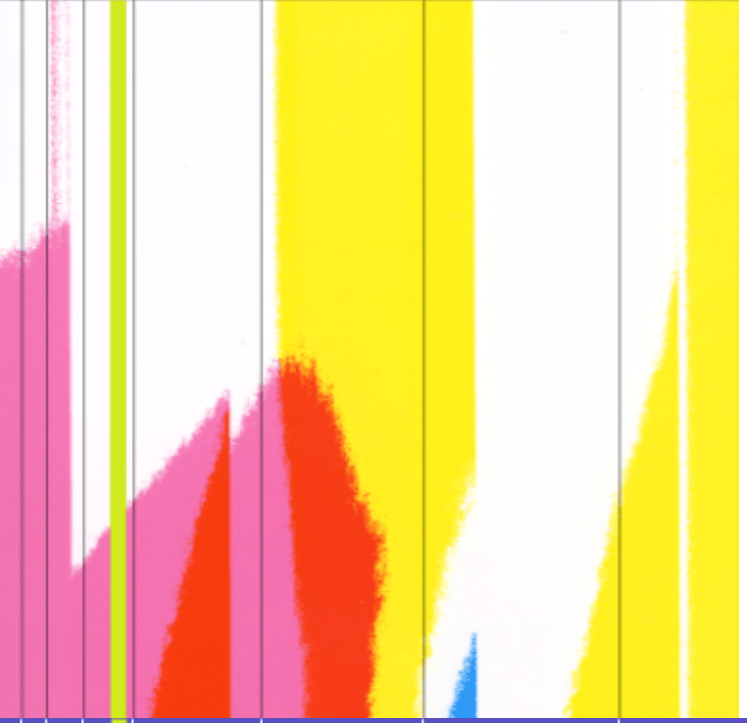


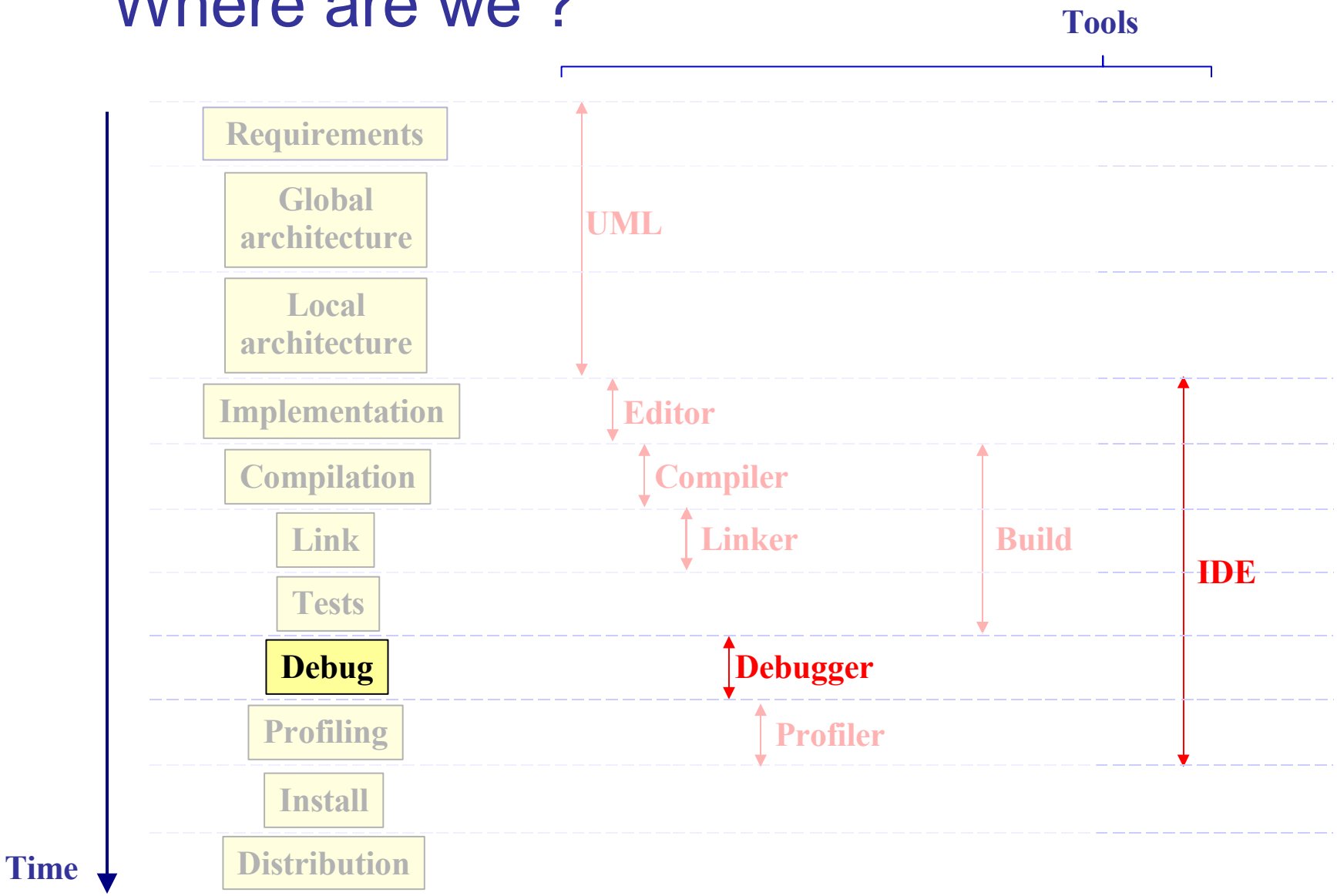
Debugging

Erwan Demairy – Dream

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA
SOPHIA ANTIPOLIS

Where are we ?


Tools
Requirements
Global architecture
Local architecture
Implementation
Compilation
Link
Tests
Debug
Profiling
Install
Distribution
Time
UML
Editor
Compiler
Linker
Build
IDE
Debugger
Profiler

Overview

Introduction

Debugging Rules

- Is it really a bug ?
- Reproduce and document the bug
- Dichotomic search
- Ask for help

Tools

- Static checking
- Assert statements
- Print statements
- Log files
- Debuggers
- Memory checkers

Demonstrations

Bug = « something is wrong »

Bug causing crash

- Memory mismanagement

Bug causing unexpected result

- Right algorithm badly written
- Bad algorithm

Some common causes

Analyse

- Behaviour of the written code
  float x = 3/2;
- Behaviour of the libraries
  strcpy(static_string, ¨ blablabla ¨);
- Behaviour of the compilation tools
  Failure in linking, ...
- Behaviour of the OS, or of the programs use
  Several simultaneous signals, ...
- Hardware Failures
  Hard disk sectors broken
- ...

Debugging Rules

1. Document THE problem (how to reproduce the bug)
   - BUGS file, Bugzilla, Forge bug-tracker, ...
2. Clarify whether it is really a bug
   - Is it a requirements or design mistake?
     E.g.: an interrupt-driven program
   - Bad understanding of the usage?
     E.g.: use your washing-machine to wash your dishes
   - Transient problems that are not considered by the program?
     E.g.: network failure
3. Estimate the priority and the time needed to find the bug
4. If your schedule allows it or the bug-priority requires it:
   - Write a non-regression test
   - Find the cause(s) of the bug

Find the cause of the bug: where is it?

Dichotomy on the program

- Divide-and-conquer
  - Starting point of the interval = the program's start
  - Ending point = when the program is obviously wrong (crash or incorrect behaviour or result).
  - The failure is before or after midway ?
- Refine criteria that defines a program failure

Ask for help when stuck

- Explaining objectively what is going wrong can unlock your mind
  - Do not present your conclusions
  - Accept a naive view
- Getting a fresh view on your code can open your eyes

Overview

Introduction

Debugging Rules

- Is it really a bug ?
- Reproduce and document the bug
- Dichotomic search
- Ask for help

Debugging Tools

- Static checking
- Assert statements
- Print statements
- Log files
- Debuggers
- Memory checkers

