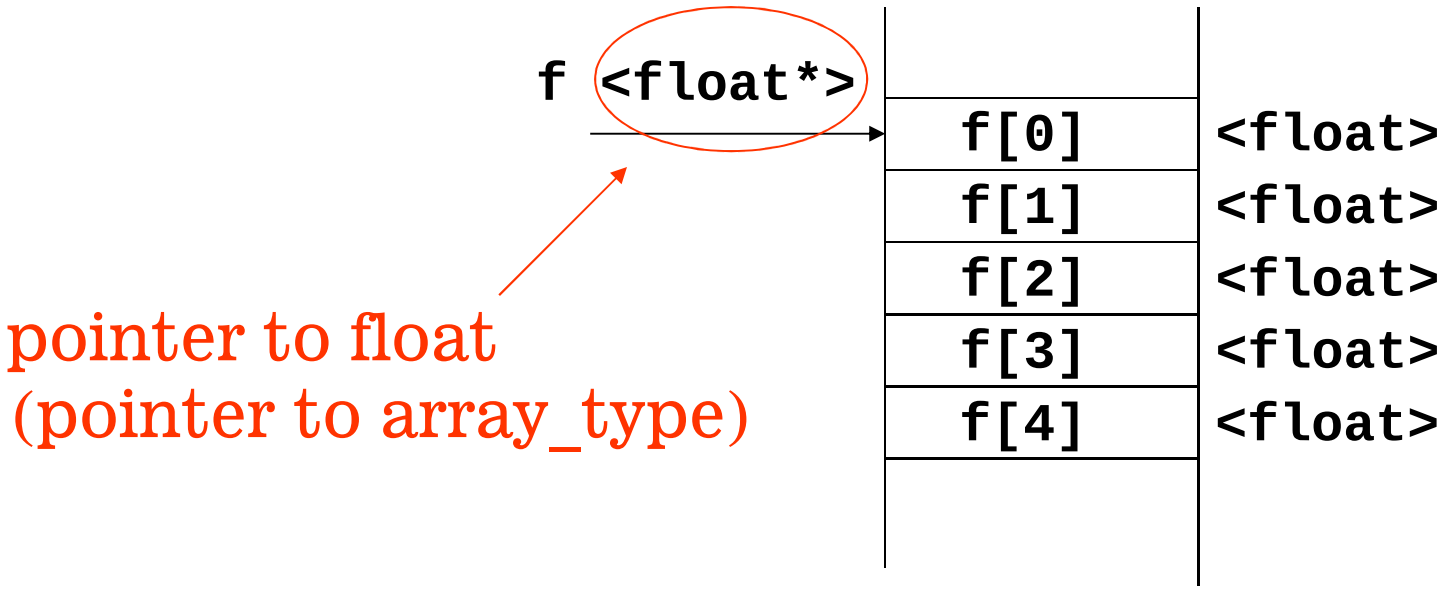
Array



Static array

```
<array_type> <array_name>[<dimension>];
```

```
// example
float f[5];
```



Static array (2)

www.sbVB.com.br

```
#include <iostream>
using namespace std;
int main()
{
  float f[5];
  for (int i=0 ; i < 5 ; i++) {
    f[i] = 1 + i / 10.0;
    cout << "f[" << i << "]="
      << f[i] << endl;
  }
  return 0;
}
```

Bad programming practice. These are 2 related pieces of the code. That should not exist, for the sake of maintainability

wrong

```
#include <iostream>
using namespace std;
#define N 5
int main()
{
  float f[N];
  for (int i=0 ; i < N ; i++)
  {
    f[i] = 1 + i / 10.0;
    cout << "f[" << i << "]="
      << f[i] << endl;
  }
 return 0;
}
```

```
f[0]=1
f[1]=1.1
f[2]=1.2
f[3]=1.3
f[4]=1.4
```

correct

Static array (3)

- The dimension of the static array must be known in *early time* (build time)
- To have an array whose dimension is known in *late time* (execution time) is also possible; this is known as dynamic array, that “alloc memory”
- When using the brackets ([ ]), we are “referencing the pointer” that defines the array.
- There’s no implicit range check.

Static array (4)

```
#include <iostream>
using namespace std;
int main()
{
  const char* names[4] =
    {"Camila", "Mathew", "Sergio", "Mary"};
  for (int i = 0; i < 5; i++)
    cout << "Name[" << i << "] = " << names[i] << endl;
  return 0;
}
```

bug

```
Name[0] = Camila
Name[1] = Mathew
Name[2] = Sergio
Name[3] = Mary
Name[4] = sldflaskf j;alsdkj (not defined)
```

Static array (5)

dimension is defined by the code following

www.sbVB.com.br

```
#include <iostream>
using namespace std;
int main()
{
  double data[] =
  {
     1.23442,
     2.23232,
     4.32343,
     2.31222,
     7.43453,
     6.29387,
     9.42387,
     1.11111,
     2.22222
  };
  unsigned arraySize = sizeof(data) / sizeof(double);

  for (unsigned i=0 ; i < arraySize ; i++)
  {
    cout << "data[" << i << "] = " << data[i] << endl;
  }
  return 0;
}
```

array dimension is defined by the code following

```
data[0] = 1.23442
data[1] = 2.23232
data[2] = 4.32343
data[3] = 2.31222
data[4] = 7.43453
data[5] = 6.29387
data[6] = 9.42387
data[7] = 1.11111
data[8] = 2.22222
```

getting the array dimension

www.sbVB.com.br

Static array (6)

dimension is defined by the code following

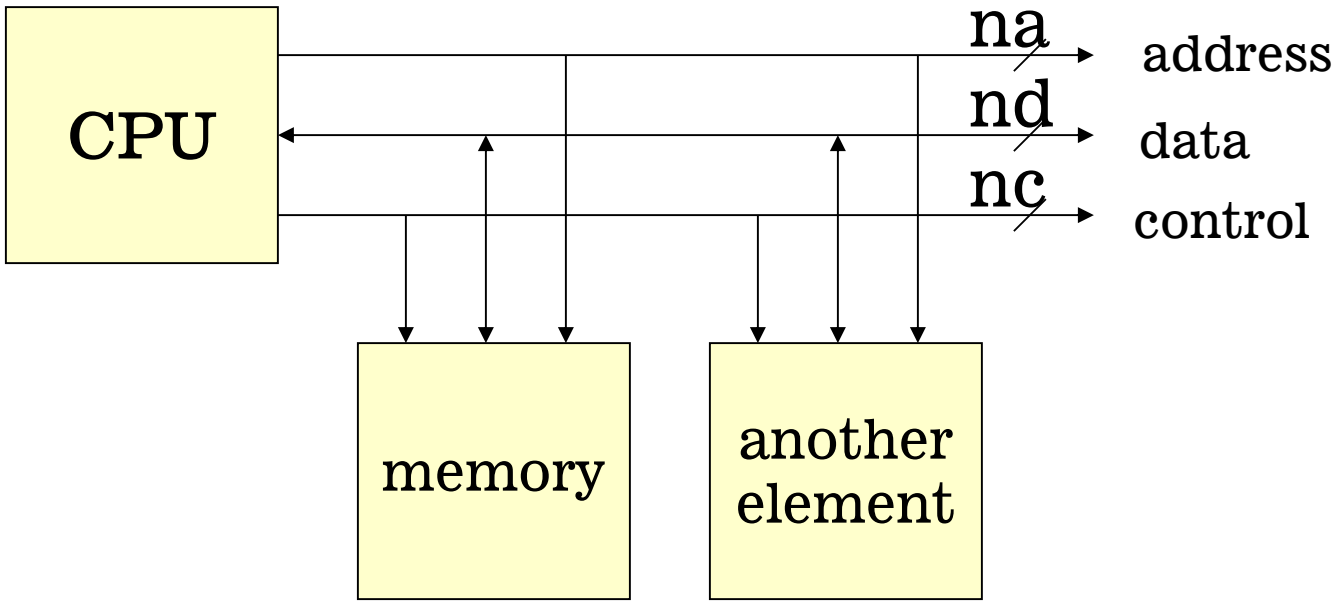
```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

#include <iostream>
using namespace std;
void fun(double *dataIn)
{
  unsigned arraySize = sizeof(dataIn) / sizeof(double);
  VBAssert(sizeof(dataIn)==4); // size of any pointer
  VBAssert(sizeof(double)==8);
  VBAssert(arraySize==0);

  for (unsigned i=0 ; i < arraySize ; i++)
  {
    cout << "dataIn[" << i << "] = " << dataIn[i] << endl;
  }
}
int main()
{
  double data[] =
  {
     1.23442,
     2.23232,
     4.32343,
     2.31222,
     7.43453,
  };
  VBAssert(sizeof(data)==40,"sizeof(data)"); // 20 = 5 * sizeof(double)
  fun(data);
  return 0;
}
```

can't get the array dimension from inside a function (the result is wrong)

array dimension is defined by the code following

Pointers

Pointer



Using CPU 386 (or more recent),
na=32 (that is: 4 bytes or 8 nibbles); 2^{32}=4G
more recently,
na=64 (that is: 8 bytes or 16 nibbles); 2^{64}=16GG

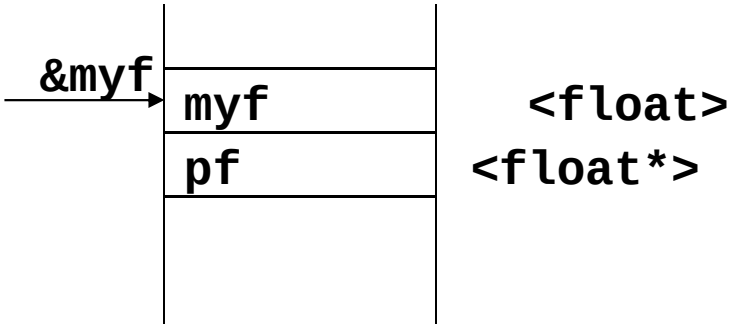
Pointers (2)

- The size of a pointer is related to the number of tracks in the address bus of the CPU. There's no relation between the pointer size and the size of the object it points to.
- The pointer can point to data or to code
  - To point to data means to point to variables (objects).
  - A pointer can point to a data type that is itself a pointer. In this case, it turns to be a "pointer to pointer". There's no limit of how many times a pointer can point to another pointer.
  - To point to code is "point to function".

