Development Tools

Akim Demaille akim@lrde.epita.fr
Roland Levillain roland@lrde.epita.fr

EPITA — École Pour l'Informatique et les Techniques Avancées

June 14, 2012

Development Tools

tc Tasks

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

The Tiger Compiler layout

- One module, one namespace
- One library per module, with a pure interface (‘`libfoo.*`’)
- One task set per module, maybe impure (‘`tasks.*`’)
- Tasks describe the command line interface
- Requirements over tasks order
- One class, one file base name:

foo.hh	Interface
foo.hxx	Inline implementation
foo.cc	Implementation

Tasks: 'ast/tasks.hh'

```
namespace ast
{
  namespace tasks
  {
    /// Global root node of abstract syntax tree.
    extern ast::DecsList* the_program;

    TASK_GROUP ("2. Abstract Syntax Tree");

    /// Display the abstract syntax tree
    TASK_DECLARE ("A|ast-display", "display the AST",
                  ast_display, "parse");

    /// Free the ast (if defined)
    TASK_DECLARE ("D|ast-delete", "delete the AST",
                  ast_delete, "parse");
  } // namespace tasks
} // namespace ast
```

Tasks: ‘ast/tasks.cc’

```
namespace ast
{
  namespace tasks
  {
    ast::DecsList* the_program = 0;

    void ast_display ()
    {
      precondition (the_program);
      std::cout << "/* Abstract Syntax Tree. */" << std::endl
                << *the_program << std::endl;
    }

    void ast_delete ()
    {
      delete the_program;
      the_program = 0;
    }
  } // namespace tasks
} // namespace ast
```

Maintaining Packages

GNU Tools

GNU Autotools

aclocal	Create ‘aclocal.m4’ from ‘configure.ac’’s requests
autoconf	Create ‘configure’ from ‘configure.ac’ and ‘aclocal.m4’
autoheader	Create ‘config.h.in’ from ‘configure.ac’ (and ‘aclocal.m4’)
automake	Create ‘Makefile.in’ from ‘Makefile.am’ and ‘configure.ac’
autoreconf	Run them as needed (autoreconf -fivm)

Read Alexandre Duret-Lutz’s Tutorial [1]

GNU Autotools

aclocal	Create 'aclocal.m4' from 'configure.ac''s requests
autoconf	Create 'configure' from 'configure.ac' and 'aclocal.m4'
autoheader	Create 'config.h.in' from 'configure.ac' (and 'aclocal.m4')
automake	Create 'Makefile.in' from 'Makefile.am' and 'configure.ac'
autoreconf	Run them as needed (autoreconf -fivm)

Read Alexandre Duret-Lutz's Tutorial [1]

GNU Autotools

aclocal	Create ‘aclocal.m4’ from ‘configure.ac”s requests
autoconf	Create ‘configure’ from ‘configure.ac’ and ‘aclocal.m4’
autoheader	Create ‘config.h.in’ from ‘configure.ac’ (and ‘aclocal.m4’)
automake	Create ‘Makefile.in’ from ‘Makefile.am’ and ‘configure.ac’
autoreconf	Run them as needed (autoreconf -fivm)

Read Alexandre Duret-Lutz’s Tutorial [1]

GNU Autotools

aclocal	Create 'aclocal.m4' from 'configure.ac''s requests
autoconf	Create 'configure' from 'configure.ac' and 'aclocal.m4'
autoheader	Create 'config.h.in' from 'configure.ac' (and 'aclocal.m4')
automake	Create 'Makefile.in' from 'Makefile.am' and 'configure.ac'
autoreconf	Run them as needed (autoreconf -fivm)

Read Alexandre Duret-Lutz's Tutorial [1]

GNU Autotools

aclocal	Create ‘aclocal.m4’ from ‘configure.ac”s requests
autoconf	Create ‘configure’ from ‘configure.ac’ and ‘aclocal.m4’
autoheader	Create ‘config.h.in’ from ‘configure.ac’ (and ‘aclocal.m4’)
automake	Create ‘Makefile.in’ from ‘Makefile.am’ and ‘configure.ac’
autoreconf	Run them as needed (autoreconf -fivm)