Demonstrations

Debugging Tools: Static Checking

1. Use a high-level warning compiler switchs
   - Warning = Error
   - Can detect :
     - Unitialized variables
     - Dead code
     - Forgotten returns in a non-void function
   - gcc : -Wall -Wextra -Wfloat-equal -Werror
   - java : -warn:+unused,uselessTypeCheck,unnecessaryElse

2. Lint-like tools : more detailed check than the compiler
   - Splint for C (not C++)
   - Jlint for Java
   - Ftncheck for Fortran 77

Debugging Tools: Assert Statements

Assertions

- Pre- and post-conditions checked at runtime in a function
- Interrupt the program if the assertion is false and locate the failure
- Help to detect critical failures. E.g.:
  - Values obviously wrong (division by zero)
  - Impossible behavior: e.g. « assert(false) » for a *default* case.

en C/C++

- assert(x>0);
- Abort the process:
  a.out: assert.c:6: int main(int, char**): Assertion `x>0' failed.

en Java

- assert(x>0): ¨ x = ¨+x;
- Throws an AssertionError exception

Debugging Tools: Print Statements

Quick and dirty way to know values at runtime

Pollute your code

Can slow down your program

- Screen IO is much slower than disk IO
- Change the timing of some applications

Main default: you need to rebuild for each new information

Conclusion

- For immediate debugging : better to use a debugger
- For production debugging : better to log to a trace file
- Just when you can not avoid it!

Debugging Tools: Log

Used when software is distributed

Similar to print statements, with disk IO instead of screen IO

Example in C++

- #define Nominal(A) clog << NIV_NOMINAL << __FILE__ << __func__ << __LINE__ << ":" << A << endl;

Different debugging levels:

- Functional level
  File opened, algorithm applied, ...
- Logging level can be changed for more details

Tools

- Log4j, log4cpp, ...

Debugging Tools: (gdb, jdb)

Powerful command-line tools

- Thread
- Stack
- States of the variables
- Breakpoints, conditional breakpoints.

Can give a lot of information in a single build-debugging cycle

Steep Learning Curve

- Read the manual
- Fluency comes with practice
- Help command

Switchs for the compiler

- gcc -g

Debugging Tools: Memory Checking

Valgrind : runtime memory check for C/C++

- Comes with plugins to check specific program behaviour
  - Memcheck : default tools for pointer problems, memory leaks, ...
  - Massif : heap usage
  - ...

Other tools :

- Electric Fence-DUMA, Mpatrol, purify, zerofault, ...

Main difficulties:

- Slow and waste a lot of memory
- They produce large quantities of output
- A lot of false positives
- Need practice to grasp the useful data

Examples

1. Core autopsy with gdb
2. Valgrind (plugin « memcheck »)

In both cases:

Compile your software with -g

- g++ -g exemple1.cpp

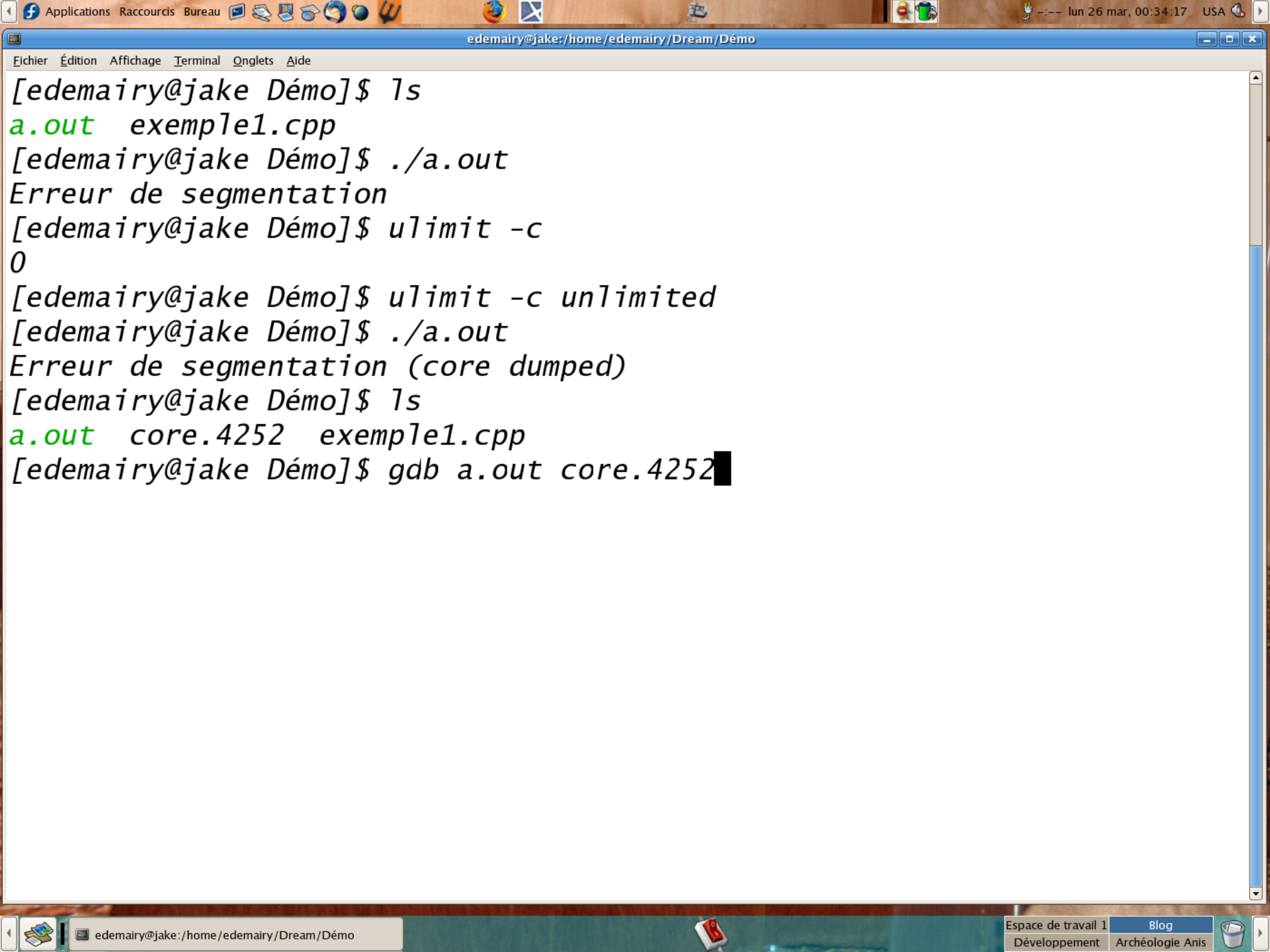
```
#include <string.h>
#include <iostream>
using namespace std;

char chaine[] = "une chaîne de caractère";

int main(int argc, char** argv) {
        strcpy( chaine, "une plus longue chaîne de caractèreaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");

        int* vecteur = new( int[10]);
        delete vecteur;

        cout << chaine << endl;
}
~
~
~
~
~
~
"exemple1.cpp" 14L, 476C                                13,2-9        Tout
```


Applications Raccourcis Bureau
-:-- lun 26 mar, 00:34:17 USA
edemairy@jake:/home/edemairy/Dream/Démo
Fichier Édition Affichage Terminal Onglets Aide
[edemairy@jake Démo]$ ls
a.out exemple1.cpp
[edemairy@jake Démo]$ ./a.out
Erreur de segmentation
[edemairy@jake Démo]$ ulimit -c
0
[edemairy@jake Démo]$ ulimit -c unlimited
[edemairy@jake Démo]$ ./a.out
Erreur de segmentation (core dumped)
[edemairy@jake Démo]$ ls
a.out core.4252 exemple1.cpp
[edemairy@jake Démo]$ gdb a.out core.4252
edemairy@jake:/home/edemairy/Dream/Démo
Espace de travail 1
Blog
Développement
Archéologie Anis

