

C/C++ セキュアコーディングセミナー

2010年度版

ROSE

JPCERT コーディネーションセンター

- 膨大なコードベース（数百万行）
- 270以上もあるルールの違反をどうやってチェックするのか？
- プログラマーはセキュアコーディングルールを知らないかもしれない
- 脆弱性はコンパイラ、デバッガ、Q/Aテストの網をかいくぐり、市場に出回る

**社会問題に発展しうる問題。
プログラマーがなんとかしない
といけない。**

CERT C セキュアコーディングルールに違反したコード

<http://www.securecoding.cert.org/>

解決したい課題

- **セキュアでないコード**をいかにして見つけるか

方法

- ツールを用いた自動セキュリティチェック
- 静的解析 (Static Analysis)
- 抽象構文木 (AST)
- rosecheckers

rosecheckers の使い方は後半に

CERTのセキュアコーディングページ (Wiki)

JPCERT CC®



Software Engineering Institute
Carnegie Mellon

Dashboard Secure Coding CERT Secure Coding Standards

Browse - Log In Sign Up

CERT Secure Coding Standards

11 Added by [Confluence Administrator](#), last edited by [Robert Seacord \(Manager\)](#) on Aug 28, 2010 ([view change](#))

Welcome to the Secure Coding Web Site

This web site exists to support the development of secure coding standards for commonly used programming languages such as C, C++, and Java. These standards are being developed through a broad-based community effort including the CERT Secure Coding Initiative and members of the software development and software security communities. For a further explanation of this project and tips on how to contribute, please see the [FAQ](#).

As this is a development web site, many of the pages are incomplete or contain errors. If you are interested in furthering this effort, you may comment on existing items or see the [FAQ](#) for information on how to request editing privileges to directly edit content on the site. If you decide to link to our guidelines, use the **Tiny Link** under Tools->Info as this URL will not change if the name of the guideline changes.

Wikiで運営されており、コントリビューターがコンテンツを編集できる

The CERT Oracle Secure Coding Standard for Java



CERT and Oracle are developing [The CERT Oracle Secure Coding Standard for Java](#).

The rules and recommendations are not globally editable, but anyone is able to add comments, and qualified individuals can be added as editors.

We are depending on the active involvement of the Java community (you) to make this effort a success. We invite you to participate in this effort by reviewing content in the Java space and providing comments, or by contributing new rules and recommendations for secure Java coding. These can be included as comments or emailed to secure-coding@cert dot org.

Java is a trademark or registered trademark of Oracle Corporation, in the US and other countries.

C, C++, Java版のセキュアコーディングルール集が日々アップデートされている

Java Concurrency Guidelines TR Released

CERT has released the [Java Concurrency Guidelines](#) technical report that documents the portion of the [CERT Oracle Secure Coding Standard for Java](#) that are related to concurrency.

The CERT C Secure Coding Standard



Version 1.0 of The CERT C Secure Coding Standard is now available as a [book](#) from Addison-Wesley. This official release can be used as a fixed point of reference for the development of compliant applications and source code analysis tools.

Development of the next version of the [CERT C Secure Coding Standard](#) is being performed here on the secure coding wiki. This version is a work-in-progress and reflects the current thinking of the secure coding community. Subsequent official releases of this standard will be issued as dictated by the needs and interests of the secure software development community.

There is also a Japanese Edition of the [CERT C Secure Coding Standard](#) thanks to our partner JPCERT/CC.

業界のエキスパート(Oracle(元Sun)のJava開発者やCの言語仕様を策定しているエンジニアなども)の作成に携わっている

The CERT C++ Secure Coding Standard

The [CERT C++ Secure Coding Standard](#) is under development. Please create a sign in account, review,



サイト内検索

検索

[トップページ](#)

情報提供

- ・注意喚起
- ・早期警戒
- ・脆弱性対策情報
- ・Weekly Report
- ・インターネット定点観測

[インシデントの報告](#)

[各種登録](#)

[制御システムセキュリティ](#)

[ラーニング](#)

・セキュアコーディング

・技術メモ

・ライブラリ

[公開資料](#)

[イベント](#)

[プレスリリース](#)

[JPCERT/CC](#)

