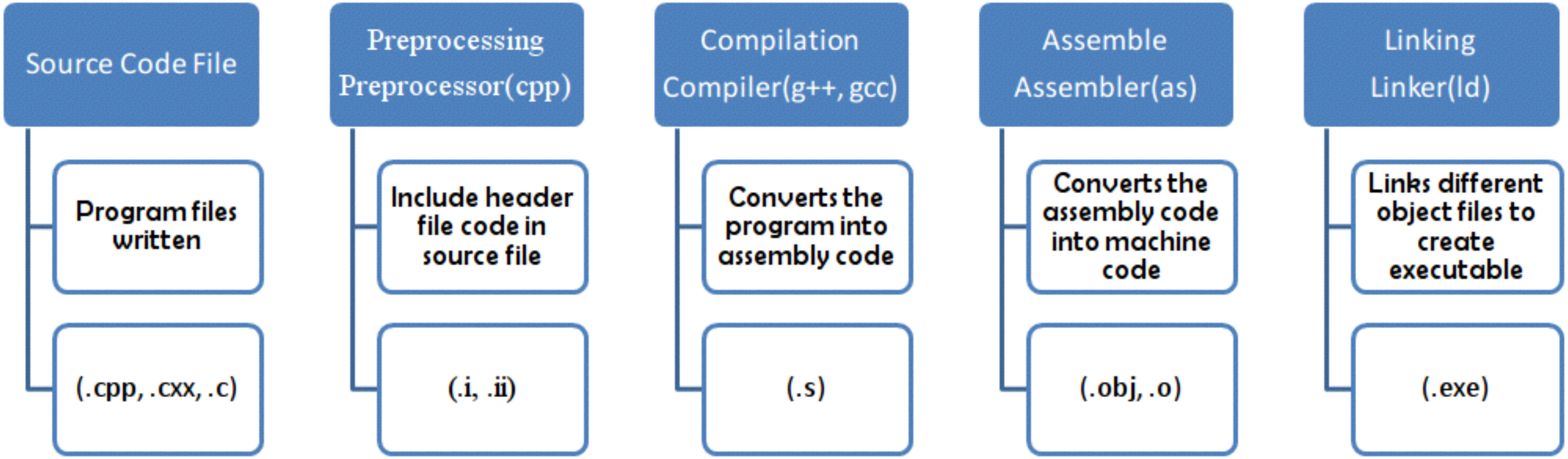
Program segmenting; preprocessor, processor, assembler and linking; libraries

code compilation

- how is source code transformed into executable code?



(cplusplus.com)

The preprocessor

- generates a single file (replaces header files) which is passed to the compiler
- 3 different usages : directives, macros and constants; for instance

```
#define VERSION 1
```

- how to see the output of the preprocessor?

```
int main() {
  if (VERSION == 1) {
    // OK
  }
  else {
    // NOT OK
  }
  return 0;
}
```