Read Alexandre Duret-Lutz’s Tutorial [1]

GNU Autotools

aclocal	Create 'aclocal.m4' from 'configure.ac''s requests
autoconf	Create 'configure' from 'configure.ac' and 'aclocal.m4'
autoheader	Create 'config.h.in' from 'configure.ac' (and 'aclocal.m4')
automake	Create 'Makefile.in' from 'Makefile.am' and 'configure.ac'
autoreconf	Run them as needed (`autoreconf -fivm`)

Read Alexandre Duret-Lutz's Tutorial [1]

Programming Tools: Packages

A set of packages to maintain packages:

Autoconf	package configuration [3]
Automake	package build [2]
Libtool	portable build of shared libs
Gettext	package internationalization
Argp	extended getopt
Flex	scanner generation
Bison	parser generation

Programming Tools: Packages

A set of packages to maintain packages:

Autoconf	package configuration [3]
Automake	package build [2]
Libtool	portable build of shared libs
Gettext	package internationalization
Argp	extended getopt
Flex	scanner generation
Bison	parser generation

Programming Tools: Packages

A set of packages to maintain packages:

Autoconf	package configuration [3]
Automake	package build [2]
Libtool	portable build of shared libs
Gettext	package internationalization
Argp	extended getopt
Flex	scanner generation
Bison	parser generation

Programming Tools: Packages

A set of packages to maintain packages:

Autoconf	package configuration [3]
Automake	package build [2]
Libtool	portable build of shared libs
Gettext	package internationalization
Argp	extended getopt
Flex	scanner generation
Bison	parser generation

Programming Tools: Packages

A set of packages to maintain packages:

Autoconf	package configuration [3]
Automake	package build [2]
Libtool	portable build of shared libs
Gettext	package internationalization
Argp	extended `getopt`
Flex	scanner generation
Bison	parser generation

Programming Tools: Packages

A set of packages to maintain packages:

Autoconf	package configuration [3]
Automake	package build [2]
Libtool	portable build of shared libs
Gettext	package internationalization
Argp	extended `getopt`
Flex	scanner generation
Bison	parser generation

Programming Tools: Packages

A set of packages to maintain packages:

Autoconf	package configuration [3]
Automake	package build [2]
Libtool	portable build of shared libs
Gettext	package internationalization
Argp	extended `getopt`
Flex	scanner generation
Bison	parser generation

Autoconf for tc

Autoconf files

Configuring a package

```
configure -----------+------------> config.log
                     |
config.h.in -.       v          .-> config.h -.
             +-> config.status -+              +--> make
Makefile.in -'                  `-> Makefile -'
```

Preparing a package for distribution

```
configure.ac --.
               |   .-----> autoconf ----> configure
               +---+
               |   `-----> autoheader --> config.h.in
aclocal.m4 ----'
```

Autoconf files

Configuring a package

```
configure -----------+------------> config.log
                     |
config.h.in -.       v          .-> config.h -.
             +-> config.status -+             +--> make
Makefile.in -'                  `-> Makefile -'
```

Preparing a package for distribution

```
configure.ac --.
               |   .-----> autoconf ----> configure
               +---+
               |   `-----> autoheader --> config.h.in
aclocal.m4 ----'
```

'configure.ac' 1: Initialization

```
AC_PREREQ([2.64])
AC_INIT([LRDE Tiger Compiler], [1.29a],
        [tiger@lrde.epita.fr], [tc])

# Auxiliary files.
AC_CONFIG_AUX_DIR([build-aux])
AC_CONFIG_MACRO_DIR([build-aux])

# Automake.
AM_INIT_AUTOMAKE([1.11.1 check-news dist-bzip2 no-dist-gzip
                  foreign
                  color-tests parallel-tests
                  nostdinc silent-rules -Wall])
AM_SILENT_RULES([yes])
```