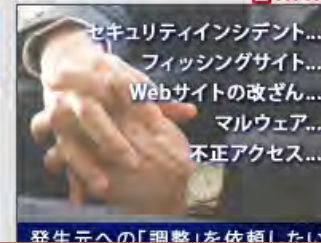
セキュアコーディング

最終更新: 2010-04-07

プログラム開発業務に携わる全ての方々に向けて、脆弱性のない、安全なソフトウェア開発のためのセミナー、コーディングのルールやそのマテリアル、書籍に関する情報を紹介しています。

- セミナー
 - ・ [C/C++ セキュアコーディングセミナーのご案内](#)
 - ・ [C/C++ セキュアコーディングセミナー資料](#)
- CERT セキュアコーディングスタンダード
 - ・ [CERT C Secure Coding Standard 日本語版](#)
- 書籍
 - ・ [C/C++ セキュアコーディング](#)
 - ・ [CERT Cセキュアコーディングスタンダード](#)
- セキュアなソフトウェア開発を支援する資料
 - ・ [OWASP「ソフトウェアセキュリティ保証成熟度モデル」](#)

[Topへ](#)



発生元への「調整」を依頼したい

CERT C Secure Coding Standard を日本語に翻訳して公開しています

インシデント報告対応レポート、活動概要を公開

2010-12-16
冬期の長期休暇を控えて
2010/12

2010-12-06
電子メールソフトのセキュリティ設定について更新版を公開

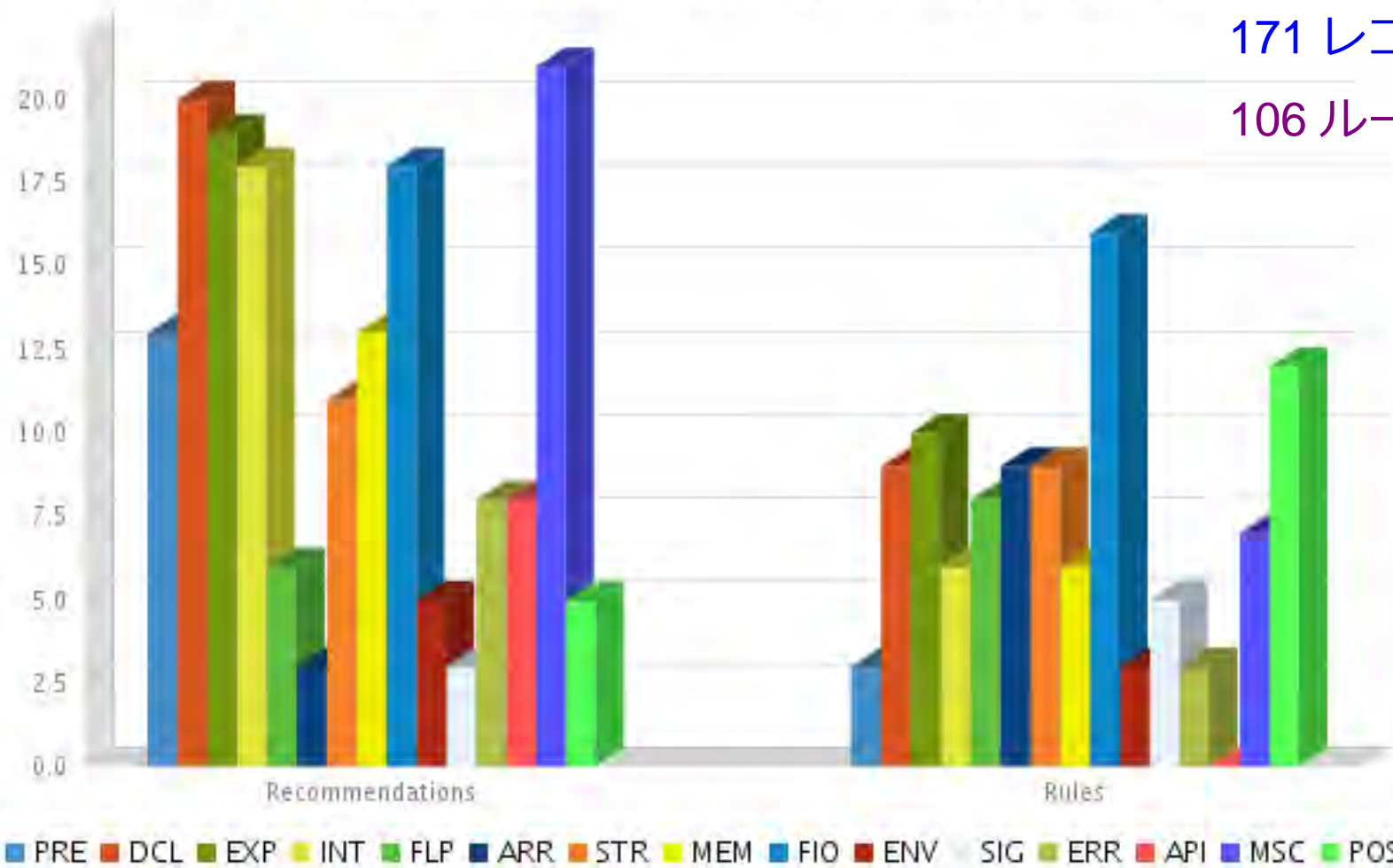
2010-11-16
マカフィー、フィッシング対策協議会ならびにJPCERT/CCと連携し、フィッシング対策機能を強化