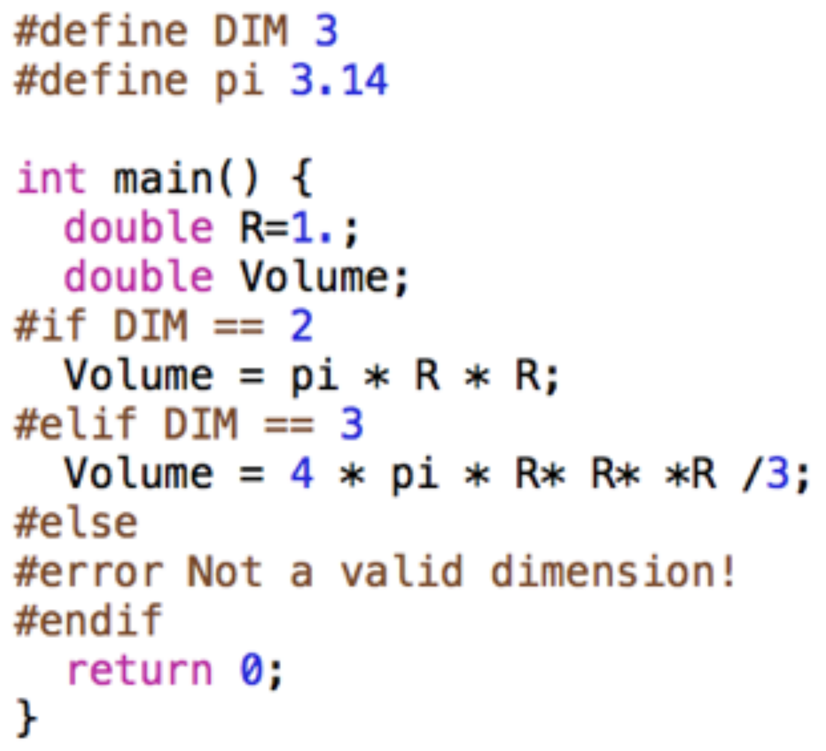
- compile preproc1.cpp with the -E flag

The preprocessor

- typical structures: #define, #undef, #error, #if, #elif, #ifdef, #ifndef, #endif, #include

- more structure can be done as follows

(preproc2.cpp)



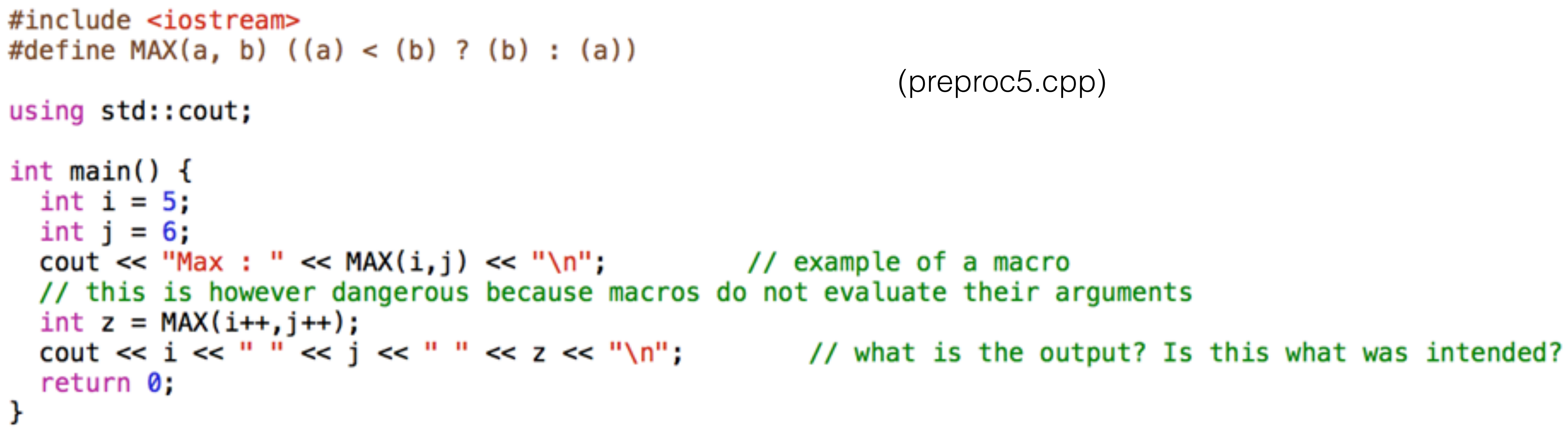
```
#define DIM 3
#define pi 3.14

int main() {
  double R=1.;
  double Volume;
#if DIM == 2
  Volume = pi * R * R;
#elif DIM == 3
  Volume = 4 * pi * R* R* *R /3;
#else
#error Not a valid dimension!
#endif
  return 0;
}
```

The compiler well never see the source code other than for DIM == 3 (so it will also not check for errors in other parts)

The preprocessor

Macros can be used as follows:

(preproc5.cpp)



```cpp
#include <iostream>
#define MAX(a, b) ((a) < (b) ? (b) : (a))

using std::cout;

int main() {
  int i = 5;
  int j = 6;
  cout << "Max : " << MAX(i,j) << "\n";         // example of a macro
  // this is however dangerous because macros do not evaluate their arguments
  int z = MAX(i++,j++);
  cout << i << " " << j << " " << z << "\n";        // what is the output? Is this what was intended?
  return 0;
}
```

In our opinion macros should be avoided in C++ whenever possible.

