

# New C compiler and analyzer lang/cparser in ports

---

*Source:* <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/hackers/2008-11/msg00282.html>

---

- *From:* Christoph Mallon <[christoph.mallon@xxxxxx](mailto:christoph.mallon@xxxxxx)>
  - *Date:* Thu, 27 Nov 2008 21:39:45 +0100
- 

A few days ago libFIRM[1] and cparser were added to the ports tree. If you want to see, what other compilers besides GCC have to offer, this might be of interest for you. libFIRM is a modern optimizing intermediate representation (IR) library. cparser is a C compiler providing many useful warnings and uses libFIRM for optimization and code generation.

libFIRM is a library implementing a modern graph based IR for compilers, which exclusively operates on program graphs in SSA-form. It is developed at the IPD at the Universität Karlsruhe (TH)[2]. SSA is maintained beginning at the construction of the IR graph until assembler code is emitted. libFIRM offers many analyses and optimizations, provides extensive debugging support and includes a backend framework. At the moment it can produce code for 32bit x86 processors. There are also unfinished backends for ARM, MIPS and PPC32. The backend uses a novel SSA based register allocator and includes several algorithms for instruction scheduling, spilling, copy coalescing and basic block scheduling. All stages of the compilation process can be dumped and inspected with our graph viewer yComp[3].

cparser is a C compiler, which can parse C89 and C99 as well as many GCC and some MSVC extensions. The handled GCC extensions include `__attribute__`, inline assembler, computed goto and statement expressions. The compiler driver is compatible with with GCC (`-fxxx`, `-Wxxx`, `-M`, ...). It also provides many useful analyses for warnings – for some examples see below.

Take a look at the examples below to get a glimpse of what cparser and libFIRM have to offer. You can find more information about them at <http://www.libfirm.org>. Of course, they are capable of compiling non-trivial programs like the SPEC INT2000 suite and Quake 3[4]. If I have sparked your interest in cparser and libFIRM, try the port lang/cparser. If you have any questions, suggestions or problems regarding cparser and libFIRM, please feel free to contact us.

Regards  
Christoph

## Examples

---

One goal is to provide concise error messages, which help to pinpoint the cause of the problem. This is achieved by using a handwritten recursive descent parser, which includes some heuristics to produce sensible error messages. A further aim is to restart parsing immediately after a parse error, in order to avoid subsequent errors. In this way, the number of stray errors after common problems, like a missing `#include` or a misspelled type name, is minimized. Here is a small example:

```
1 struct X {
```

## New C compiler and analyzer lang/cparser in ports

```
2 unsigned lng x;
3 unsigned int y;
4 };
5 void f(struct X* p) { p.x = 23; }
6 void g(struct X* p) { p->y = p.z; }
```

```
parse_error.c:2: error: discarding stray 'symbol lng' in declaration specifier
parse_error.c:5: error: 'symbol vodi' does not name a type
parse_error.c:5: error: request for member 'x' in something not a struct or union, but 'struct X*'
parse_error.c:6: error: request for member 'z' in something not a struct or union, but 'struct X*'
parse_error.c:6: error: 'struct X*' has no member named 'z'
```

For comparison, these are the error messages by GCC 4.2.1:

```
parse_error.c:2: error: expected '!', ',', ';' or '__attribute__' before 'x'
parse_error.c:5: error: expected '=', ',', ';', 'asm' or '__attribute__' before 'f'
parse_error.c: In function 'g':
parse_error.c:6: error: 'struct X' has no member named 'y'
parse_error.c:6: error: request for member 'z' in something not a structure or union
```

cparser can detect a large number of undesirable constructs and warns about them. Here are a few examples:

\* unused and never read variables

```
1 #include <stdlib.h>
2 void f(void)
3 {
4 int x;
5 if ((x = rand()))
6 x = x + 1;
7 }
```

```
write_only.c:4: warning: variable 'x' is never read
```

'x' is warned about, because it is only read to calculate its own new value. GCC does not detect this. Similar cases to line 5 have been found by Coverity in the FreeBSD source code recently[5].