[お知らせ一覧](#) [お知らせRSS](#)

Recommendations and Rules by Section

171 レコメンデーション

106 ルール



静的解析 (static analysis)

- 実行時のプログラムの振る舞いを予測するコンパイル時のアルゴリズム (*programming language pragmatics* の定義).
- プログラマのバイアスに左右されない. 一貫性のあるチェックをプログラム全体に適用できる.
- 早い段階で問題を発見 (プログラムの最初の実行よりも前!)
- 研究者が新たに発見した脆弱性を容易に再チェック可能.
- ノイズが多い.

型検査 (Type Checking)

スタイル検査 (Style Checking)

- lint (gccの-Wallに含まれるようなチェック)

プログラム理解 (Program Understanding)

- IDEに組み込まれている(「このメソッド呼出を全て検索」とか)
- Fujaba (www.uni-paderborn.de/cs/fujaba)

プログラム検査 (Program Verification)

- プログラムが仕様通りに実装されているか証明

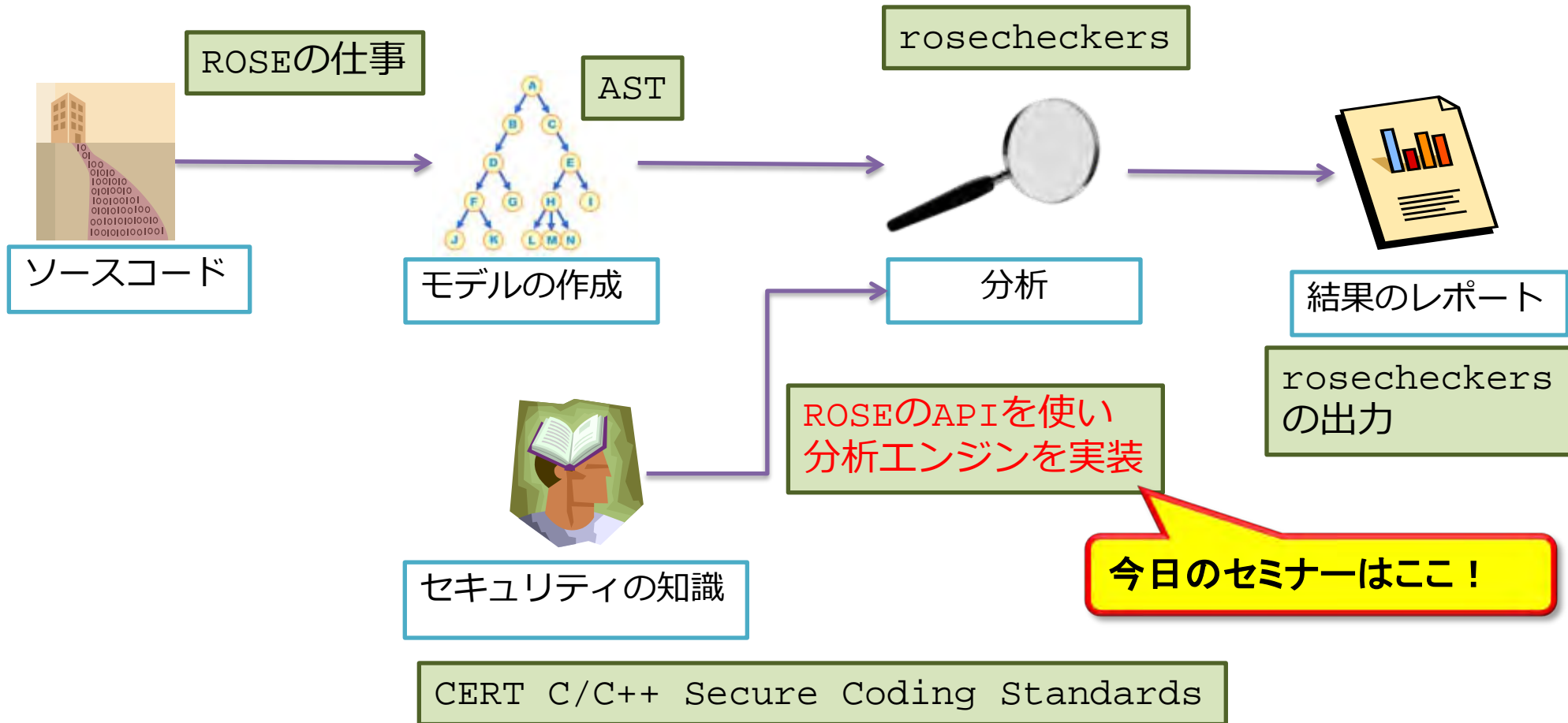
バグの検出 (Bug Finding)