'configure.ac' 2: C++ Compiler

```
# Look for a C++ compiler.
AC_LANG([C++])
AC_PROG_CXX

# Speed the compilation up.
if test "$GXX" = yes; then
  CXXFLAGS="$CXXFLAGS -pipe"
fi

# Use good warnings.
TC_CXX_WARNINGS([[-Wall], [-W], [-Wcast-align], ...])
```

'configure.ac' 3: Auxiliary Programs

```
TC_PROG([flex], [>= 2.5.4], [FLEX],
        [Flex scanner generator])
AM_PROG_LEX
TC_PROG([bison], [>= 2.4], [BISON],
        [Bison parser generator])
AC_CONFIG_FILES([build-aux/bison++],
                [chmod +x build-aux/bison++])

# We don't need static libraries, speed the compilation up.
: ${enable_shared=no}
AC_PROG_LIBTOOL

TC_PROG([monoburg], [>= 1.0.5], [MONOBURG],
        [MonoBURG code generator generator])
AC_CONFIG_FILES([build-aux/monoburg++],
                [chmod +x build-aux/monoburg++])
TC_PROG([havm], [>=0.23], [HAVM],
                [The Tree Virtual Machine])
```

‘configure.ac’ 4: Libraries

```
AC_CONFIG_SUBDIRS([lib/argp])

TC_HEADER_BOOST([1.34])
# Boost.Conversion defines lexical_cast
BOOST_CONVERSION
BOOST_FOREACH
BOOST_GRAPH
BOOST_LAMBDA
BOOST_PREPROCESSOR
BOOST_STRING_ALGO
BOOST_TRIBOOL
BOOST_VARIANT
```

'configure.ac' 5: SWIG & tcsh

```
TC_WITH_TCSH([with_tcsh=yes], [with_tcsh=no])
AM_CONDITIONAL([ENABLE_TCSH], [test x$with_tcsh = xyes])

AC_CONFIG_FILES([tcsh/Makefile
                 tcsh/python/Makefile
                 tcsh/ruby/Makefile])
AC_CONFIG_FILES([tcsh/run], [chmod +x tcsh/run])
```

‘configure.ac’ 6: Modules

```
MODULES=
for module in ‘cd $srcdir/src && ls‘
do
  if test -f $srcdir/src/$module/tasks.hh; then
    MODULES="$MODULES $module";
  fi
done
AC_SUBST([MODULES])
```

'configure.ac' 7: File Creation

```
# Ask for the creation of config.h.
AM_CONFIG_HEADER([config.h])

# Ask for the creation of the Makefiles.
AC_CONFIG_FILES([
  Makefile
    lib/Makefile
    src/Makefile
    doc/Makefile
])

# Instantiate the output files.
AC_OUTPUT
```

Automake for tc

'src/tc.mk' 1: Common Options

```
# Most headers are to be shipped and to be found in src/, e.g.
# tasks/tasks.hh is shipped in $(top_srcdir)/src/task/tasks.hh.
# Some are *built* in src, e.g., $(top_builddir)/src/modules.hh.
AM_CPPFLAGS = -I$(top_srcdir)/lib
AM_CPPFLAGS += -I$(top_srcdir)/src -I$(top_builddir)/src
AM_CPPFLAGS += $(BOOST_CPPFLAGS)
# Find the prelude.
AM_CPPFLAGS += -DPKGDATADIR="\"$(pkgdatadir)\""

AM_CXXFLAGS = $(WARNING_CXXFLAGS)
```

‘src/tc.mk’ 2: Libraries

```
# All our libraries, in the order of dependency.
libregalloc_la  = $(top_builddir)/src/regalloc/libregalloc.la
libliveness_la  = $(top_builddir)/src/liveness/libliveness.la
[...]
libtask_la      = $(top_builddir)/src/task/libtask.la
libmisc_la      = $(top_builddir)/src/misc/libmisc.la
libargp_la      = $(top_builddir)/argp/libargp.la
```