Pointer (3)

www.sbVB.com.br

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include<iostream>
using namespace std;
int main ()
{
  float myf = 1.1f;
  float *pf;
  pf = &myf;
  *pf = 2.2f;
  VBAssert(myf==2.2f);
  cout << "pf=" << pf << endl;
  return 0;
}
```

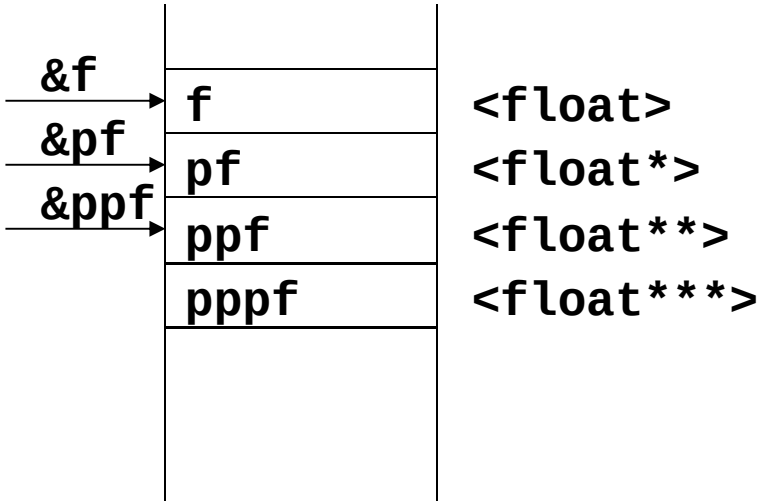



```
pf=0x0066FDF4
```

Pointer to pointer

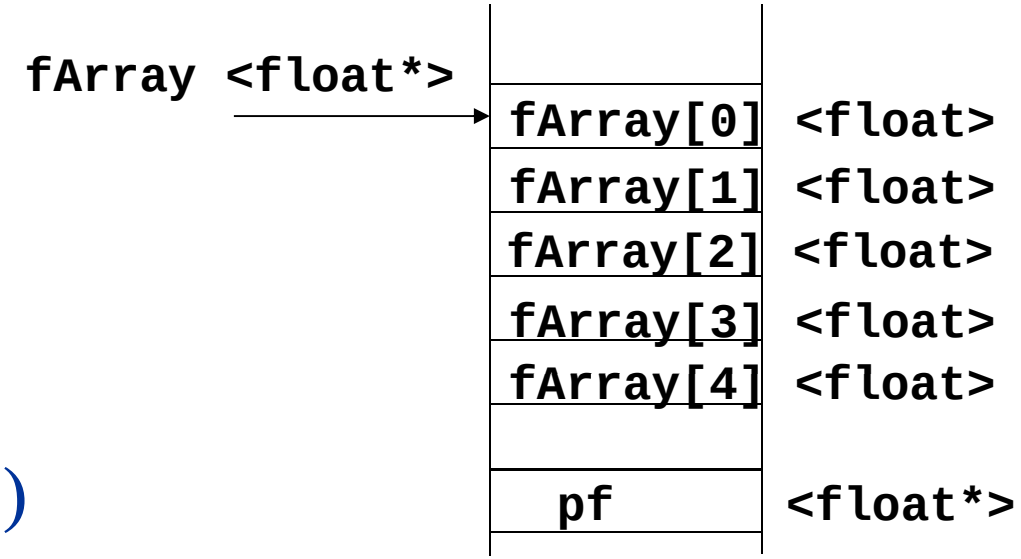
www.sbVB.com.br

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include<iostream>
using namespace std;
int main()
{
  float f = 1.1f;
  float *pf = &f;
  float **ppf = &pf;
  float ***pppf = &ppf;
  *pf = 2.2f;
  VBAssert(f==2.2f);
  **ppf = 3.3f;
  VBAssert(f==3.3f);
  ***pppf = 4.4f;
  VBAssert(f==4.4f);
  return 0;
}
```



Arrays and pointers
pointer arithmetic

- *(p+i) is equivalent to p[i]
- When adding a pointer to an int, automatically it is multiplied by the size of type
  - p+i*sizeof(TYPE)
- The syntax meaning (number of times "pointer to pointer") of * at left is the same as brackets [ ] at right.

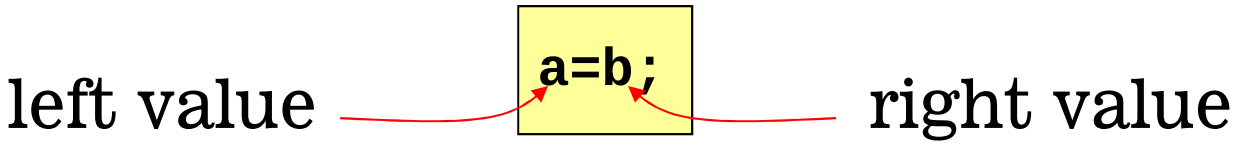


www.sbVB.com.br

Arrays and pointers
pointer arithmetic

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
int main() {
  float fArray[5];
  float *pf;
  pf = fArray; // OK
  pf[2] = 2.2f; // *(pf+2)=2.2f;
  VBAssert(fArray[2]==pf[2]);
  *(fArray+3) = 4.4f;
  VBAssert(pf[3]==4.4f);
  fArray = pf; // error
    // error C2440: '=' : cannot convert
    // from 'float *' to 'float [5]'
  return 0;
}
```

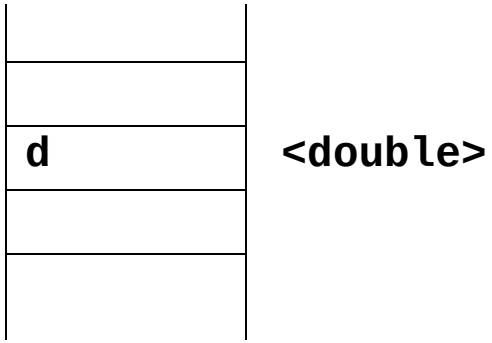
rvalue and lvalue




```
int main()
{
  int i;
  i = 5; // OK
  5 = i; // error:'=' : left operand must be l-value

  double edisonArantesDoNascimento;
  double & pele = edisonArantesDoNascimento; // nickname

  return 0;
}
```

rvalue and lvalue (2)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
using namespace std;
int main()
{
  double d = 1.1;
  double & lvalue_d = d;
  double & lvd; // error C2530: 'lvd' :
    // references must be initialized

  lvalue_d = 2.2;
  VBAssert(d==2.2);
  VBAssert(lvalue_d==2.2);
  VBAssert(&d==&lvalue_d);
  return 0;
}
```



Arguments by value and by reference

www.sbVB.com.br

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

void f1(double z) {
  z = 1.1;
}
void f2(double *x) {
  *x = 2.2;
}
void f3(double & v) {
  v = 3.3;
}
int main() {
  double a = 10.1;
  VBAssert(a==10.1);
  f1(a);
  VBAssert(a==10.1);
  f2(&a);
  VBAssert(a==2.2);
  f3(a);
  VBAssert(a==3.3);
  return 0;
}
```

f1: z=a (by value)

f2: x=&a (pseudo by reference)

f3: v=a (true by reference)

!

lvalue and a function that returns more than one argument

www.sbVB.com.br

$$(x, y) = f(a, b, c)$$

$$x = a + b + c$$

$$y = \frac{2a + 3b + 4c}{9}$$

x, y are the returns of myFun

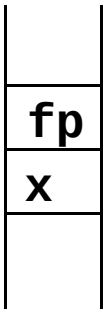
```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

void myFun(double a, double b, double c, double & x, double & y) {
  x = a + b + c;
  y = (2*a + 3*b + 4*c)/9;
}

int main() {
  double r, s;
  double eps = 1e-8;
  myFun(1.1,2.2,3.3,r,s);
  VBAssert(nearEqual(r,6.6,eps));
  VBAssert(nearEqual(s,2.44444444,eps));
  return 0;
}
```

One shouldn't compare floating point numbers (scientific computing) with ==, but with nearEqual

Pointer to function

www.sbVB.com.br

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include<cmath>
#include<iostream>
using namespace std;
double sin2(double x) {
  double ret = ::sin(x);
  ret *= ret;
  return ret;
}
double cos2(double x) {
  double ret = ::cos(x);
  ret *= ret;
  return ret;
}
int main ()
{
  // function pointer
  double (*fp)(double);
  double x = 0.4;
  fp = sin2;
  VBAssert(fp(x)==sin2(x));
  VBAssert(fp(x)==::sin(x)*::sin(x));
  fp = cos2;
  VBAssert(fp(x)==cos2(x));
  VBAssert(fp(x)==::cos(x)*::cos(x));
  return 0;
}
```




```
<double (*)(double)>
<double>
```

Pointer to function (2)

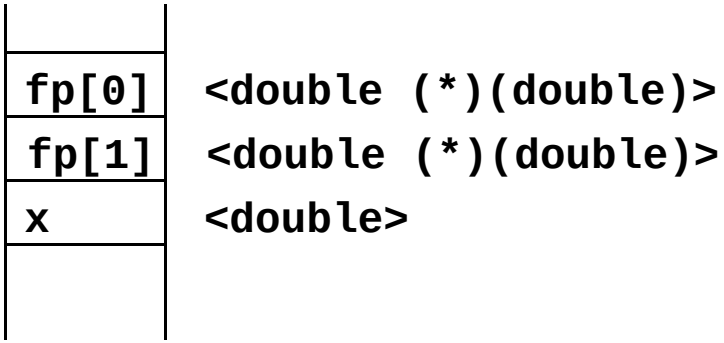
```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include<cmath>
#include<iostream>
using namespace std;

// return square of given function
double fun2(double (*fp)(double), double x)
{
  double ret = fp(x);
  ret *= ret;
  return ret;
}

int main ()
{
  double x = 0.4;
  VBAssert(fun2(::sin,x)==::sin(x)*::sin(x));
  VBAssert(fun2(::cos,x)==::cos(x)*::cos(x));
  return 0;
}
```

Pointer to function (2)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include<cmath>
#include<iostream>
using namespace std;

// return square of given function
double fun2(double (*fp)(double), double x)
{
  double ret = fp(x);
  ret *= ret;
  return ret;
}

double mitochondria(double z)
{
          return sqrt(2*z + 1.1);
}

int main ()
{
  double x = 0.4;
  VBAssert(fun2(mitochondria,mitochondria(x))==
            mitochondria(mitochondria(x))*
            mitochondria(mitochondria(x)));
  return 0;
}
```

Pointer to function array

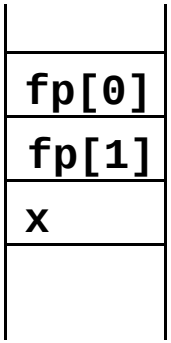
```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include<cmath>
#include<iostream>
using namespace std;
double sin2(double x) {
  double ret = ::sin(x);
  ret *= ret;
  return ret;
}
double cos2(double x) {
  double ret = ::cos(x);
  ret *= ret;
  return ret;
}
int main () {
  double (*fp[2])(double);
  // array of function pointer
  double x = 0.4;
  fp[0] = sin2;
  fp[1] = cos2;
  VBAssert(fp[0](x)==sin2(x));
  VBAssert(fp[1](x)==cos2(x));
  return 0;
}
```



typedef and pointer to function

www.sbVB.com.br

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include<cmath>
#include<iostream>
using namespace std;
double sin2(double x) {
  double ret = ::sin(x);
  ret *= ret;
  return ret;
}
double cos2(double x) {
  double ret = ::cos(x);
  ret *= ret;
  return ret;
}
typedef double (*fp_type)(double);
int main ()
{
  fp_type fp[2];
  // array of function pointer
  double x = 0.4;
  fp[0] = ::sin2;
  fp[1] = ::cos2;
  VBAssert(fp[0](x)==::sin2(x));
  VBAssert(fp[1](x)==::cos2(x));
  return 0;
}
```




```
<double (*)(double)>
<double (*)(double)>
<double>
```

constness

constness (const)

```
void f1() {
  const int m = 3; // OK. a value can be set to a const
     // in the moment of its instantiation

  const float v[] = { 1.1, 2.2, 3.3 }; // const vector
     // values set in the moment of instantiation

  const int & pm = m; // OK. One can get the const lvalue
     // of a const

  m = 100; // error C2166: l-value specifies const object
  m++; // error C2105: '++' needs l-value
  v[1] = 5.5; // error C2166: l-value specifies const object
  int & pm2 = m; // error C2440: 'initializing' :
     // cannot convert from 'const int' to 'int &'

}
```

www.sbVB.com.br

constness (const) (2)

```
void fun_non_const(int & b)
{
  b++; // OK. a non-const can be written to
}

void fun_const(const int & a)
{
  a++; // error C2166: l-value specifies const object
  fun_non_const(a); // error C2664: 'fun_non_const' :
    // cannot convert parameter 1 from 'const int' to 'int &'
  int i = a; // OK. a l-value can be read from
}

void f4(const int & k) {
  int z = k; // OK. a l-value can be read from
  k = 4; // error C2166: l-value specifies const object
  fun_const(k); // OK. One can get the const lvalue of a const
  fun_const(z); // OK. One can get the const lvalue of a non const
  fun_non_const(k); // error C2440: 'initializing' :
    // cannot convert from 'const int' to 'int &'
  fun_const(6); // OK. One can get the const lvalue
    // of a constant literal
  fun_non_const(6); // error C2440: 'initializing' :
     // cannot convert from 'const int' to 'int &'
}
```

constness (const) (3)

www.sbVB.com.br

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;


int main()
{
  int q = 5;
  q++;