The preprocessor

When using a header file

```
// preproc3.h header file

struct cmpl {
  double real_;
  double imag_;
};
```

we can include it in the main program as follows

```
// preproc3.cpp

#include "preproc3.h"

int main() {
  cmpl MycomplexNumber;
}
```

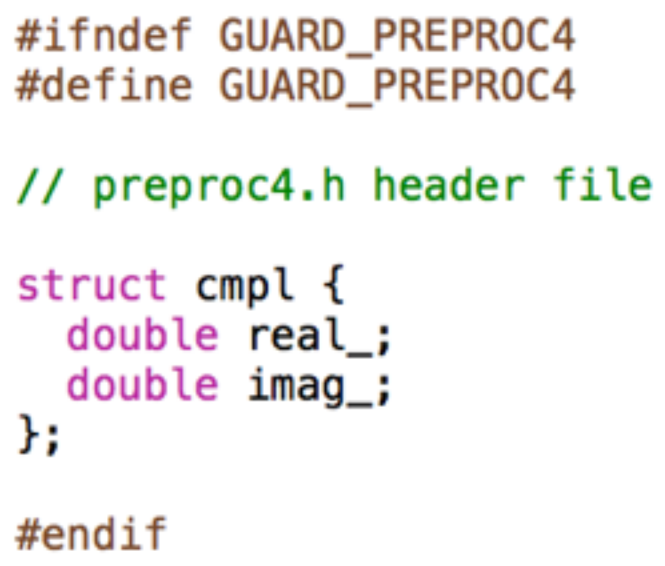
what does the precompiler do when typing g++ -E preproc3.cpp?

what happens when we write

```
#include "preproc3.h"
#include "preproc3.h"
```

The preprocessor

Such a situation occurs frequently without noticing. In order to avoid this one introduces guards



```
#ifndef GUARD_PREPROC4
#define GUARD_PREPROC4

// preproc4.h header file

struct cmpl {
  double real_;
  double imag_;
};

#endif
```

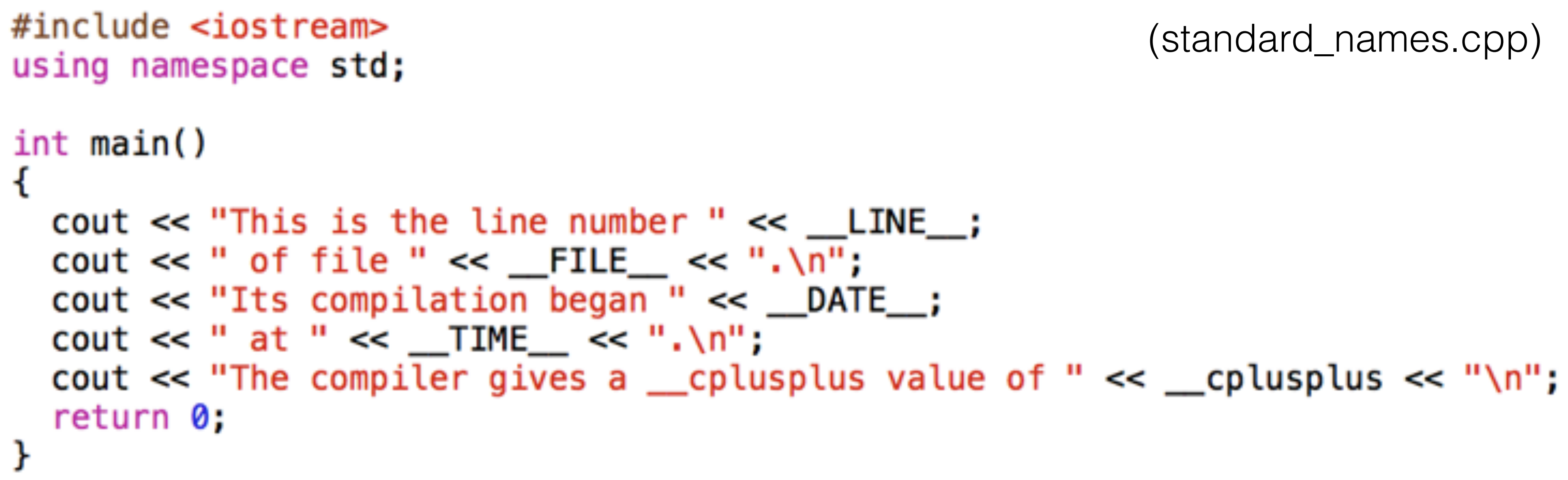
Check the output of the precompiler of preproc4.cpp

Such constructions are often used for version checking, compiler dependent issues and further options of the program

The preprocessor

There exists other directives (#line, #pragma) that we do not discuss here, as well as some other predefined names, eg

(standard_names.cpp)