‘src/tc.mk’ 3: Libraries Dependencies

```
# All our libraries...
libregalloc  = $(libregalloc_la)
libliveness  = $(libliveness_la)
[...]
libtask      = $(libtask_la)
libmisc      = $(libmisc_la)
libargp      = $(libargp_la)

# ... and their dependencies.
libregalloc  += $(libliveness)
libliveness  += $(libassem)
[...]
libast       += $(libmisc)
```

‘src/Makefile.am’ 1: Variables

```
include $(top_srcdir)/src/tc.mk
AUTOMAKE_OPTIONS = subdir-objects

AM_DEFAULT_SOURCE_EXT = .cc

BUILT_SOURCES =
CLEANFILES =
EXTRA_DIST =
MAINTAINERCLEANFILES =
TESTS =
EXTRA_PROGRAMS = $(TESTS)
dist_noinst_DATA =
noinst_LTLIBRARIES =

RECHECK_LOGS =

# Find the configuration headers.
AM_CPPFLAGS += -I $(top_builddir)
```

'src/Makefile.am' 2: Tasks

```
TASKS =
include task/local.mk
include ast/local.mk
[...]
include regalloc/local.mk

EXTRA_DIST += tiger_common.i
```

‘src/Makefile.am’ 3: Building libtc

```
lib_LTLIBRARIES = libtc.la
libtc_la_SOURCES = version.hh
nodist_libtc_la_SOURCES = version.cc
BUILT_SOURCES = $(nodist_libtc_la_SOURCES)
CLEANFILES += $(nodist_libtc_la_SOURCES)

## Don't forget that if liba depends on libb, then libb must
## be specified *after* liba.
##
## We cannot use $(libfoo) because some libraries appear
## several times, resulting in a library comprising several
## definitions of some symbols.
libtc_la_LIBADD =                                   \
        $(libregalloc_la)                           \
        ...
        $(libargp_la)
```

‘src/Makefile.am’ 4: Building tc

```
bin_PROGRAMS = tc
dist_tc_SOURCES =                         \
  doc.hh                                  \
  $(TASKS)                                \
  common.cc common.hh                     \
  tc.cc

tc_LDADD = $(libtask_la) libtc.la
```

'src/bind/local.mk': Binding Names

```
EXTRA_DIST += bind/tiger_bind.i

noinst_LTLIBRARIES += bind/libbind.la
bind_libbind_la_SOURCES =                                  \
  bind/binder.hh bind/binder.hxx bind/binder.cc            \
  bind/renamer.hh bind/renamer.hxx bind/renamer.cc         \
  bind/libbind.hh bind/libbind.cc

TESTS += bind/test-bind
bind_test_bind_LDADD = $(libbind)

TASKS += bind/tasks.hh bind/tasks.cc
```

Tools for the Developer

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer

- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer

- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Other developing tools

- Use warnings
- Use the `assert` macro
- Electric Fence
- DUMA
- Dmalloc
- AddressSanitizer
- Bound Checking GCC
- Mudflap (built in GCC)
- Purify (proprietary)
- GDB
- Valgrind
- DTrace
- Clang Static Analyzer (LLVM)

Mudflap

```
int
main ()
{
  int tab[10];
  int i;

  for (i = 0; i <= 10; ++i)
    tab[i] = 0;
  return 0;
}
gcc -fmudflap -lmudflap bounds-violation.c
```

Mudflap

```
env MUDFLAP_OPTIONS=-viol-abort ./a.out
*******
mudflap violation 1 (check/write): time=1292501299.526454 ptr=0xbfc35d34 size=44
pc=0xb77d13bd location=‘bounds-violation.c:8:5 (main)’
      /usr/lib/libmudflap.so.0(__mf_check+0x3d) [0xb77d13bd]
      ./a.out(main+0xb7) [0x804883b]
      /usr/lib/libmudflap.so.0(__wrap_main+0x49) [0xb77d0b89]
Nearby object 1: checked region begins 0B into and ends 4B after
mudflap object 0x8c7a080: name=‘bounds-violation.c:4:7 (main) tab’
bounds=[0xbfc35d34,0xbfc35d5b] size=40 area=stack check=0r/4w liveness=4
alloc time=1292501299.526444 pc=0xb77d0b2d
number of nearby objects: 1
zsh: abort      env MUDFLAP_OPTIONS=-viol-abort ./a.out
```