  VBAssert(q==6);
  const int & lq = q;
  q++;
  VBAssert(lq==7);


  lq++; // error

  return 0;
}
```

lq is const nickname of q

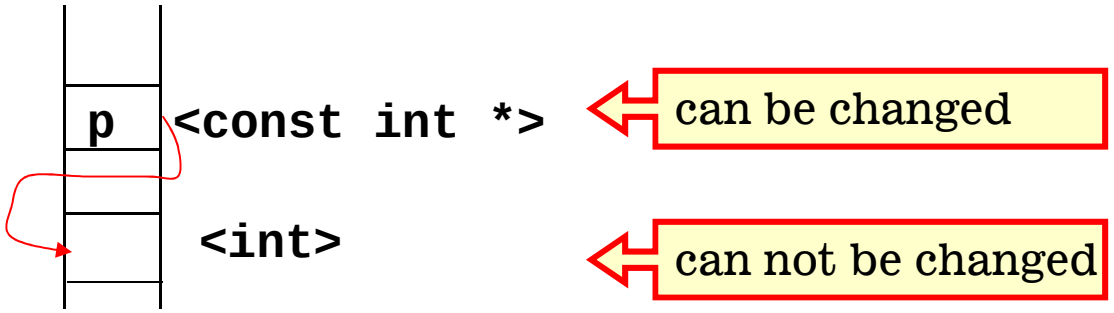
lq altered

www.sbVB.com.br

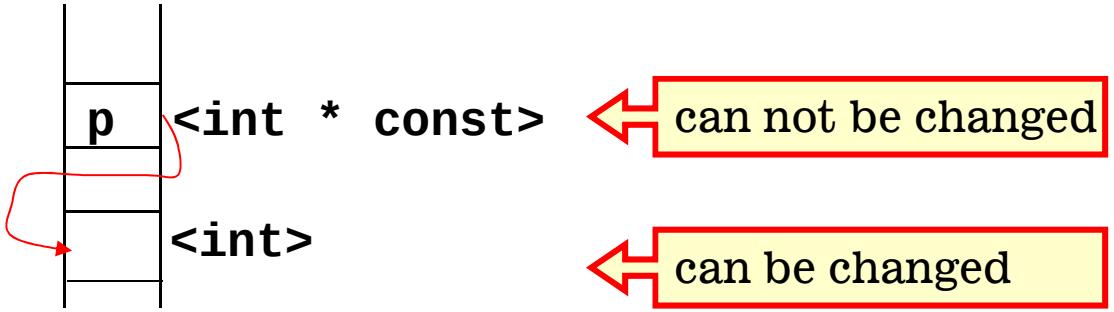
const and pointers

```
void f1(const int * p,const int * p2)
{
  p[2] = 3; // error C2166: l-value specifies const object
    // can not set value to the referenced pointer

  p = p2; // OK; a value can be set to the pointer because
    // it is not const itself
}

void f2(int * const p,int * const p2)
{
  p[2] = 3; // OK; can set a value to the referenced pointer

  p = p2; // error C2166: l-value specifies const object
    // a value can not be set to the pointer because it is
    // itself a const
}

void f3(int * p, const int * const p2)
{
  p[2] = 3; // OK
  p = p2; // OK
  p2 = p; // error C2166: l-value specifies const object
  p2[2] = 3; // error C2166: l-value specifies const object
}
```

const and pointers (2)




```
p[2] = 3; // error
p = p2; // ok
```




```
p[2] = 3; // OK
p = p2; // error
```

const and pointer to pointer

- to be continued !!!!

Memory alloc

Memory alloc (C++ style)

- There are only 2 situations to consider
- A multidimensional array can be created as a nesting of single dimension arrays

1) Alloc of 1 object

```
int main ()
{
  double *pd;
  pd = new double;
  *pd = 1.1; // pd[0] = 1.1;
  delete pd;
  return 0;
}
```

2) Alloc of an array of objects

```
int main ()
{
  int n = 10;
  double *pd;
  pd = new double [n];
  for (int i=0 ; i < n ; i++)
    pd[i] = i + 1.1;
  delete [] pd;
  return 0;
}
```

Don't forget to use these brackets [ ] to delete an array of objects

Here the variable that contains the number of objects to be allocated

Treating the memory alloc fail

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include<iostream>
using namespace std;
int main ()
{
  double *pd;
  pd = new double;
  VBAssert(pd!=0, "Memory Alloc error");
  *pd = 1.1; // pd[0] = 1.1;
  delete pd;
  return 0;
}
```

```
// bug example due to double pointer delete
int main ()
{
  double *pd = new double;
  delete pd;
  delete pd; // bug !
  return 0;
}
```

To delete a memory more than once generates a bug in the execution of the program. Optionally, use the safe delete, as below.

```
if (pd) { delete pd; pd=0; }
```

www.sbVB.com.br

Possible Scopes of an Object

- Any object (variable) that exists in some software must belong to one of the scopes below
  - global scope
  - stack
  - heap

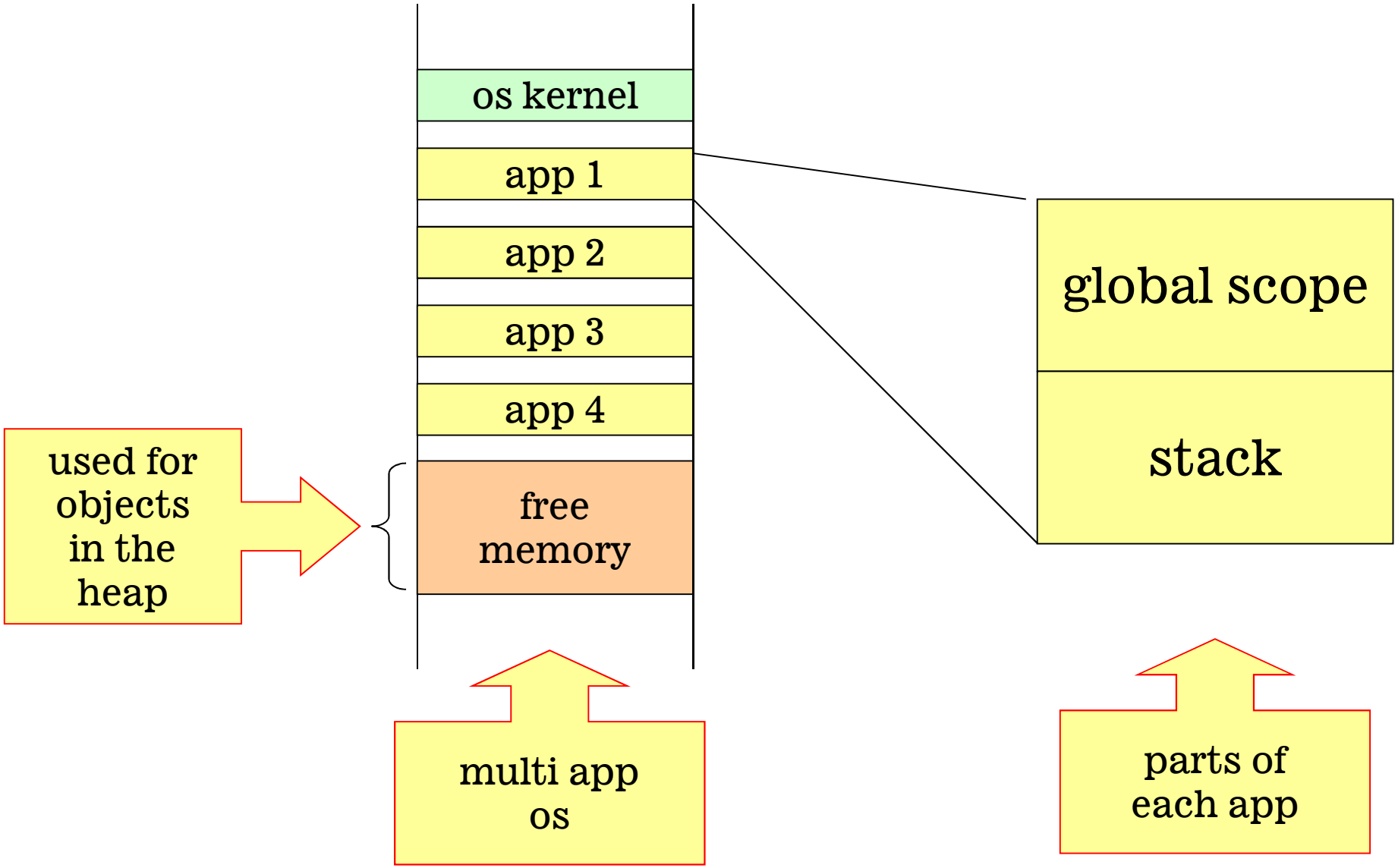
www.sbVB.com.br

Example of Possible Scopes of an Object

```
// in this example, when you see “double”,
// think of “example_of_user_type”

// “d_global” exists in global scope
double d_global;

int main ()
{
  // “d_stack” exists in stack
  double d_stack;

  // “d_static” exists in global scope
  static double d_static;

  double *pd; // “pd” exists in stack
  // but actual double object exists in the heap
  pd = new double;
  *pd = 1.1; // referencing and using object in heap
  delete pd;

  return 0;
}
```

Applications in a Multi App Operating System

www.sbVB.com.br

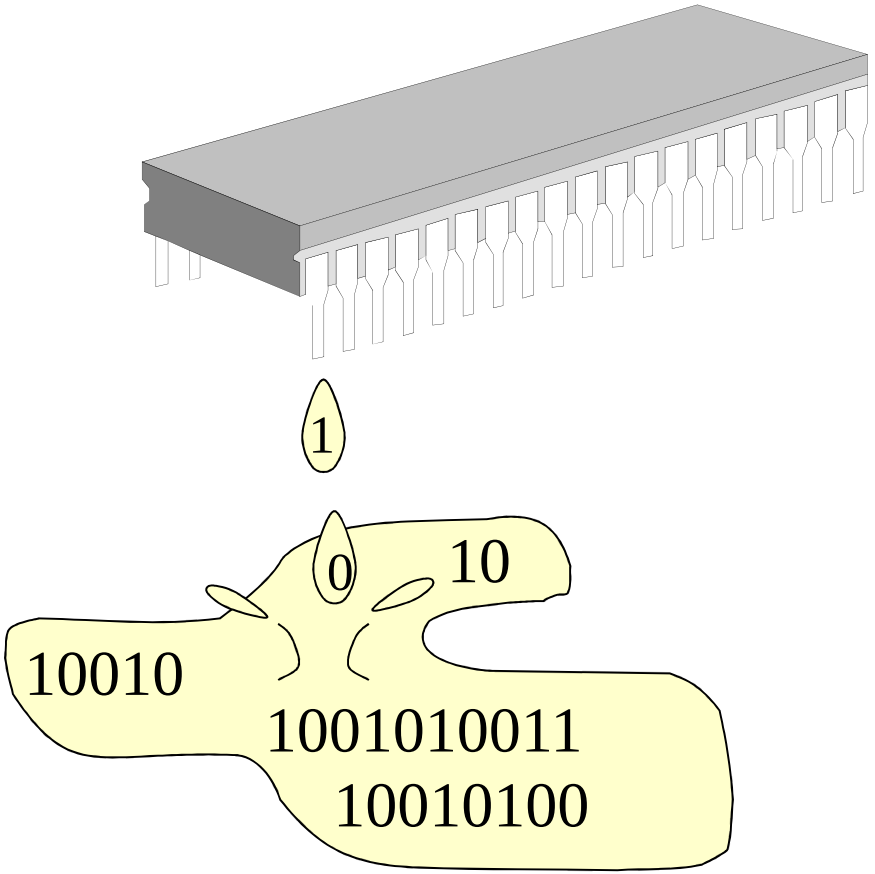


www.sbVB.com.br

Applications in an OS (2)

- Each application has its own global scope space, and its own stack space.
  - The app's global space is limited by the application's size.
  - The app's stack is defined by a build option.
- The free memory of the OS is available for the app's to be used as space for variables in the heap.