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

int main()
{
  cout << "This is the line number " << __LINE__;
  cout << " of file " << __FILE__ << ".\n";
  cout << "Its compilation began " << __DATE__;
  cout << " at " << __TIME__ << ".\n";
  cout << "The compiler gives a __cplusplus value of " << __cplusplus << "\n";
  return 0;
}
```

The compiler

Consider an implementation for an alternative complex number structure

```
// cmpl.h
struct cmpl {
  double real_;
  double imag_;
};

double get_modulus_squared(const cmpl&);
double get_arg(const cmpl&);
```

The functions are worked out in cmpl.cpp

```
double get_modulus_squared(const cmpl& c) {
  return c.real_ * c.real_ + c.imag_ * c.imag_;
}

double get_arg(const cmpl& c) {
  // we want a value between [0, 2pi[, there is a branch cut on the positive x-axis which should be excluded
  assert (!(c.real_ == 0 && c.imag_ == 0));
  assert (!(c.imag_ == 0 && c.real_ > 0));
  double phase = (c.imag_ < 0 ? 2*pi : 0);
  return std::atan2(c.imag_ , c.real_) + phase; // atan2 returns a value between [-pi, pi[
}
```

The main() function is however in prog.cpp

we can compile with g++ -c cmpl.cpp , which will construct an object (.o) file
we also need to compile main: g++ -c prog.cpp

The compiler

linking can be done with g++ -o prog_cmpl prog.o cmpl.o

due to the choice of our branch cut, we cannot accept arguments on the real positive axis. A complex number (1,0) will produce an error message of the type

```
Assertion failed: (!(imag_ == 0 && real_ > 0)), function get_arg, file cmpl.cpp, line 12.
Abort trap: 6
```

this is due to assert defined in <cassert>

compare the output of g++ -E cmpl.cpp with g++ -E -DNDEBUG cmpl.cpp

you will see that the asserts have been replaced by voids.

the -D *var* flag is equivalent to a #define *var* , hence the no-debug code only is passed on to the compiler. Likewise, the -U *var* flag is equivalent to a #undef *var*

after debugging the code and making sure the code is fast, we can decide to switch off these asserts (which slow down the code). For more sophisticated error handling, see later in the course.

The compiler

The compiler provides a code which can be directly translated into assembler code. Its contents can be read with

```
g++ -c -S cmpl.cpp
```

which generates a file cmpl.s. You can also add -fverbose-asm for extra comments. Can you make sense of its contents?

compile ex_inline.cpp with different optimization levels (-g, -O0 , -O3)

```
Lode.Pollet@th-sv-clhead:~/Temp$ g++ -g ex_inline.cpp
Lode.Pollet@th-sv-clhead:~/Temp$ objdump -S a.out > out_debug_version
Lode.Pollet@th-sv-clhead:~/Temp$ vi out_debug_version
```

Can you observe automatic inlining?

A function call is expensive and can can take several hundred CPU cycles : function arguments may need to be stored in memory and read, and all pipelines might be stopped.

The compiler may try to replace ‘easy’ functions directly with the function body when optimizing aggressively . A better solution is function inlining (write inline before the function definition), which for class functions can also be achieved by putting the function body inside the class header

Libraries

in cmpl2.cpp we define the addition of 2 complex numbers, through which we want to extend the functionality of our complex number structure

```
// cmpl2.cpp
#include "cmpl.h"

cmpl addition(const cmpl& c1, const cmpl& c2) {
  cmpl c;
  c.real_ = c1.real_ + c2.real_;
  c.imag_ = c1.imag_ + c2.imag_;
  return c;
}
```

we also compile it with g++ -c cmpl2.cpp

important: we must add the declaration cmpl addition(const cmpl&, const cmpl&) to the end of the header file cmpl.h

we want to use our structure for complex numbers repeatedly (or share with other team members), including the addition function, and turn it into a single library for easier use

```
$ ar cr libcmpl.a cmpl.o cmpl2.o
```

the name of the library must begin with lib and end with .a (in windows: .lib)

we can view its object files with

```
$ ar t libcmpl.a
```

further possibilities to manipulate the library exist (extract, add, delete, …); see the manual of the ar command

Libraries

ranlib can be used to make a table of contents (today most likely not needed any more: ranlib is incorporated into ar)

The main program does not know the addition function. The linker will look into the libraries for such a function:

```
g++ -o cmpl_main main.cpp -L. -lcmpl
```

the -L option tells the linker in which directories to look for libraries — in this case the current directory (.) but we could write any -L/path/ such as /usr/local/lib for instance.

the -l option tells the linker the name of the library; it will translate -l*name* into libname.a

This is for statically linked libraries which become part of the implementation; for dynamically linked libraries (.so — shared object or .dll in windows, or .dylib in OSX) which are linked at runtime we refer to the literature. To see to which libraries the program is linked, use ldd (linux) or otool -L (OSX)

```
$ otool -L cmpl_main
cmpl_main:
   /usr/lib/libc++.1.dylib (compatibility version 1.0.0, current version 120.1.0)
   /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1226.10.1)
```

here we can see which version of the standard library was used (clang compiler); for the standard libraries we did not have to specify anything extra because the linker knows where to look for it

Shared vs static libraries

shared

- all code in the library
- referenced at run-time
- reduces amount of code copied in each application
- + : keeps binary small
- - : slower because of execution of function calls and loading cost

static

- all code in the library
- referenced at compile time
- you don’t need a copy of the library
- - : larger binary files
- + : faster (no loading cost)

Lapack, BLAS, FFT

In numerical work, it is very common to link with external libraries. Here are a few that are often used in C++:

- Boost libraries : often used libraries that may become part of the standard or TR1 one day (graph library, special functions, concurrency, template meta programming, parsing, filesystem, …)
- Blitz++ : convenient for efficient tiny vector and matrix operations (however little activity in the last 5 years)
- Armadillo : matrix, vector, decompositions, factorizations, equation solvers, …
- Eigen : for linear algebra, matrix and vector operations, numerical solvers, …
- FFTW : Fast Fourier Transform (the fastest one in the West)
- Blas and Lapack : Basic Linear Algebra Subprograms and Linear Algebra Package — originally written in Fortran
- MKL : specifically optimized Blas and Lapack functions for Intel compilers

Libraries are optimized for their tasks. They will typically perform the same task many orders of magnitude more efficiently than a naive implementation.

BLAS and LAPACK

http://www.netlib.org/blas/

3 levels:

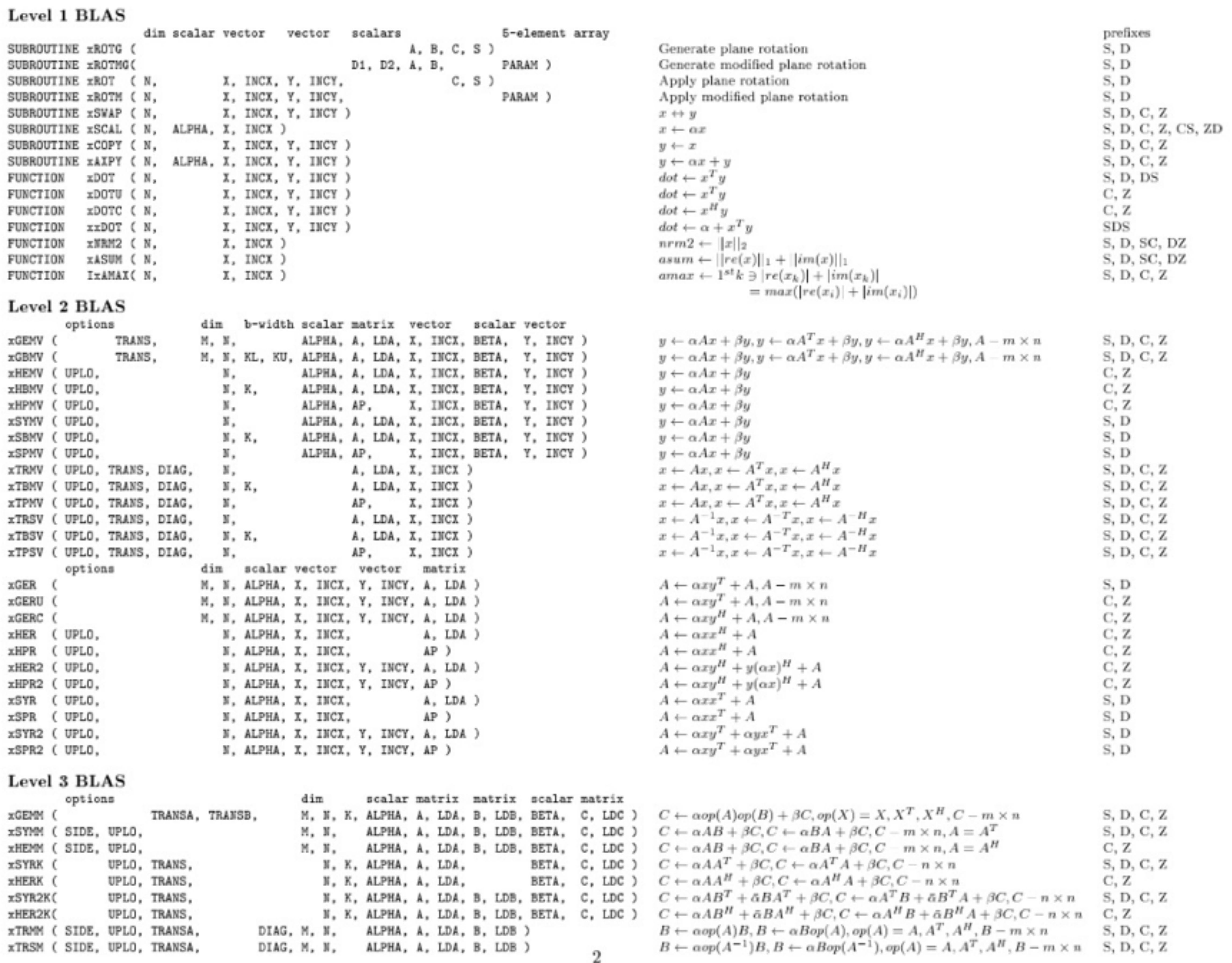
1. scalar, scalar-vector and vector-vector (eg axpy)
2. matrix-vector
3. matrix-matrix (eg DGEMM)

There are machine specific optimized BLAS libraries
(eg: AMD : ACML, Apple : Accelerate, Intel : MKL)

naming convention: DGEMM

x = S, D, C, Z

Blas



Level 1 BLAS

		prefixes
`dim scalar vector vector scalars 5-element array`		
`SUBROUTINE xROTG ( A, B, C, S )`	Generate plane rotation	S, D
`SUBROUTINE xROTMG( D1, D2, A, B, PARAM )`	Generate modified plane rotation	S, D
`SUBROUTINE xROT ( N, X, INCX, Y, INCY, C, S )`	Apply plane rotation	S, D
`SUBROUTINE xROTM ( N, X, INCX, Y, INCY, PARAM )`	Apply modified plane rotation	S, D
`SUBROUTINE xSWAP ( N, X, INCX, Y, INCY )`	$x \leftrightarrow y$	S, D, C, Z
`SUBROUTINE xSCAL ( N, ALPHA, X, INCX )`	$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
`SUBROUTINE xCOPY ( N, X, INCX, Y, INCY )`	$y \leftarrow x$	S, D, C, Z
`SUBROUTINE xAXPY ( N, ALPHA, X, INCX, Y, INCY )`	$y \leftarrow \alpha x + y$	S, D, C, Z
`FUNCTION xDOT ( N, X, INCX, Y, INCY )`	$dot \leftarrow x^T y$	S, D, DS
`FUNCTION xDOTU ( N, X, INCX, Y, INCY )`	$dot \leftarrow x^T y$	C, Z
`FUNCTION xDOTC ( N, X, INCX, Y, INCY )`	$dot \leftarrow x^H y$	C, Z
`FUNCTION xxDOT ( N, X, INCX, Y, INCY )`	$dot \leftarrow \alpha + x^T y$	SDS
`FUNCTION xNRM2 ( N, X, INCX )`	$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ
`FUNCTION xASUM ( N, X, INCX )`	$asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$	S, D, SC, DZ
`FUNCTION IxAMAX( N, X, INCX )`	$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

`options dim b-width scalar matrix vector scalar vector`		
`xGEMV ( TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