- FindBugs

セキュリティチェック

- Codesurfer
- coverity
- fortify
- LDRA
- klocworks
- **Rosecheckers**

静的コード解析の概念図



抽象構文木 (AST) とは、プログラムの構文上の解析木(parse tree)を表現するデータ構造

コンパイラは、ASTを用いてソースコードをアセンブリコードに翻訳する

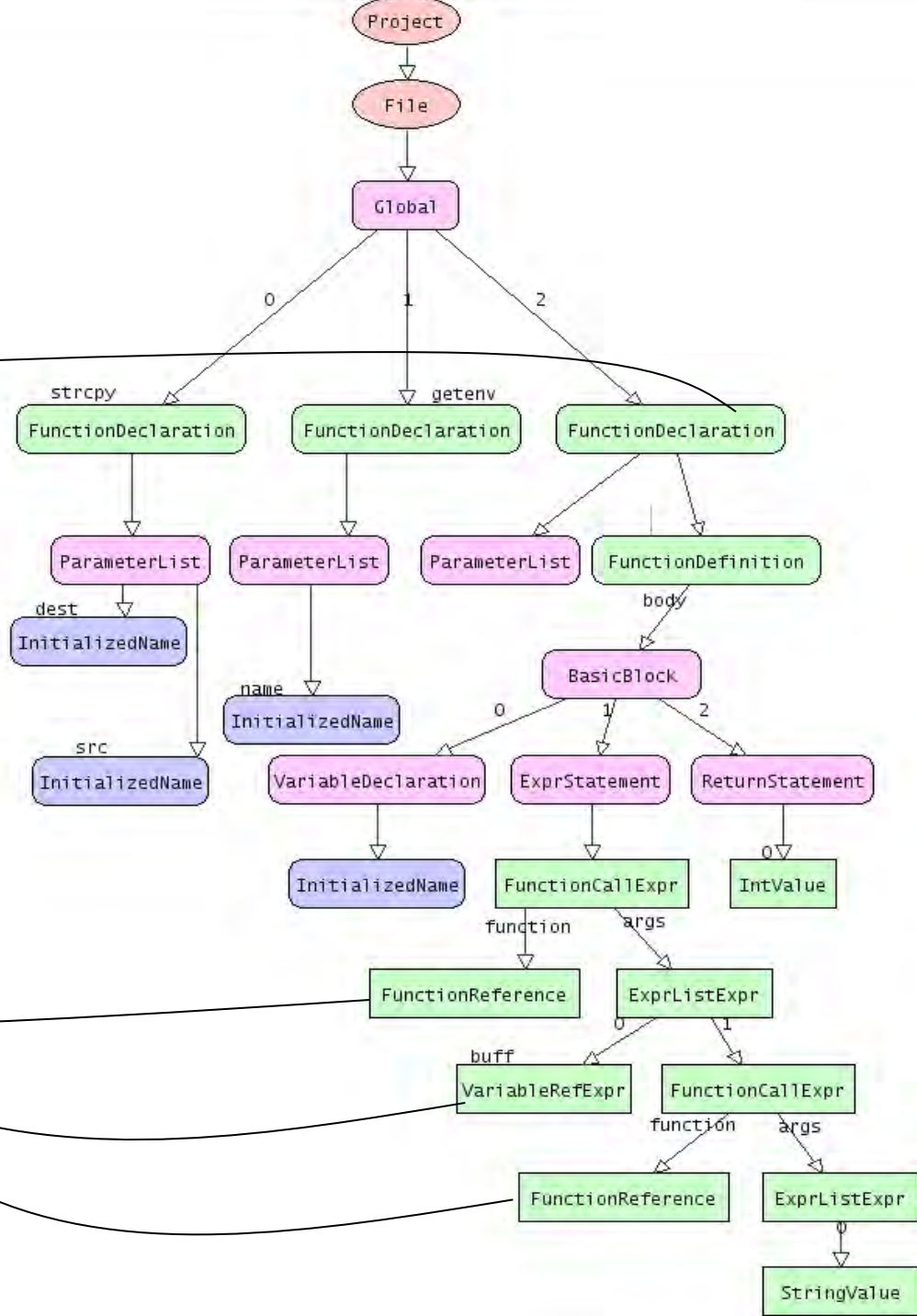
静的コード解析ツールもASTを利用

我々はASTを使ってセキュリティ上の脆弱性を見つけることができる

抽象構文木

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

```
int main() {
    /* ... */
    char buff[256];
    strcpy(buff, getenv("EDITOR"));
    /* ... */
    return 0;
}
```





Lawrence Livermore National Lab. (LLNL)で開発

- ソースコードの解析, 変換を行うためのコンパイラフレームワーク
- 抽象構文木 (AST) を生成
- 上記の機能を使えば静的解析機能を実現できる!

<http://rosecompiler.org/>

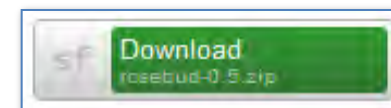


- ROSEを使ったコードチェッカー
- CERT C セキュアコーディングスタンダードのルールチェッカ
 - C++ セキュアコーディングスタンダードのチェッカ部分もあり
- CERT/CC の David Svoboda さんを中心に開発
- JPCERT/CCも開発に参加
- sourceforge.net のプロジェクトとして登録
 - <http://sourceforge.net/projects/rosecheckers/>



ROSEおよびrosecheckersインストール済みのLinux仮想イメージ.

- ROSEもrosecheckersもコンパイル済み, すぐ使える
- 仮想イメージなのでホストOSに依存しない
 - 中身は8GBディスクにXubuntuをインストールしたもの
- 主な開発ツールもインストール済み (Eclipse, emacs, etc)
- rosecheckersのサイトからダウンロードできる



C言語仕様のメジャーアップデート



JTC1/SC22/WG14 - C

- 脆弱性についてまとめたannex(付録)
- C Secure Coding Guidelines Study Groupで議論されており、CERT C Secure Coding Standardがベース
- 静的解析ツールで解析可能なコーディングガイドライン

参考: C Secure Coding Guidelines Study Group WG14 Liason Report (2010-03-16)

(<http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1450.pdf>)

- **C1x が確定するまでは、WG14のリソースを削らないよう、study group として議論する形で続ける**
- **成果は type 2 technical report としてまとめる予定**
- **C1x 以降の言語仕様にとりこむかどうかはあらためて検討**