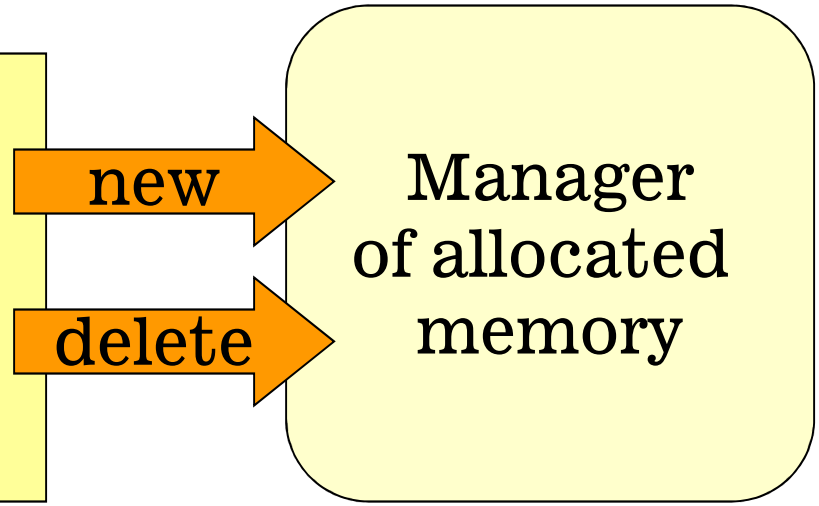
Memory leak



Memory leak (2)

Heap Manager

```
// mem leak example
int main ()
{
  double *pd = new double;
//  delete pd;
  return 0;
}
```



Memory leak (3)

- The detection of memory leak, if exists, depends on the compiler, that is, is non portable.
- A code that does not produce memory leak in computer/operating system A, won't leak in computer/operating system B.
  - So, the developer can debug the code in system A (e.g. Windows) and use in system B (e.g. Linux).

Static multidimensional array

```
#include <iostream>
using namespace std;
const int ROW = 3;
const int COL = 2;
typedef double MyMatrix[ROW][COL]; // define type MyMatrix
int main()
{
  MyMatrix m1,m2,m3;
  // double m1[ROW][COL], m2[ROW][COL], m3[ROW][COL];
  for (int i=0 ; i < ROW ; i++)
  {
    for (int j=0 ; j < COL ; j++)
    {
      m1[i][j] = 1 + i*10 + j*100;
      cout << m1[i][j] << "  ";
    }
    cout << endl;
  }
  return 0;
}
```

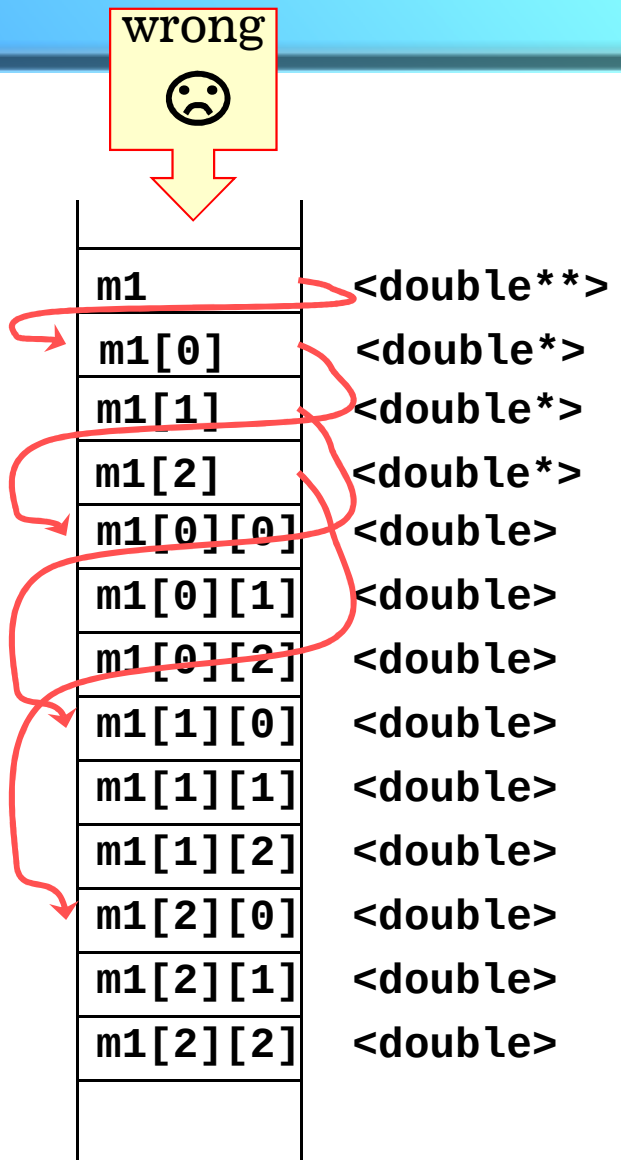
```
1  101
11  111
21  121
```

Static multidimensional array (2)

www.sbVB.com.br

```
#include <iostream>
using namespace std;
#define ROW 3
#define COL 3
typedef double myMatrix[ROW][COL];
int main()
{
  myMatrix m1;
  for (int i=0 ; i < ROW ; i++)
  {
    for (int j=0 ; j < COL ; j++)
    {
      m1[i][j] = 1 + i*10 + j*100;
      cout << m1[i][j] << "  ";
    }
    cout << endl;
  }
  return 0;
}
```

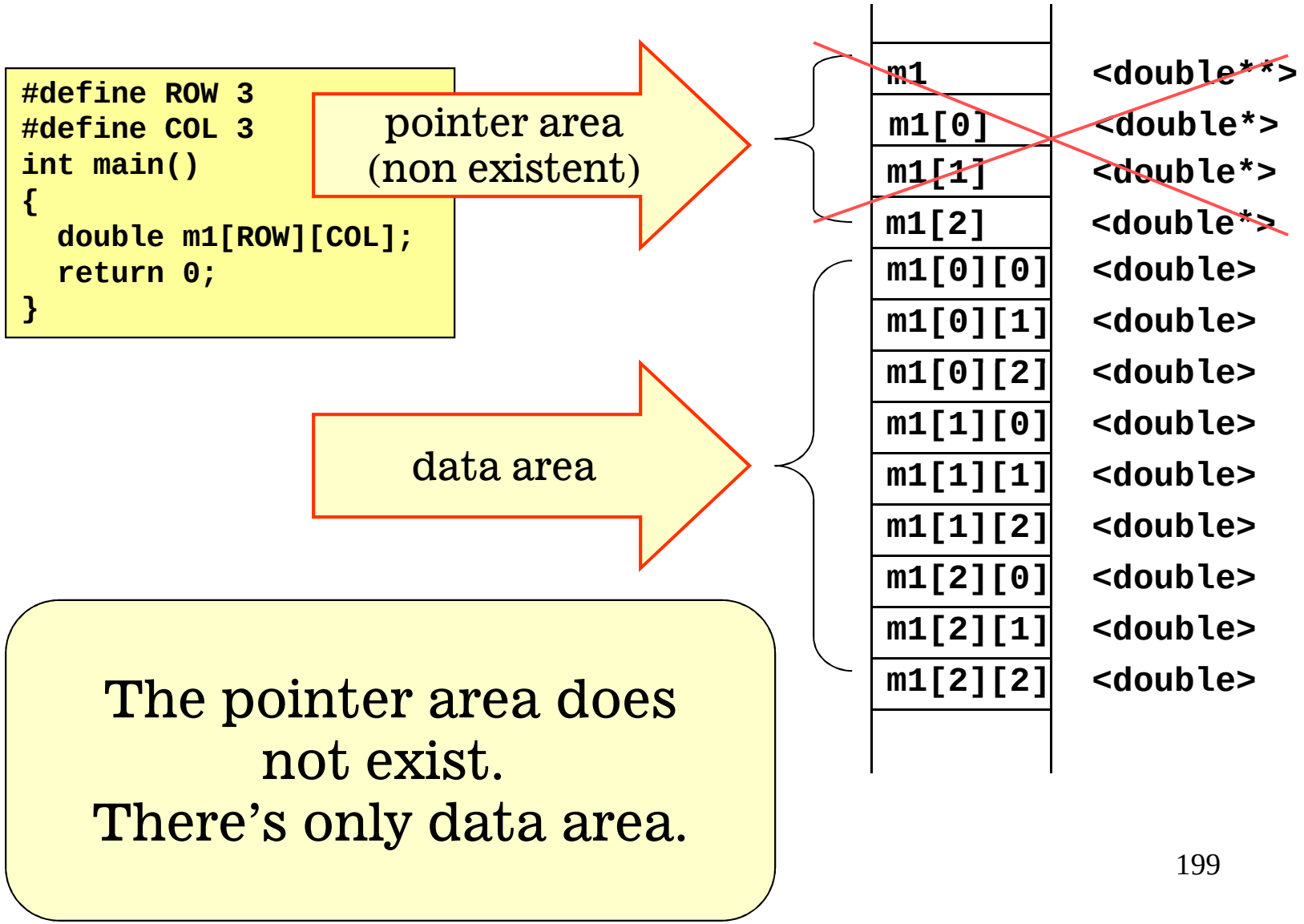
That is correct for dynamic multidimensional array, but not correct for static multidimensional array.



Static multidimensional array (3)

www.sbVB.com.br

```
#define ROW 3
#define COL 3
int main()
{
  double m1[ROW][COL];
  return 0;
}
```



The pointer area does not exist.
There's only data area.