edemairy@jake:/home/edemairy/Dream/Démo

Fichier Édition Affichage Terminal Onglets Aide

```
Loaded symbols for /lib/ld-linux.so.2
#0  0x004a16aa in std::ostream::sentry::sentry ()
   from /usr/lib/libstdc++.so.6
(gdb) back
#0  0x004a16aa in std::ostream::sentry::sentry ()
   from /usr/lib/libstdc++.so.6
#1  0x004a27da in std::operator<< <std::char_traits<char> > ()
   from /usr/lib/libstdc++.so.6
#2  0x080487a4 in main () at exemple1.cpp:13
(gdb) list 10
5       char chaine[] = "une chaîne de caractère";
6
7       int main(int argc, char** argv) {
8               strcpy( chaine, "une plus longue chaîne de caractèreaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");
9
10              int* vecteur = new( int[10]);
11              delete vecteur;
12
13              cout << chaine << endl;
14      }
(gdb)
```

```
   from /usr/lib/libstdc++.so.6
#2  0x080487a4 in main () at exemple1.cpp:13
(gdb) list 10
5       char chaine[] = "une chaîne de caractère";
6
7       int main(int argc, char** argv) {
8               strcpy( chaine, "une plus longue chaîne de caractèreaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");
9
10              int* vecteur = new( int[10]);
11              delete vecteur;
12
13              cout << chaine << endl;
14      }
(gdb) print chaine
$2 = "une plus longue chaîne de"
(gdb) x/s chaine
0x8049b64 <chaine>:      "une plus longue chaîne de caractère", 'a' <rep
eats 163 times>...
```

Valgrind Usage Example

Valgrind ./a.out

edemairy@jake:/home/edemairy/Dream/Démo

Fichier Édition Affichage Terminal Onglets Aide

```
==4394== Copyright (C) 2000-2005, and GNU GPL'd, by Julian Seward et al.
==4394== For more details, rerun with: -v
==4394==
==4394== Mismatched free() / delete / delete []
==4394==    at 0x4004B85: operator delete(void*) (vg_replace_malloc.c:24
6)
==4394==    by 0x804878F: main (exemple1.cpp:11)
==4394==  Address 0x4027028 is 0 bytes inside a block of size 40 alloc'd
==4394==    at 0x4005628: operator new[](unsigned) (vg_replace_malloc.c:
197)
==4394==    by 0x8048781: main (exemple1.cpp:10)
==4394==
==4394== Invalid read of size 4
==4394==    at 0x4A16AA: std::ostream::sentry::sentry(std::ostream&) (in
 /usr/lib/libstdc++.so.6.0.8)
==4394==    by 0x4A27D9: std::basic_ostream<char, std::char_traits<char>
 >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, s
td::char_traits<char> >&, char const*) (in /usr/lib/libstdc++.so.6.0.8)
==4394==    by 0x80487A3: main (exemple1.cpp:13)
==4394==  Address 0x63617255 is not stack'd, malloc'd or (recently) free
'd
==4394==
==4394== Invalid read of size 4
==4394==    at 0x4A16BD: std::ostream::sentry::sentry(std::ostream&) (in
```

```
#include <string.h>
#include <iostream>
using namespace std;

char chaine[] = "une chaîne de caractère";

int main(int argc, char** argv) {
        strcpy( chaine, "une plus longue chaîne de caractèreaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");

        int* vecteur = new( int[10]);
        delete vecteur;

        cout << chaine << endl;
}
~
~
~
~
~
~
"exemple1.cpp" 14L, 476C                                13,2-9        Tout
```

Conclusion

Use static checking tools as often as possible

- Set your compiler's warning-level properly and understand the output

When debugging

- Better use a debugger
- Use print or log statements if no other choice
- Use memory checker to track down memory mismanagement

Complementary Tools

Ask for help when you are stuck

References

“Debugging: the 9 indispensable rules for finding the most elusive software and hardware problems” by David J. Agans

Tool manuals

Library manuals

Developper documentation

Links:

- valgrind: http://valgrind.org
- gdb: http://www.gnu.org/software/gdb
- jdb: http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/jdb.html