Valgrind and Memory Violation

```c
#include <stdio.h>

typedef struct list_s { int val; struct list_s *next; } list_t;

list_t *list_new (int val, list_t *next) {
  list_t res = { val, next };
  return &res;
}

void list_print (const list_t *const list) {
  if (list)
    printf ("%d\n", list->val), list_print (list->next);
}
int main (void) {
  list_print (list_new (2, list_new (1, list_new (0, NULL))));
  return 0;
}
```

Valgrind and Memory Leaks

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

typedef struct list_s { int val; struct list_s *next; } list_t;

list_t *list_new (int val, list_t *next) {
  list_t *res = (list_t *) malloc (sizeof (list_t));
  res->val  = val; res->next = next;
  return res;
}

void list_print (const list_t *const list) {
  if (list)
    printf ("%d\n", list->val), list_print (list->next);
}

int main (void) {
  list_print (list_new (2, list_new (1, list_new (0, NULL))));
  return 0;
}
```

Valgrind and Memory Leaks

```
gcc -g memory-leaks.c
valgrind --leak-check=full ./a.out
==9702== Memcheck, a memory error detector
==9702== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==9702== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==9702== Command: ./a.out
==9702==
2
1
0
==9702==
==9702== HEAP SUMMARY:
==9702==     in use at exit: 24 bytes in 3 blocks
==9702==   total heap usage: 3 allocs, 0 frees, 24 bytes allocated
==9702==
==9702== 24 (8 direct, 16 indirect) bytes in 1 blocks are definitely lost in loss record 3 of 3
==9702==    at 0x4023F50: malloc (vg_replace_malloc.c:236)
==9702==    by 0x8048405: list_new (memory-leaks.c:7)
==9702==    by 0x804848D: main (memory-leaks.c:18)
==9702==
==9702== LEAK SUMMARY:
==9702==    definitely lost: 8 bytes in 1 blocks
==9702==    indirectly lost: 16 bytes in 2 blocks
==9702==      possibly lost: 0 bytes in 0 blocks
==9702==    still reachable: 0 bytes in 0 blocks
==9702==         suppressed: 0 bytes in 0 blocks
==9702==
==9702== For counts of detected and suppressed errors, rerun with: -v
==9702== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 11 from 6)
```

A Clear Winner?

- Valgrind doesn’t catch the previous Mudflap example.
    - Padding
    - Overrun into neighbor regions
- Mudflap doesn’t know about uninitialized regions.

A Clear Winner?

- Valgrind doesn't catch the previous Mudflap example.
  - Padding
  - Overrun into neighbor regions
- Mudflap doesn't know about uninitialized regions.

A Clear Winner?

- Valgrind doesn't catch the previous Mudflap example.
  - Padding
  - Overrun into neighbor regions
- Mudflap doesn't know about uninitialized regions.

A Clear Winner?

- Valgrind doesn't catch the previous Mudflap example.
  - Padding
  - Overrun into neighbor regions
- Mudflap doesn't know about uninitialized regions.

Version Control

- Makes working in group *a lot* easier.
- Gives the possibility to travel back in time (e.g, to hunt bugs).
- Allows several, non-linear developing models (branches).
- Add some semantics to the development itself.
- Provides a kind of backup
  But cannot make up for the lack of a real backup solution!
- EPITA provides Git repositories for the Tiger project. :-)

Version Control

- Makes working in group *a lot* easier.
- Gives the possibility to travel back in time (e.g, to hunt bugs).
- Allows several, non-linear developing models (branches).
- Add some semantics to the development itself.
- Provides a kind of backup
  But cannot make up for the lack of a real backup solution!
- EPITA provides Git repositories for the Tiger project. :-)

Version Control

- Makes working in group *a lot* easier.
- Gives the possibility to travel back in time (e.g, to hunt bugs).
- Allows several, non-linear developing models (branches).
- Add some semantics to the development itself.
- Provides a kind of backup