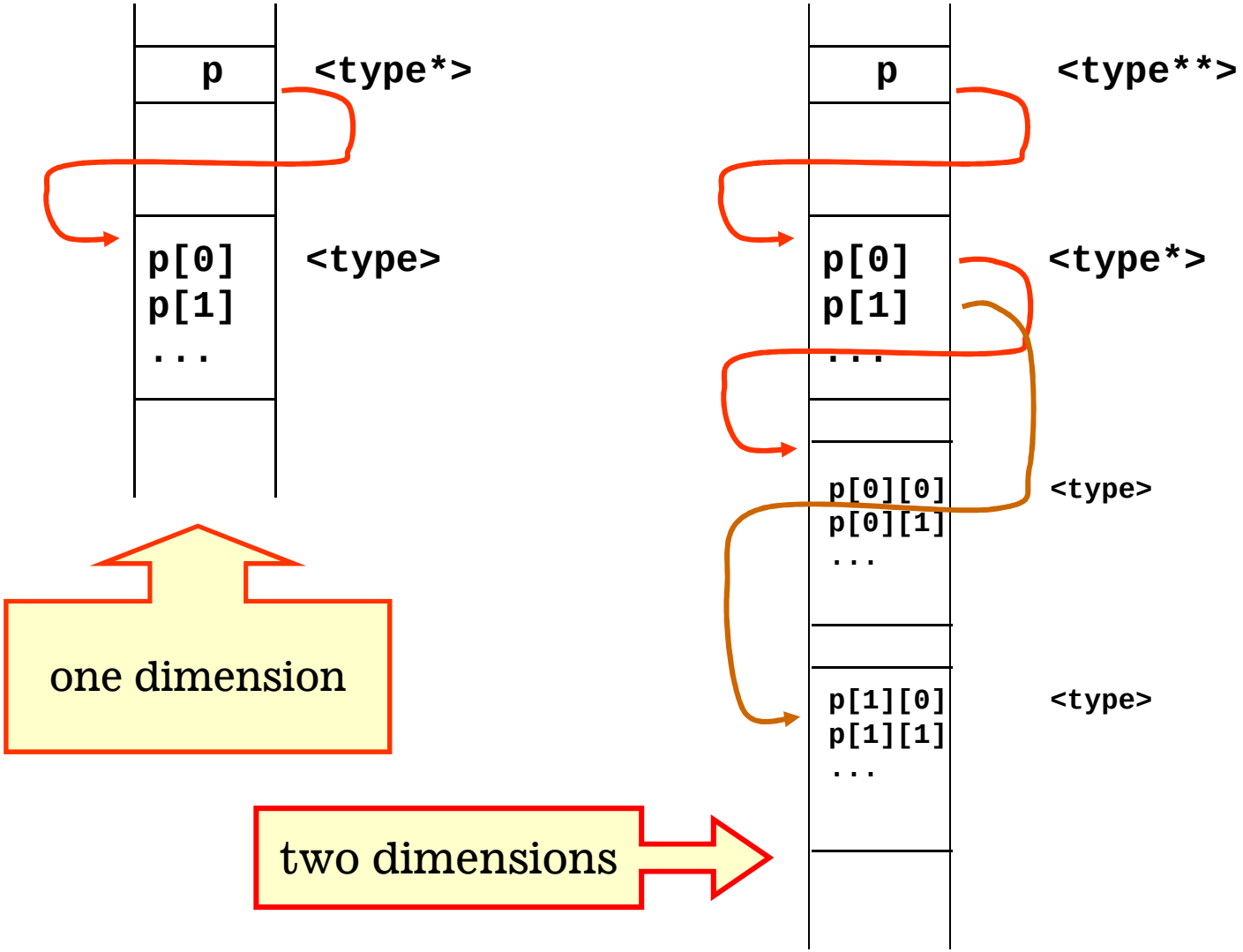
Passing a static multidimensional array as argument

```
#include <iostream>
void fun(int arrayThreeDimensions[][3][4])
{
  int i,j,k;
  for (i=0 ; i < 2 ; i++)
    for (j=0 ; j < 3 ; j++)
      for (k=0 ; k < 4 ; k++)
        ::std::cout << arrayThreeDimensions[i][j][k]
          << ::std::endl;
}
int main()
{
  // <int *arrayType[3][4]>
  // <int arrayType[][3][4]>
  int threeDimensions[][3][4] =
  {
    {
      {111, 112, 113, 114},
      {121, 122, 123, 124},
      {131, 132, 133, 134}
    },
    {
      {211, 212, 213, 214},
      {221, 222, 223, 224},
      {231, 232, 233, 234}
    }
  };
  fun(threeDimensions);
  return 0;
}
```

define all dimensions except the one at left

```
111
112
113
114
121
122
123
124
131
132
133
134
211
212
213
214
221
222
223
224
231
232
233
234
```

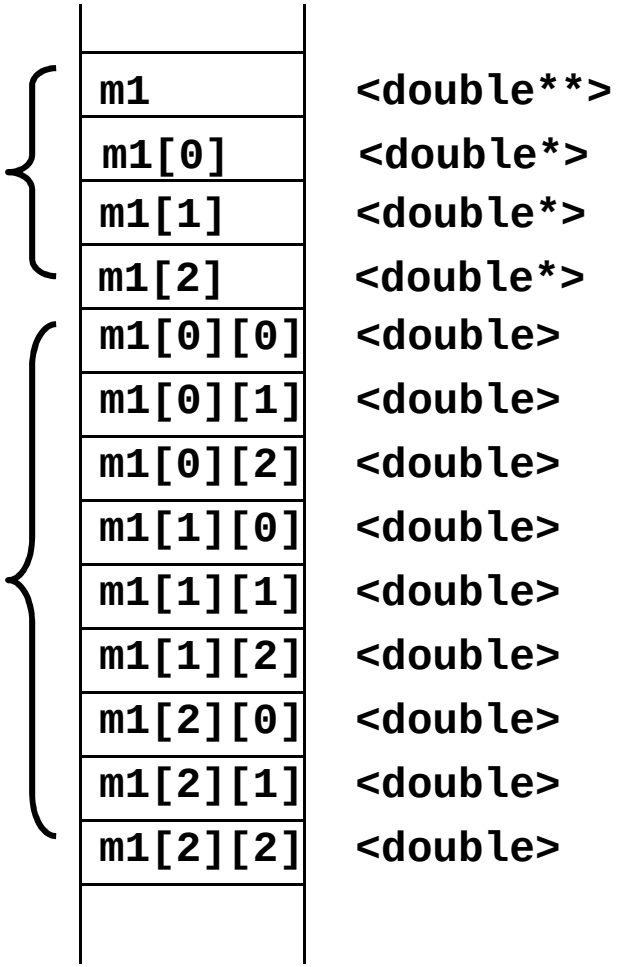
Dynamic multidimensional array



Dynamic multidimensional array (2)

www.sbVB.com.br

```
#define TYPE double
#include <iostream>
using namespace std;
TYPE **allocMatrix(int r, int c) {
  TYPE **ret;
  ret = new TYPE* [r];
  for (int i=0 ; i < r ; i++)
    ret[i] = new TYPE [c];
  return ret;
}
void fillMatrix(TYPE **p, int r, int c,
  double d, bool show) {
  for (int i=0 ; i < r ; i++) {
    for (int j=0 ; j < c ; j++) {
      p[i][j] = d;
      if (show) cout << p[i][j] << " ; ";
    }
    if (show) cout << endl;
  }
}
void deleteMatrix(TYPE **p, int r) {
  for (int i=0 ; i < r ; i++)
    delete [] p[i];
  delete [] p;
}
```



Dynamic multidimensional array (3)

```
void do_all()
{
  TYPE **p;
  int row = 3, col = 3;
  p = allocMatrix(row,col);
  fillMatrix(p,row,col,1.1,true);
  deleteMatrix(p,row);
}

int main()
{
  do_all();
  return 0;
}
```

Compiler directives

#define

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>

#define Z N+3
#define N 3
#define X uncompilable_code
#define MY_FUN(x) ((x)*(x))

int main()
{
  VBAssert(N==3);
  VBAssert(Z==6);
  unsigned i=N;
  VBAssert(i==3);

  VBAssert(MY_FUN(3.3+1.1)==(3.3+1.1)*(3.3+1.1));
  double d = MY_FUN(3.3+1.1);
  VBAssert(d==(3.3+1.1)*(3.3+1.1));

  return 0;
}
```

don't use ; (semi-colon)

#define (2)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>

int myFun(int i) {
  return (i+1)*3 - 8;
}

#define MY_FUN(x) \
((x)+1)  /* comment 1 */ \
*3  /* comment 2 */ \
-8  /* comment 3 */

int main()
{
  for (int i=-100; i<100; i++)
    VBAssert(myFun(i)==MY_FUN(i));

  return 0;
}
```

#define (3)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
using namespace std;

#define MY_MESSAGE \
"This is a program that does many things\n\
One thing\n\
Other thing\n\
Yet another thing"

int main()
{
  cout << MY_MESSAGE << endl;
  return 0;
}
```

```
This is a program that does many things
One thing
Other thing
Yet another thing
```

#define redefinition

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
using namespace std;

#define MY_MESSAGE "Message 1"

void fun_1() {
  VBString s = MY_MESSAGE;
  VBAssert(s=="Message 1");
}

// overwrite MY_MESSAGE with different meaning
#define MY_MESSAGE "Message 2"

void fun_2() {
  VBString s = MY_MESSAGE;
  VBAssert(s=="Message 2");
}

int main()
{
  fun_1();
  fun_2();
  return 0;
}
```

www.sbVB.com.br

(for string)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

#define PATH(logid,cmd) "/usr/" #logid "/bin/" #cmd
#define PATH2(logid,cmd) "/usr/" logid "/bin/" cmd

int main ()
{
  VBString userPath = PATH(francisco,readmail);
  // expanded to: VBString userPath =
  //   "/usr/" "francisco" "/bin/" "readmail";
  // equivalent to: VBString userPath =
  //    "/usr/francisco/bin/readmail";
  VBAssert(userPath=="/usr/francisco/bin/readmail");

  VBString userPath2 = PATH2("francisco","readmail");
  VBAssert(userPath2=="/usr/francisco/bin/readmail");
  return 0;
}
```

(for string) (2)

```
#include <iostream>
using namespace std;
#define GETVAR(x) VBString x = cgi.getVarContent(#x)
int main ()
{
  GETVAR(name);
  // expanded to:  VBString name = cgi.getVarContent("name");
  return 0;
}
```

Conditional compilation

```
#include <cmath> // sin
#include <iostream>
using namespace std;
#define DEBUG

void show(double z)
{
#if 0
/* commented out
  double q;
// this function is a test
  q = 1.1;
  cout << q << endl;
*/
  cout << z << endl;
#endif
}

int main ()
{
 double a,b,c,d;
  a = ::sin(0.4);
#ifdef DEBUG
  cout << "DEBUG a=" << a << endl;
#endif
  show(a);
  return 0;
}
```

```
DEBUG a=0.389418
```

Conditional compilation (2)

```
// this is a cross platform code

// additional includes for dll and so
#ifdef WIN32
// #ifdef _MSC_VER  // Visual C++ only !!!
  // code for Windows
  #include <windows.h>
#else
  // code for Unix
  #include <dlfcn.h>
#endif
```

Conditional compilation and #error

```
// this code is for Unix only

// additional includes for dll and so
#ifdef _MSC_VER  // Visual C++ only !!!
  #error This code is for Unix only
#else
  // code for Unix
  #include <dlfcn.h>
#endif
```

#error
generating a compilation error

```
#include <cmath> // sin
#include <iostream>
using namespace std;
#define DEBUG
#define DEBUG_SPECIAL
// #define DEBUG_MATH

void show(double z)
{
#ifdef DEBUG_SPECIAL
  cout << z << endl;
#endif
}

int main ()
{
#ifdef DEBUG
 double a,b,c,d;
 a = ::sin(0.4);
#ifdef DEBUG_MATH
#error This line is to be compiled
  cout << "a=" << a << endl;
#endif // DEBUG_MATH
  show(a);
#endif // DEBUG
  return 0;
}
```

This source produces or not a compilation error ?

#pragma, extra information for the compiler

- pragma is a non portable way to set extra information to the compiler
  - For example: to disable one specific warning

```
// disable warning #4786
#pragma warning(disable:4786)
```

www.sbVB.com.br

#pragma, extra information for the compiler

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
using namespace std;

#pragma warning(disable:4018)

int main ()
{
      int i=3;
      unsigned u=5;
      if (i<u) // warning C4018: '<' :
               //   signed/unsigned mismatch
            cout << "i<u" << endl;

  return 0;
}
```

Compiler directives

concatenation of identifiers

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
using namespace std;

#define TIPO(a) fut ## a ## l
int main ()
{
  VBString futsal = "futebol de salão";
  VBString futbol = "tradicional futebol de campo";

  VBAssert(TIPO(sa) == "futebol de salão");
  VBAssert(TIPO(bo) == "tradicional futebol de campo");

  return 0;
}
```

and code "pseudo generic"

```
// normal struct
struct Point
{
  double x, y;
};

// define pseudo generic struct
#define POINT(type) \
struct Point_ ## type \
{ \
  type x, y; \
}

// define typed structs from pseudo generic struct
POINT(double); // Point_double
POINT(int); // Point_int

int main ()
{
  Point a; a.x = 1.1; a.y=2.2;
  Point_double b; b.x = 1.1; b.y=2.2;
  Point_int c; c.x = 1; c.y=2;

  return 0;
}
```

After the concept of "template" was introduced to C++, the use of ## as pseudo-generic code became deprecated (but still works)

Miscellaneous

Arguments of main function