Static Analysis Tool Exposition (SATE) 2010

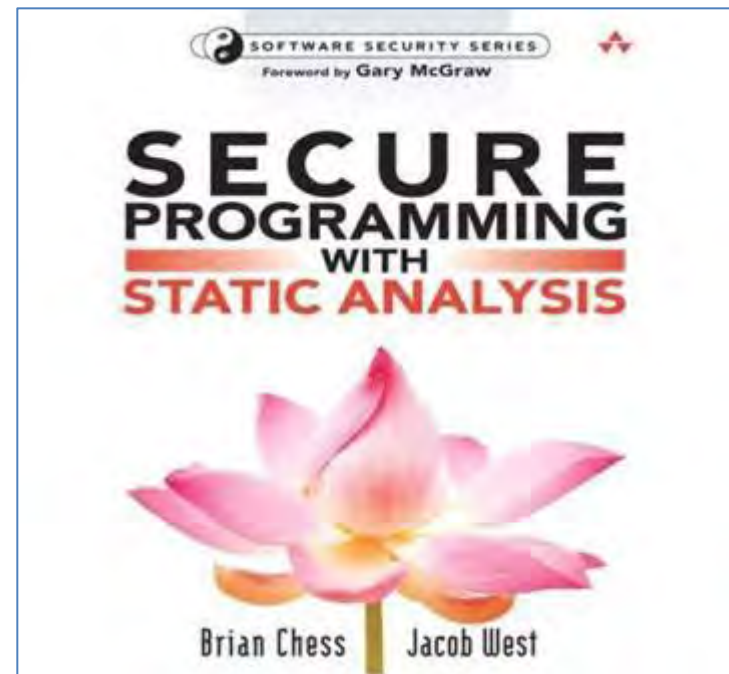
- 静的解析ツールベンダが参加
- 課題のコードを自社ツールで解析し、結果をNISTにレポート。NISTの研究者が結果を分析してレポートにまとめる。
- 2009年のレポート

http://samate.nist.gov/docs/NIST_Special_Publication_500-287.pdf

Tool	Version	Tracks
Armorize CodeSecure	3.5.9	Java
Checkmarx CxSuite	2.7.5.0	Java
Coverity Prevent	4.5.0	C
Grammatech CodeSonar	3.4p0	C
Klocwork Insight ¹	8.2	C, Java
LDRA Testbed	8.1.0	C
SofCheck Inspector for Java	2.17250, 2.18479 ²	Java
Veracode SecurityReview ³	As of 08/31/2009	C, Java



CERT C Secure Coding Standard
の日本語訳



Fortifyの創始者/チーフサイエンティスト
による静的コード解析の入門書

- rosebud のセットアップ
- ROSE のドキュメント
- rosecheckers の実行
- ASTを調べよう
- ルールチェッカーの作成: STR31-Cを例に

rosebud を使って実際に rosecheckersを試してみましよう。
また, セキュアコーディングルールのチェッカーの開発作業を
ひとつおとり体験しましょう。

rosebud のセットアップ

自分のPCに rosebud をインストールしよう



Rosebud-0.5のダウンロードと展開

rosebud を使うためには, VMWareプレイヤーが必要. また, ダウンロードしたファイルを解凍するために 7zip が必要.

www.vmware.com



www.7-zip.org



VirtualBoxでも動作します: www.virtualbox.org



Rosebud-0.5: vmware での使用



- Rosebudをダウンロードしたら7zipで解凍.
- VMプレーヤーのメニューから “*Open an Existing Virtual Machine*” を選択.
- ファイル選択ダイアログが開いたら **rosebud** ディレクトリに移動して **rosebud.vmx** を選択.
- Rosebud 仮想イメージが起動して数秒するとログインプロンプトが現れる.
- ユーザ: **rose** パスワード: **roserose** でログイン.
- Xfceのデスクトップ環境が立ち上がる.