\* unreachable code

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(void)
4 {
5 for (int i = 0; i != 10; ++i) {
6 if (rand() == 23) /* mind the missing {} */
7 puts("ouch");
8 return 1;
9 }
10 return 0;
11 }
```

```
unreachable.c:5: warning: step of for-statement is unreachable
```

GCC does not warn here.

\* format string checker for char and wide char (wchar\_t) functions

```
1 #include <stdio.h>
2 #include <wchar.h>
3 int main(int argc, char* argv[])
4 {
5 printf("%u %s", argc);
6 wprintf(L"%S", "hello", 42);
7 return 0;
8 }
```

```
format.c:5: warning: argument type 'int' does not match conversion specifier '%u' at position 1
format.c:5: warning: too few arguments for format string
format.c:6: warning: argument type 'char*' does not match conversion specifier '%S' at position 1
format.c:6: warning: 2 arguments but only 1 format specifier
```

GCC is only capable of checking char, but not wchar\_t, format strings.

\* missing return statments

```
1 int f(int x)
2 {
3 switch (x) {
4 case 42: return 0;
5 case 23: break;
6 default: return 1;
7 }
8 }
```

```
missing_return.c:5: warning: control reaches end of non-void function
```

The location is exactly pinpointed: The last statement before leaving the function is the break. In contrast GCC refers to line 8.

\* typical security warnings

```
1 #include <sys/types.h>
2 #include <unistd.h>
3 void f(char* x)
4 {
5 if (getpid || &&x || x == "bla") {}
6 }
```

```
security.c:5: warning: the address of 'getpid' will always evaluate as 'true'
security.c:5: warning: the address of 'x' will always evaluate as 'true'
security.c:5: warning: comparison with string literal results in unspecified behaviour
```

\* missing and redundant declarations

```
1 static void f(void);
2 int x;
3 static void f(void) { }
```

decl.c:2: warning: no previous declaration for 'int x'

decl.c:1: warning: unnecessary static forward declaration for 'void f(void)'

The warning about 'x' is a typical indicator for a missing header include or that the variable should be static. The first declaration of 'f' is unnecessary, because there is no use before the definition of the function.

The generated assembler is heavily annotated.

Some example C source code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int f(void)
4 {
5 int x = rand();
6 if (rand())
7 puts("bla");
8 return x;
9 }
```

In the resulting assembler output below you see:

\* list of predecessors and estimated execution frequency for every basic block

```
/* .L897: preds: 1034, freq: 0.500000 */
```

\* comments telling the corresponding node in the intermediate representation and the original source line (if available) of an instruction. The numbers in [] are internal node counters, which can be referred to in yComp or be used in conjunction with the debugging features of libFIRM.

```
/* ia32_CallT[1086:53] annotate.c:6 */
```

\* function begin/end markers (good for scripting)

```
# -- Begin f
```

```
# -- End f
```

The resulting assembler output:

```
.section .text
.Ltext0:
# -- Begin f
.p2align 4,,15
.globl f
.type f, @function
f:
/* .L1034: preds: none, freq: 1.000000 */
subl $4, %esp /* be_IncSPlu[1001:12] */
call rand /* ia32_CallT[1059:26] ann.c:5 */
movl %eax, (%esp) /* ia32_StoreM[1213:171] */
```

## New C compiler and analyzer lang/cparser in ports

```
call rand /* ia32_CallT[1086:53] ann.c:6 */
testl %eax, %eax /* ia32_TestIu[1220:178] ann.c:6 */
jne .L776 /* ProjX[805:139] ann.c:6 */
/* fallthrough to .L897 */ /* ProjX[806:140] ann.c:6 */
/* .L897: preds: 1034, freq: 0.500000 */
popl %eax /* ia32_PopT[1218:176] */
ret /* be_ReturnX[1031:125] ann.c:8 */
.p2align 4,,7
.L776: /* preds: 1034, freq: 0.500000 */
pushl $Lstr.168 /* ia32_PushT[1223:181] ann.c:7 */
call puts /* ia32_CallT[1118:85] ann.c:7 */
addl $4, %esp /* be_IncSPIu[982:113] */
popl %eax /* ia32_PopT[1221:179] */
ret /* be_ReturnX[1020:107] ann.c:8 */
.size f, .-f
# --- End f
```

```
.section .rodata
.type Lstr.168, @object
.size Lstr.168, 4
Lstr.168:
.string "bla"
```

- 
- [1] <http://www.libfirm.org/>
  - [2] <http://www.info.uni-karlsruhe.de/>
  - [3] <http://www.info.uni-karlsruhe.de/software.php/id=6&lang=en>
  - [4] specifically iquake3
  - [5] for example see <http://lists.freebsd.org/pipermail/svn-src-all/2008-November/001623.html>

---

freebsd-hackers@xxxxxxxxxxx mailing list  
<http://lists.freebsd.org/mailman/listinfo/freebsd-hackers>  
To unsubscribe, send any mail to "freebsd-hackers-unsubscribe@xxxxxxxxxxx"