```
// test.cpp
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
  for (int i=0; i < argc ; i++)
    cout << "arg["<< i << "]: " << argv[i] << endl;
  return 0;
}
```

```
$ test -c filename.dat "one string with spaces"↵
arg[0]: test.exe
arg[1]: -c
arg[2]: filename.dat
arg[3]: one string with spaces
$
```

struct

```
struct MyStruct
{
 int i;
 double d;
};

int main()
{
  MyStruct a, b;
  a.i = 1;
  a.d = 1.1;
  b.i = 2;
  b.d = 2.2;
  b = a; // copy entire struct
  VBAssert(b.i==1);
  VBAssert(b.d==1.1);
  return 0;
}
```

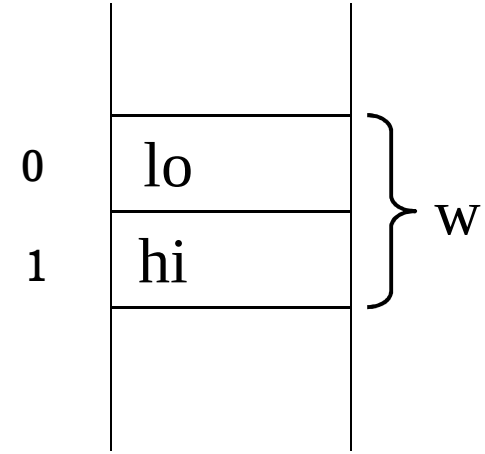
"free" assignment operator (operator=)

struct (2)

```
struct MyStruct
{
 int i;
 double d;
};

int main()
{
  MyStruct a, b, *p;
  a.i = 1;
  a.d = 1.1;

  p = &a;
  p->i = 2;   // (*p).i = 2;
  p->d = 2.2; // (*p).d = 2.2;
  p = &b;
  *p = a; // b = a
  VBAssert(b.i==2);
  VBAssert(b.d==2.2);

  p = new MyStruct;
  p->i = 2;   // (*p).i = 2;
  p->d = 2.2; // (*p).d = 2.2;
  delete p;
  return 0;
}
```

union

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
using namespace std;
typedef unsigned char Byte; // sizeof(Byte)==1
typedef unsigned short Word; // sizeof(Word)==2
union MyUnion
{
  struct // anonymous struct
  {
    Byte lo;
    Byte hi;
  };
  Word w;
};
int main()
{
  MyUnion u; // VBAssert(sizeof(MyUnion)==2);
  u.w = 2;
  VBAssert(u.w==2);
  u.hi = 1; u.lo = 0; VBAssert(u.w==256);
  u.hi = 1; u.lo = 1; VBAssert(u.w==257);
  u.hi = 2; u.lo = 1; VBAssert(u.w==513);
  return 0;
}
```



w and (hi, lo)
use the same memory

w = 1*256 + 0

www.sbVB.com.br

Type cast

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
using namespace std;

int main()
{
  double eps=1e-8;
  double d;
  d = 2/5; // dividing ints
  VBAssert(d==0); // would you expect that???
  d = (double)2/(double)5; // type cast assures float division
  VBAssert(nearEqual(d,0.4,eps));
  d = 2.0/5.0;
  VBAssert(nearEqual(d,0.4,eps));
  d = 5.99999;
  int i = d; // conversion from double to int: truncation
  VBAssert(i==5);
  return 0;
}
```

Type cast (2)

- Some types can not be cast to some other types. E.g.: float can not be cast to float*.
- A pointer to anything can be converted to a pointer to anything else.
- int can be cast to pointer and vice-versa.

```
int main ()
{
 int i, *pi;
 i = 3;
 pi = (int*)i; // OK
 i = (int)pi;  // OK
 float f, *pf;
 f = 3.3;
 pf = (float*)f; // error
 return 0;
}
```

www.sbVB.com.br

Opaque pointer (void *)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <iostream>
using namespace std;
void fun(void *p,int test)
{
  int *pi = (int*)p;
  int i = *pi;
  VBAssert(i==test); //  VBAssert(*(int*)p==test);
}

int main() {
  int k = 3;
  void *z = (void*)&k;
  fun (z,3); // fun((void*)&k,3);

  unsigned char a[4]; // 4 = sizeof(int)
  // 00000001 00000001 00000000 00000000
  // a[0]     a[1]     a[2]     a[3]
  a[0] = 1;
  a[1] = 1;
  a[2] = 0;
  a[3] = 0;
  fun ((void*)&a,257);
  fun ((void*) a,257);
  return 0;
}
```

Opaque pointer and callBackFunction

```
#include <iostream>
#include "VBLib/VBLib.h"
using namespace std;
using namespace br::com::sbVB::VBLib;

struct Point3D // user class
{
  double x,y,z;
};

void userCallBackFunction(double *array, unsigned s,void *p)
{
  Point3D *p3d = (Point3D *)p;
  array[0] = p3d->x;
  array[1] = p3d->y;
  array[2] = p3d->z;
}
```

Opaque pointer and callBackFunction (2)

```
void call_fun(void (*callBackFun)(double *,
  unsigned,void *))
{
  if (callBackFun == 0)
    return; // do nothing

  const unsigned size = 3;
  double d[size] = { 0, 0, 0};
  Point3D point3D;
  point3D.x = 1.1;
  point3D.y = 2.2;
  point3D.z = 3.3;

  callBackFun(d,size,(void*)&point3D);

  for (unsigned i=0;i<size;i++)
  {
    cout << "d=[" << i << "]=" << d[i] << endl;
  }
}

int main()
{
  call_fun(userCallBackFunction);
  call_fun(0); // invalid callBackFunction
  return 0;
}
```

```
d=[0]=1.1
d=[1]=2.2
d=[2]=3.3
```

www.sbVB.com.br

Opaque pointers and explicit link with dll or so. Example: functions.cpp

```
#include "VBLib/VBLib.h"
using namespace std;
using namespace br::com::sbVB::VBLib;

int main()
{
  VBCrossDLL dll;

  typedef int (*MyFunPtrType)(int);
  MyFunPtrType myFunPtr;

  const char *dllName = "plus_n.dll"; // plus_1 .. plus_5

  // check if can load dll or so, but not load it
  if(!dll.loadOK(dllName))
  {
    cout << "won't be able to load dll or so" << endl;
    return -1;
  }

  // load dll or so. Exception if can not load
  dll.load(dllName);

  const char *exportedNamesDllMustHave[] = {
    "plus_1","plus_2","plus_3","plus_4","plus_5"
  };
  unsigned s = sizeof(exportedNamesDllMustHave)/sizeof(const char *);
```

functions.cpp (2)

```
  // loop to test if the dll has all must have exported names
  for (unsigned i=0; i < s ;i++)
  {
    if(!dll.hasExportedName(exportedNamesDllMustHave[i]))
    {
      cout << "this dll or so don't have all must have exported names" << endl;
      return -1;
    }
  }

  // load a function from dll or so.
  // Exception if can not get it
  myFunPtr = (MyFunPtrType) dll.getAddress("plus_3");
  // use the function from dll or so and check it works
  VBAssert (myFunPtr(6)==9);

  // another test
  myFunPtr = (MyFunPtrType) dll.getAddress("plus_1");
  VBAssert (myFunPtr(4)==5);

  dll.unLoad();

  return 0;

}
```

getAddress returns void*, and by type cast it turns to a function pointer

idem

www.sbVB.com.br

Load all dll's with signature Example: functions_dll.cpp

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
using namespace std;
int main() {
  string ans;
  do {
    const char *exportedNamesDllMustHave[] = {
      "fun"
     ,"fun_name"
    };
    unsigned size =
      sizeof(exportedNamesDllMustHave)/sizeof(const char *);

    VBCrossDLLContainer dllContainer;
    dllContainer.loadDllsWithAllExportedNames
       (exportedNamesDllMustHave,size,".", "dll");

    unsigned nFuns = dllContainer.getNumberOfDlls();

    if (nFuns==0) {
      cout << "No dll's loaded" << endl;
      return 0;
    }

   cout << "There are " << nFuns << " dll's" << endl;
```

these are the names the dll's must have; if a dll has all these names this dll meets the *signature*

dllContainer loads all dll's meeting *signature* from current directory (.)

functions_dll.cpp (2)

```
// display function names and ask for function of choice
int i;
cout << "==== Choose function:" << endl;
for (i=0; i<nFuns; i++) {
  // get fun_name address of i_th dll
  const char *(*fun_name)() =
    (const char *(*)()) dllContainer[i].getAddress("fun_name");

  cout << i << ":" << fun_name() << endl;
}

int funChoice;
cout << "Your choice:";
cin >> funChoice;

if (funChoice >= nFuns)  {
  cout << "invalid choice" << endl;
  return 0;
}
```

functions_dll.cpp (3)

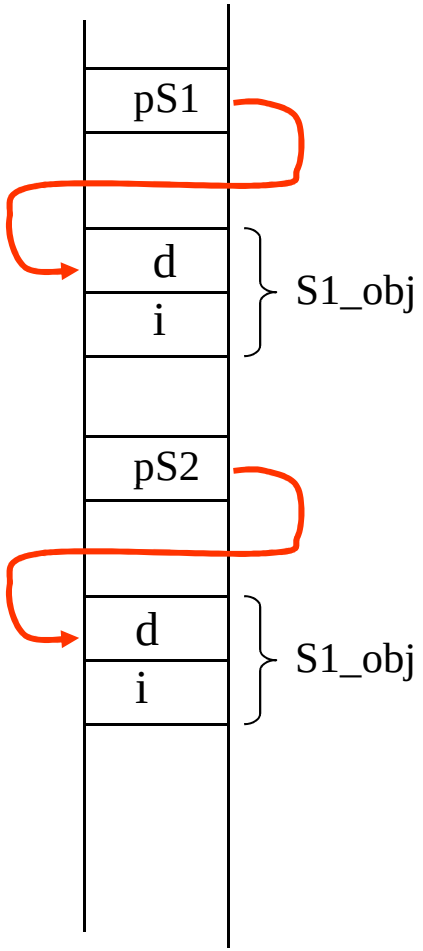
```
    // exercise function
    cout << "==== Function table:" << endl;
    double x_min = 0;
    double x_max = 1;
    double dx = 0.1;
    double x;