`xGBMV ( TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
`xHEMV ( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	C, Z
`xHBMV ( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	C, Z
`xHPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	C, Z
`xSYMV ( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	S, D
`xSBMV ( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	S, D
`xSPMV ( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )`	$y \leftarrow \alpha Ax + \beta y$	S, D
`xTRMV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )`	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
`xTBMV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )`	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
`xTPMV ( UPLO, TRANS, DIAG, N, AP, X, INCX )`	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
`xTRSV ( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )`	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
`xTBSV ( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )`	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
`xTPSV ( UPLO, TRANS, DIAG, N, AP, X, INCX )`	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
`options dim scalar vector vector matrix`		
`xGER ( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )`	$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
`xGERU ( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )`	$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
`xGERC ( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )`	$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
`xHER ( UPLO, N, ALPHA, X, INCX, A, LDA )`	$A \leftarrow \alpha xx^H + A$	C, Z
`xHPR ( UPLO, N, ALPHA, X, INCX, AP )`	$A \leftarrow \alpha xx^H + A$	C, Z
`xHER2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA )`	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
`xHPR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )`	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
`xSYR ( UPLO, N, ALPHA, X, INCX, A, LDA )`	$A \leftarrow \alpha xx^T + A$	S, D
`xSPR ( UPLO, N, ALPHA, X, INCX, AP )`	$A \leftarrow \alpha xx^T + A$	S, D
`xSYR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA )`	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
`xSPR2 ( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )`	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

Level 3 BLAS

`options dim scalar matrix matrix scalar matrix`		
`xGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC )`	$C \leftarrow \alpha op(A)op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
`xSYMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )`	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S, D, C, Z
`xHEMM ( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )`	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
`xSYRK ( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )`	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
`xHERK ( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )`	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
`xSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC )`	$C \leftarrow \alpha AB^T + \bar{\alpha} BA^T + \beta C, C \leftarrow \alpha A^T B + \bar{\alpha} B^T A + \beta C, C - n \times n$	S, D, C, Z
`xHER2K( UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC )`	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C, Z
`xTRMM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )`	$B \leftarrow \alpha op(A)B, B \leftarrow \alpha B op(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
`xTRSM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )`	$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha B op(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

2

BLAS and LAPACK

http://www.netlib.org/lapack/

LAPACK is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision…

Linking with C libraries

Since Blas and Lapack are written in Fortran (or automatically translated into C by f2c), how do we tell the compiler and linker that we are using libraries written in C (instead of C++)?

answer: with the keyword "extern"

the example below shows how to obtain the norm of a vector using a call to the BLAS function dnrm2:

```
#include <iostream>

extern "C" double dnrm2_(int* n, double* x, int* incx);

int main() {
  const int N = 10;
  double vec[N];
  for (unsigned int n=0; n < N;n++) vec[n] = 1.0;
  int vecsize = N;
  int incx = 1;
  std::cout << "norm of vector : " << dnrm2_(&vecsize, vec, &incx) << "\n";
  return 0;
}
```

note in particular the underscore following the function name (dnrm2_). Fortran does not know pass by value; hence the pointers for every argument in dnrm2_(int*, double*,int*). In the main program we must take the address of every variable used as function argument.

Often there are C++ variants of these libraries available. The documentation makes then clear how they should be used.

column-major and row-major

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

row major (C, C++, Python):

offset = (row * NUMCOLS) + column

```
int A[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

row = offset / NUMCOLS
column = offset % NUMCOLS

contiguously in memory as

```
1  2  3  4  5  6
```

column major (Fortran, Matlab):

contiguously in memory as

```
1  4  2  5  3  6
```

offset = row + column*NUMROWS

this generalizes to higher dimensions

When calling libraries (BLAS, Lapack, FFTW, …), care needs to be taken with respect to this issue

contract programming

Good software design requires that you properly document your libraries with the following information:

- semantics: how to use the libraries, what does it do?
- invariants: variables guaranteed not to change (eg, size of an array)
- pre-conditions: the conditions that must hold when using the library, for instance, range checks, overflow/underflow, or in our example for the arg function: no complex numbers on the real positive axis. Asserts are allowed to make sure that the pre-condition is met.
- post-conditions: guarantees when the pre-conditions are met, for instance, the polar representation of our complex class agrees within machine precision with the decimal one. Asserts are allowed to make sure that the post-condition is met.
- dependencies : other libraries and classes on which the library relies, eg the standard library, BLAS/LAPACK, …
- exception guarantees : see later
- for templates: type requirements (see later)

It is one of the topics that could become part of the C++17 standard

provenance

Provenance means metadata/trace information about a computation.

- every scientific result should be reproducible
- imagine the PhD student leaving the group and a few years later questions arise about the results
- what information should we store in order to make a scientific simulation reproducible?
- only table of results? published figures?
- or only the results leading to the final answer? or all obtained results?
- source code?
- compiler and library versions, operating system, hardware information?
- date and time? weather information?
- how the fitting was done in obtaining the final figure? (type of fit, type of curve, fitting interval, …) ?

This is still a topic of active research in computer science