  But cannot make up for the lack of a real backup solution!
- EPITA provides Git repositories for the Tiger project. :-)

Version Control

- Makes working in group *a lot* easier.
- Gives the possibility to travel back in time (e.g, to hunt bugs).
- Allows several, non-linear developing models (branches).
- Add some semantics to the development itself.
- Provides a kind of backup
  But cannot make up for the lack of a real backup solution!
- EPITA provides Git repositories for the Tiger project. :-)

Version Control

- Makes working in group *a lot* easier.
- Gives the possibility to travel back in time (e.g, to hunt bugs).
- Allows several, non-linear developing models (branches).
- Add some semantics to the development itself.
- Provides a kind of backup
  But cannot make up for the lack of a real backup solution!
- EPITA provides Git repositories for the Tiger project. :-)

Version Control

- Makes working in group *a lot* easier.
- Gives the possibility to travel back in time (e.g, to hunt bugs).
- Allows several, non-linear developing models (branches).
- Add some semantics to the development itself.
- Provides a kind of backup
  But cannot make up for the lack of a real backup solution!
- EPITA provides Git repositories for the Tiger project. :-)

Document with Doxygen

- Use comments to annotate code entities (namespaces, files, functions, classes, `typedefs`, etc.).
- Generate a hyper-text reference documentation.
- Several back-ends: HTML, PDF, RTF, etc.
- Use `make doc` in the Tiger Project.
- For more information, see `http://www.stack.nl/~dimitri/doxygen/manual.html`.

Document with Doxygen

- Use comments to annotate code entities (namespaces, files, functions, classes, `typedefs`, etc.).
- Generate a hyper-text reference documentation.
- Several back-ends: HTML, PDF, RTF, etc.
- Use `make doc` in the Tiger Project.
- For more information, see `http://www.stack.nl/~dimitri/doxygen/manual.html`.

Document with Doxygen

- Use comments to annotate code entities (namespaces, files, functions, classes, `typedefs`, etc.).
- Generate a hyper-text reference documentation.
- Several back-ends: HTML, PDF, RTF, etc.
- Use `make doc` in the Tiger Project.
- For more information, see `http://www.stack.nl/~dimitri/doxygen/manual.html`.

Document with Doxygen

- Use comments to annotate code entities (namespaces, files, functions, classes, `typedefs`, etc.).
- Generate a hyper-text reference documentation.
- Several back-ends: HTML, PDF, RTF, etc.
- Use `make doc` in the Tiger Project.
- For more information, see `http://www.stack.nl/~dimitri/doxygen/manual.html`.

Document with Doxygen

- Use comments to annotate code entities (namespaces, files, functions, classes, `typedefs`, etc.).
- Generate a hyper-text reference documentation.
- Several back-ends: HTML, PDF, RTF, etc.
- Use `make doc` in the Tiger Project.
- For more information, see `http://www.stack.nl/~dimitri/doxygen/manual.html`.

Document with Doxygen: ‘type/libtype.hh’

```
/// \file type/libtype.hh
/// \brief Declare the function exported by type module.
#ifndef TYPE_LIBTYPE_HH
# define TYPE_LIBTYPE_HH

# include "misc/error.hh"
# include "ast/fwd.hh"

/// Type-checking an ast::Ast.
namespace type
{

  /** \brief Check types in a (bound) AST.
   ** \param tree   abstract syntax tree's root.
   ** \return       synthesis of the errors possibly found. */
  misc::error types_check (ast::Ast& tree);

} // namespace type

#endif // !TYPE_LIBTYPE_HH
```

Bibliography I

Alexandre Duret-Lutz.
The Autotools Tutorial.
http://www-src.lip6.fr/homepages/Alexandre.Duret-Lutz/dl/autotools.pdf/, 2006.

Alexandre Duret-Lutz and Tom Tromey.
GNU Automake.
http://www.gnu.org/software/automake/, 2003.

David J. MacKenzie, Ben Elliston, and Akim Demaille.
GNU Autoconf.
http://www.gnu.org/software/autoconf/, 2003.