    // get fun address of funChoice_th dll
    double(*fun)(double) = // instantiate variable
      (double(*)(double)) // type cast
      dllContainer[funChoice].getAddress("fun");

    for (x=x_min; x<=x_max; x += dx) {
      // call fun from dll
      cout << "f(" << x << ")=" << fun(x) << endl;
    }

    cout << "repeat (y/n):";
    cin >> ans;
  } while (ans[0]!='n'); // with “do”

  return 0;
}
```

Caution: type cast can be insecure

www.sbVB.com.br

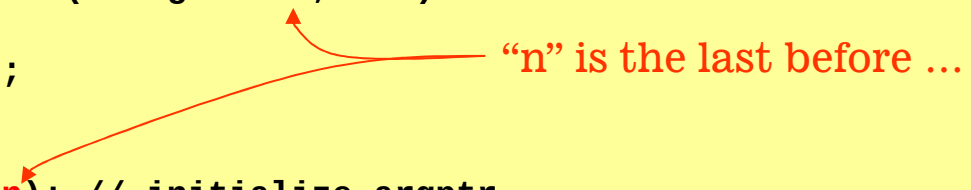
```
struct S1
{
  double d;
  int i;
};
struct S2
{
  struct {
    char ch;
    int k;
  } inStruct;
  float f;
};

int main ()
{
  S1 *pS1 = new S1; // OK
  pS1->d=1.1; // OK
  pS1->i=2; // OK
  S2 *pS2 = (S2*) new S1; // insecure typecast
  pS2->inStruct.ch = 'c';// BUG
  pS2->inStruct.k = 3; // BUG
  pS2->f=4.4; // BUG
  return 0;
}
```

anonymous struct



function with variable number of arguments

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
#include <stdarg.h>
#include <iostream>
using namespace std;
double sum_parameters(unsigned n , ...)
{
  double sum = 0.0;
  double t;
  va_list argptr;
  va_start(argptr,n); // initialize argptr
  // sum the parameters
  for ( ; n ; n--)
  {
    t = va_arg(argptr,double);
    sum += t;
  }
  va_end(argptr); // clean memory
  return sum;
}

int main ()
{
  VBAssert(sum_parameters(2, 1.1, 2.2)==1.1+2.2);
  VBAssert(sum_parameters(3, 1.1, 2.2, 3.3)==1.1+2.2+3.3);
  return 0;
}``
```



www.sbVB.com.br

Projects and global variables

```
// main.cpp

extern double eps;

int main()
{
  double e = eps;
  return 0;
}
```

```
// MyFunctions.cpp

double eps = 1e-8;
```

www.sbVB.com.br

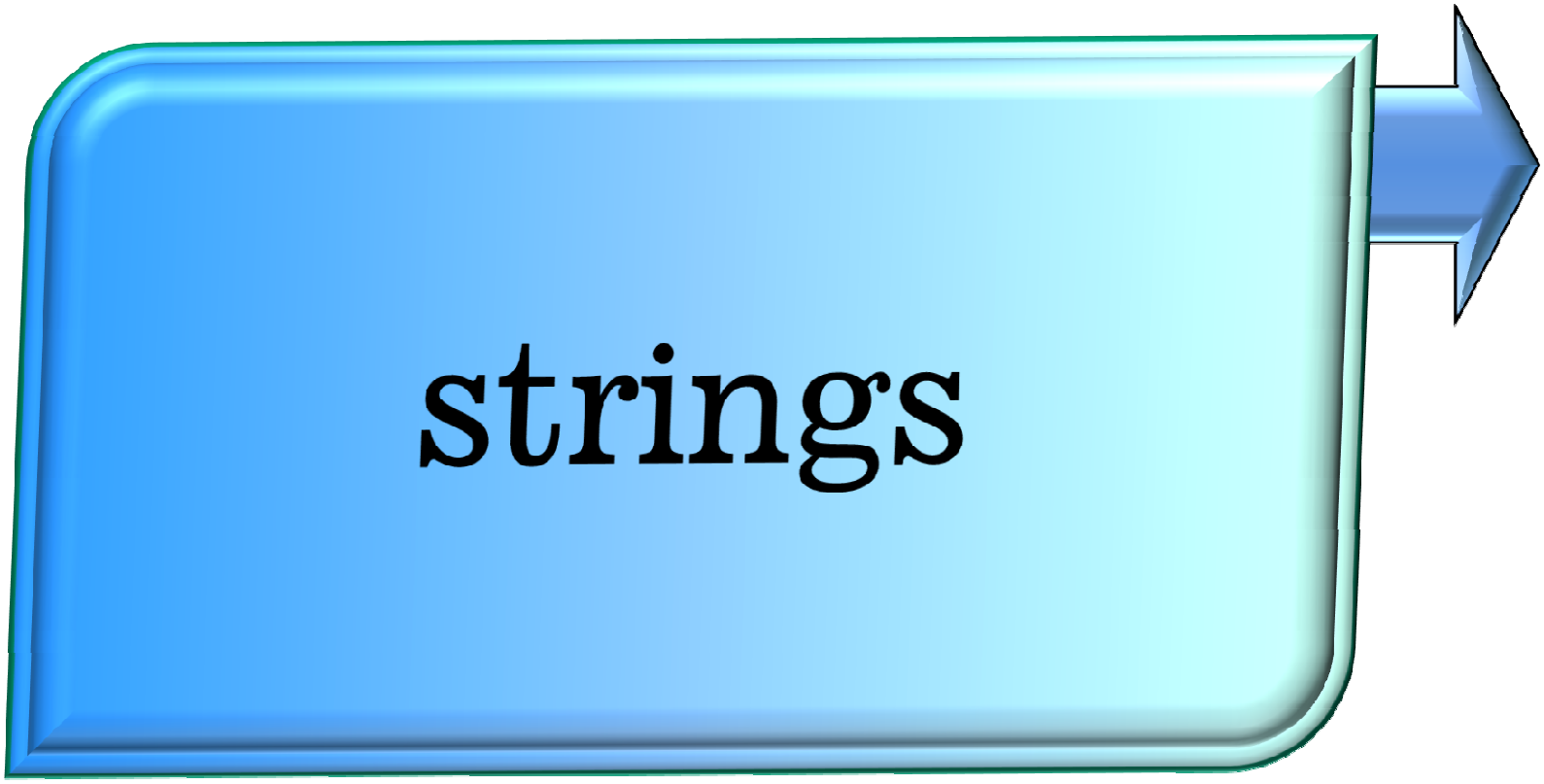
Projects and global variables (2)

```
// MyFunctions.h
#ifdef DEFINE_GLOBAL_VARIABLES_HERE
  #define EXTERN
#else
 #define EXTERN extern
#endif

EXTERN double eps;
```

```
// main.cpp
#include "MyFunctions.h"

int main()
{
  double e = eps;
  return 0;
}
```

```
// MyFunctions.cpp
#define DEFINE_GLOBAL_VARIABLES_HERE
#include "MyFunctions.h"
```

Handling strings

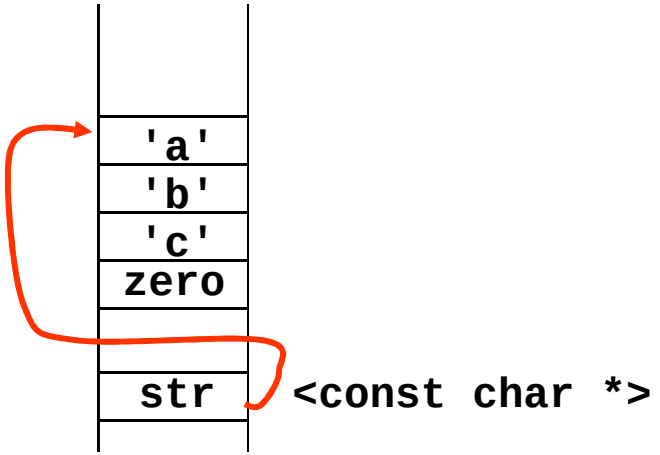


Handling strings

- In C/C++, any sequence between quotes ("sequence") returns a const char *.
- The sequence itself is somewhere in the memory, and the pointer points to the beginning of the sequence.
- Every char sequence terminates with char zero (not char '0').

"xyz" returns const char *

```
int main()
{
  const char * str = "abc";
  return 0;
}
```



Handling strings (2)

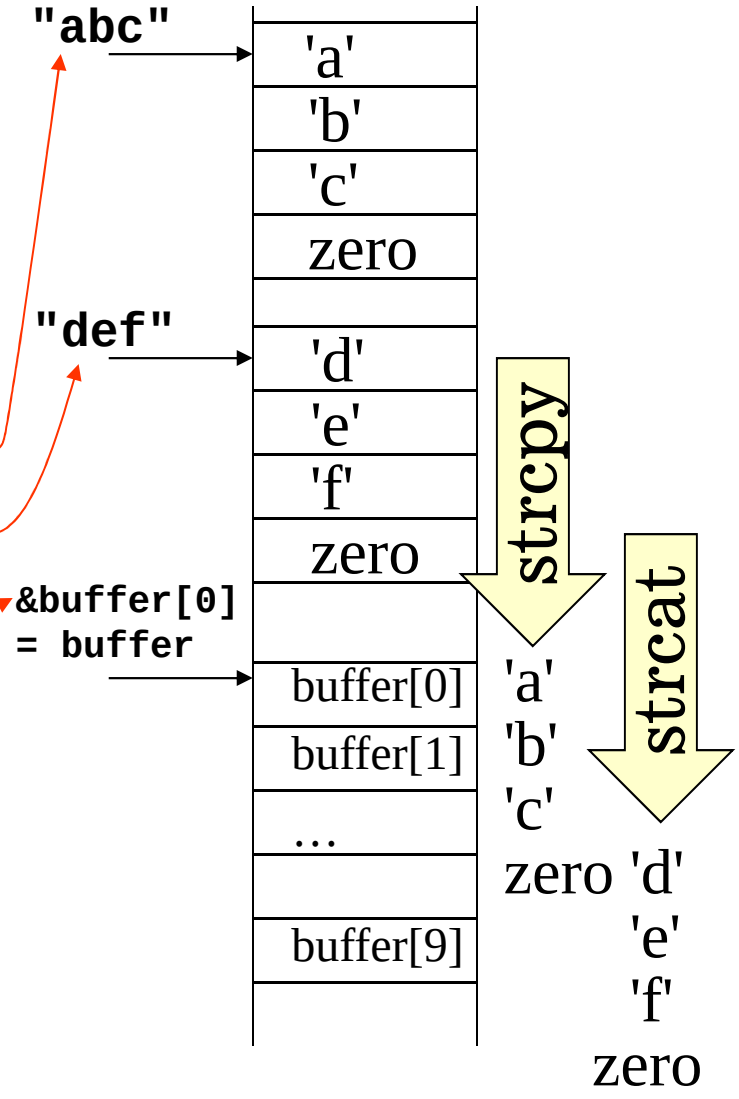
- C style
  - Array of char (static or dynamic)
- C++ style
  - String class
- In almost all practical cases, you should treat strings in the C++ style
- Should a bug happen, the program behaviour is random. That's worse than freezing the computer (because in this case, it becomes immediate to identify the bug).

Strings C style
strcpy, strcat

www.sbVB.com.br

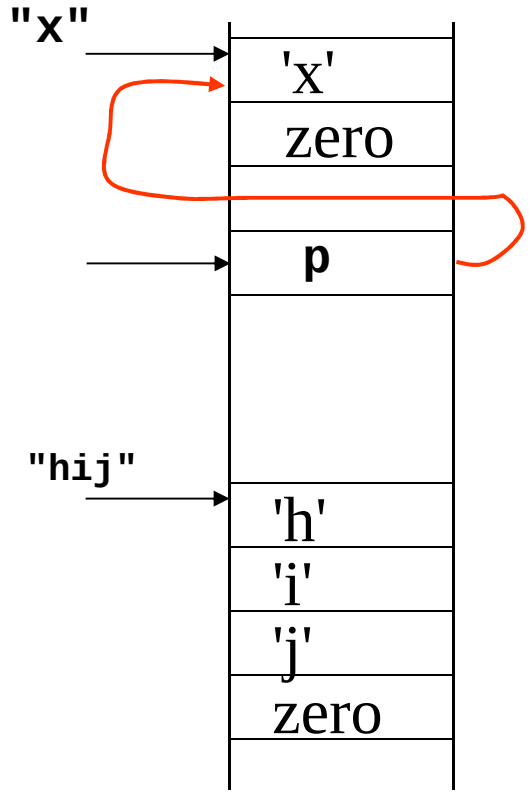
```
#include <iostream>
#include <string>
using namespace std;
// strcpy(char *, const char *);
// strcat(char *, const char *);
int main()
{
  char buffer[10];
  strcpy(buffer,"abc"); // OK
  strcat(buffer,"def"); // OK
  cout << buffer << endl;
  return 0;
}
```

```
abcdef
```



String bug type 1
Non-initialized buffer

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  char buffer[50];
  char *p = "x";
  // p = buffer;
  strcpy(p,"hij"); // BUG
  return 0;
}
```



String bug type 2
Buffer too small

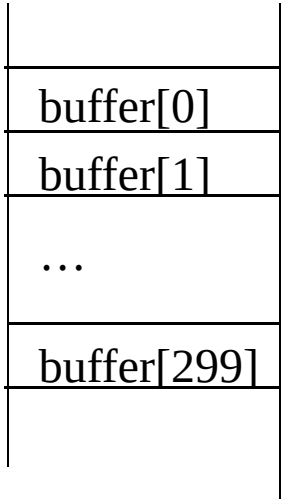
- For a program not to have a bug of type 2, the buffer must be always bigger or equal to the biggest string that happen to be stored there (including the terminating char zero).
- A experienced developer knows that a software component end up used in a situation not considered at first. In the new situation, should the buffer be too small, than the but type 2 happens.
- If the developer uses huge sizes for string buffers, the probability for bug type 2 diminishes (but does not get zero). But in that case, it will be a waste of memory.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  char buffer[3]; // small buffer
  strcpy(buffer,"abc");
  strcat(buffer,"def");
  cout << buffer << endl;
  return 0;
}
```

String bug type 3
String as return of function

```
#include <iostream>
#include <string>
using namespace std;
char *htmlPath(const char *userName)
{
  const char *path = "/home/usr/";
  const char *html = "/public_html";
  static char buffer[300];
  strcpy(buffer,path);
  strcat(buffer,userName);
  strcat(buffer,html);
  return buffer; // BUG
  // returning reference to local variable
}

int main()
{
  char *completePath = htmlPath("fred");
  cout << completePath << endl;
  return 0;
}
```

```
/home/usr/fred/public_html
```



www.sbVB.com.br

Treating strings in a safe way, using C++ style

- Use string classes, and not array of char.
- There are many string classes to choose from:
  - ::std::string
    pro: it is standard;
    against: you can't easily change the source, so it is not easy to add new features.
  - VBString
    has improved features, when compared to the standard string.
    Use VBString downloading VBLib. Perfect for use with the sbVB C++ course
  - CString
    pro: perfect for Windows, VisualC and MFC
    against: does not work with unix/linux.
  - wxString
    perfect for using with wxWidgets
  - SAString
    perfect for using with SQLAPI++.
- It is no problem to use more than one string class at the same time.

VBString

- To use VBString, download VBLib. Add VBLib.lib to your project and the include folder to the header path.
- To use it is a good practice of how to use external libraries.
- VBLib is written in cross-plataform, C++ (license LGPL).

VBString (2)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

int main()
{
  VBString a;
  a = "12345 ";
  VBString b = a + "33";
  VBString c = b;

  VBAssert(c == b);
  VBAssert(c >= b);
  VBAssert(!(c < b));
  VBAssert(c <= b);
  VBAssert(a != b);
  VBAssert(b=="12345 33");

  // common error:
  // a = "abc" + "def"; // can not add 2 pointers directly
  // use instead
  a = "abc";  a += "def";
  // or a = "abc" + VBString("def");
  VBAssert(a=="abcdef");

  return 0;
}
```

www.sbVB.com.br

VBString (3)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
using namespace std;
int main()
{
  VBString a;
  a = "123";
  VBAssert(a=="123");
  a = "456" + a;
  VBAssert(a=="456123");
  a = a + 78;  // int
  VBAssert(a=="45612378");
  a.setDecs(3);
  a = a + 0.12345;  // double
  VBAssert(a=="456123780.123");
  a = "abc";
  a += 12; // int
  VBAssert(a=="abc12");
  a = "abc";
  a += 12.12345; // double
  VBAssert(a=="abc12.123");
  a = "abc";
  a += "12"; // const char *
  VBAssert(a=="abc12");

  return 0;
}
```

```
#include <sstream> // istringstream
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
using namespace std;
int main()
{
  VBString a;
  cout << "Enter anything:";
  cin >> a;
  cout << "a=" << a << endl;

  istringstream iss;
  iss.str("something");
  iss >> a;
  VBAssert(a=="something");

  return 0;
}
```

```
Enter anything:abc
a=abc
```

www.sbVB.com.br

VBString (4)
beforeFind, afterFind, replace

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

int main()
{
  VBString a,b;
  a = "123456789";
  b = a.beforeFind("56"); // b = "1234";
  VBAssert(b=="1234");
  b = a.afterFind("xx"); // b = ""; if string not found
  VBAssert(b=="");

  a = "01234567890123456789012345678901234567890123456789";
  VBString find = "34";
  VBString replace = "abc";
  a.replace(find,replace); // (changes a)
  VBAssert(a=="012abc56789012abc56789012abc56789012abc56789");

  return 0;
}
```

VBString (5)
replace, strtok

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

int main()
{
  VBString a,b;
  //   0   1   2   3   4   5    6   (tok)
  a = "abc,def,ghi,123,456,7890,yyy";
  char tok = ',';
  VBAssert(a.strtok(tok,0)=="abc");
  VBAssert(a.strtok(tok,1)=="def");
  VBAssert(a.strtok(tok,5)=="7890");
  bool found;
  b = a.strtok(tok,3,found); // found is a return value
  VBAssert(found);
  VBAssert(b=="123");

  return 0;
}
```

www.sbVB.com.br

VBString (6)

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;
int main()
{
  VBString a,b;
  a = "01234567890123456789012345678 9";
  b = a.strAfterPosition(3);
  VBAssert(b=="3456789012345678901234567 89");
  b = a.strUntilPosition(1);
  VBAssert(b=="01");
  b = a.strInside(1,5);
  VBAssert(b=="12345");
  int pos, occurrence = 2;
  char ch = '6';
  pos = a.strtokpos(ch,occurrence);
  VBAssert(pos==16);
  ch = a.getChar(pos); // equivalent to ch = a[pos];
  VBAssert(ch=='6');
  return 0;
}
```

VBString (7)

VBString conversion to const char*

```
#include <string> // strcpy
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

void myOldCFun(const char *str)
{
  printf("myOldCFun:%s\n", str);
}

void myOldCFun2(char *str)
{
  printf("myOldCFun2:%s\n", str);
}

int main()
{
  VBString a = "abc";
  const char *a_str = a.c_str();  myOldFun(a_str);
  myOldCFun(a); // automatic conversion of VBString to const char*
  myOldCFun(a.c_str()); // explicit conversion to const char *
  myOldCFun2(a); // error. Can't convert VBString to char*
  char buffer[300];
  strcpy(buffer,a); // automatic conversion of VBString to const char*
  VBAssert(VBString("abc")==buffer);
  myOldCFun2(buffer);
  return 0;
}
```

```
myOldCFun:abc
myOldCFun2:abc
```

www.sbVB.com.br

VBString (8)
function returning a string

```
#include "VBLib/VBLib.h"
using namespace br::com::sbVB::VBLib;

VBString htmlPath(VBString userName)
{
  const char *path = "/home/usr/";
  const char *html = "/public_html";
  VBString ret;
  ret = path;
  ret += userName;
  ret += html;
  return ret;
}

int main()
{
  VBString completePath = htmlPath("fred");
  VBAssert(completePath=="/home/usr/fred/public_html");
  return 0;
}
```

Object oriented software development

Why C++ (and OO) is better than C (procedural) ?

www.sbVB.com.br

Bad programming practice!
2 pieces of code that are related!

- Easier to learn, harder to make mistakes; see example below

C

```
#include <stdio.h>
int main() {
 float myf = 1;
 printf("myf=%f\n",myf);
 return 0;
}
```

```
#include <stdio.h>
int main() {
 int myf = 1;
 printf("myf=%d\n",myf);
 return 0;
}
```

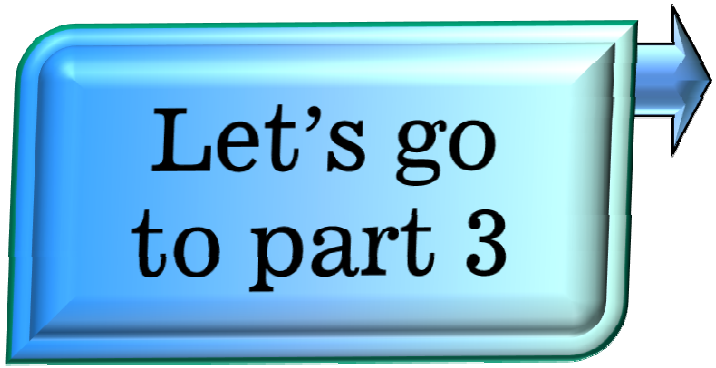
C++

```
#include <iostream>
using namespace std;
int main() {
 float myf = 1;
 cout << "myf=" << myf << endl;
 return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {
 int myf = 1;
 cout << "myf=" << myf << endl;
 return 0;
}
```

Why C++ (and OO) is better than C (procedural) ? (2)

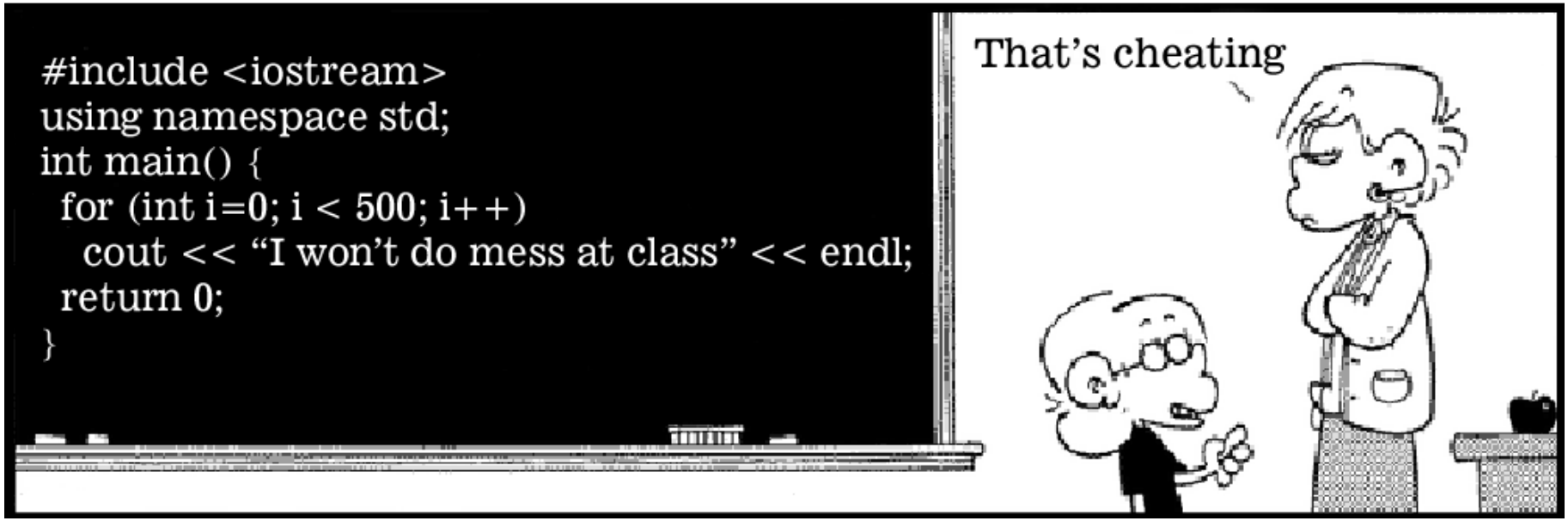
- The standard library of C++ is developed using the OO concepts
  - The classes the developer does “fit better” to the standard library of C++ than would do to C.
  - Since the features of C standard library were all inserted in C++, it is not worth to invest time to learn the C standard library. Better to use the time to learn the C++ standard library.
- A C++ source code, developed using OO, is very similar to a code of java.



Humor

Humor

- conflict of generations . . .



32 / 64 bits

- http://code.google.com/p/msinttypes/