Rosebud-0.5: VirtualBox での使用

- Rosebudをダウンロードしたら7zipで解凍
- VirtualBoxで新たに仮想マシンを作成
- ハードディスクの設定で **rosebud.vmdk** を選択.
- Rosebud 仮想イメージが起動して数秒するとログインプロンプトが現れる.
- ユーザ: **rose** パスワード: **roserose** でログイン.
- Xfceのデスクトップ環境が立ち上がる.
- Guest Additions をインストールしておくこと、X window system がホストOS上のGUI窓のサイズ変更に従ってくれて便利

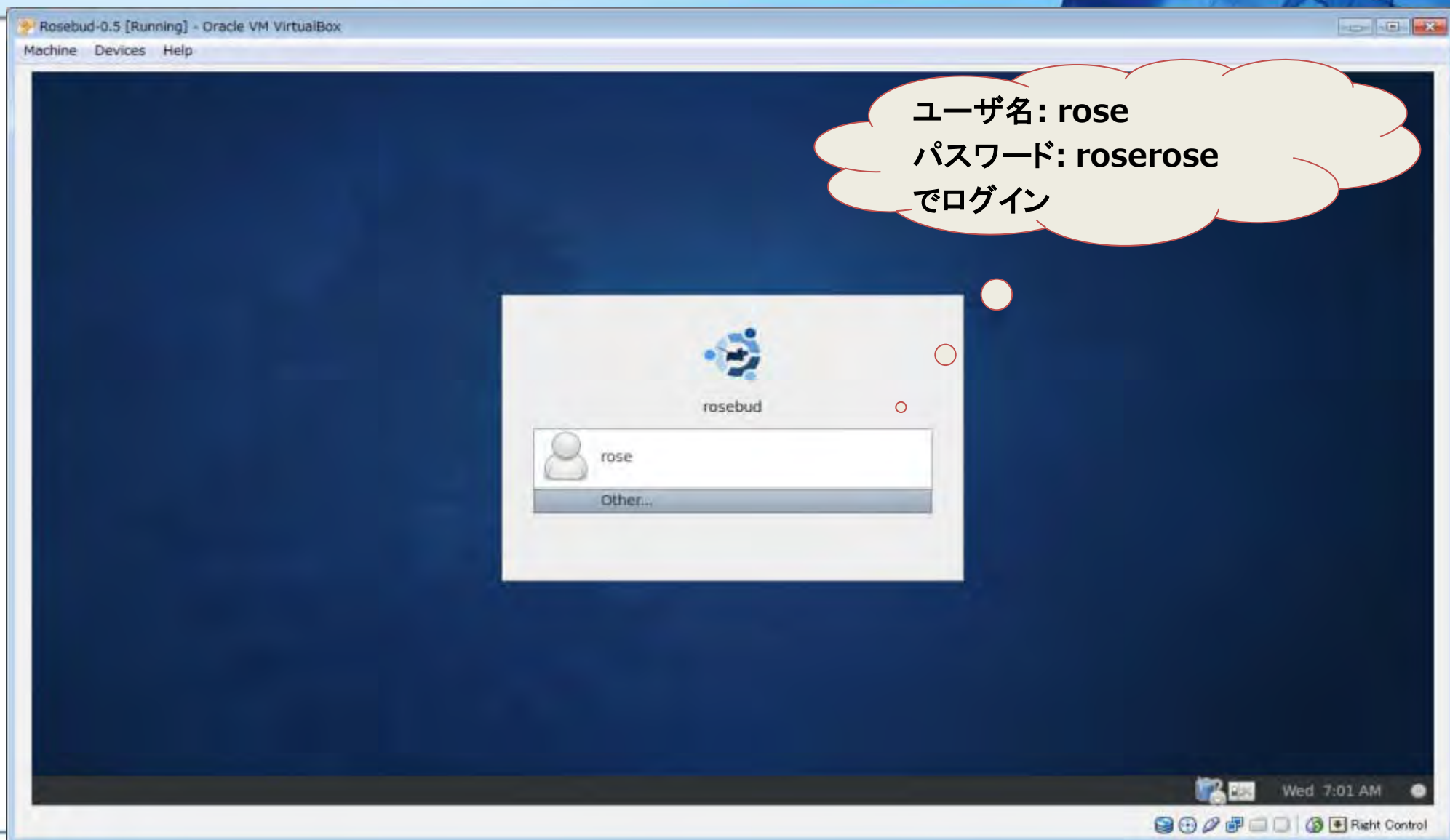


Rosebud-0.5: ディレクトリ構成

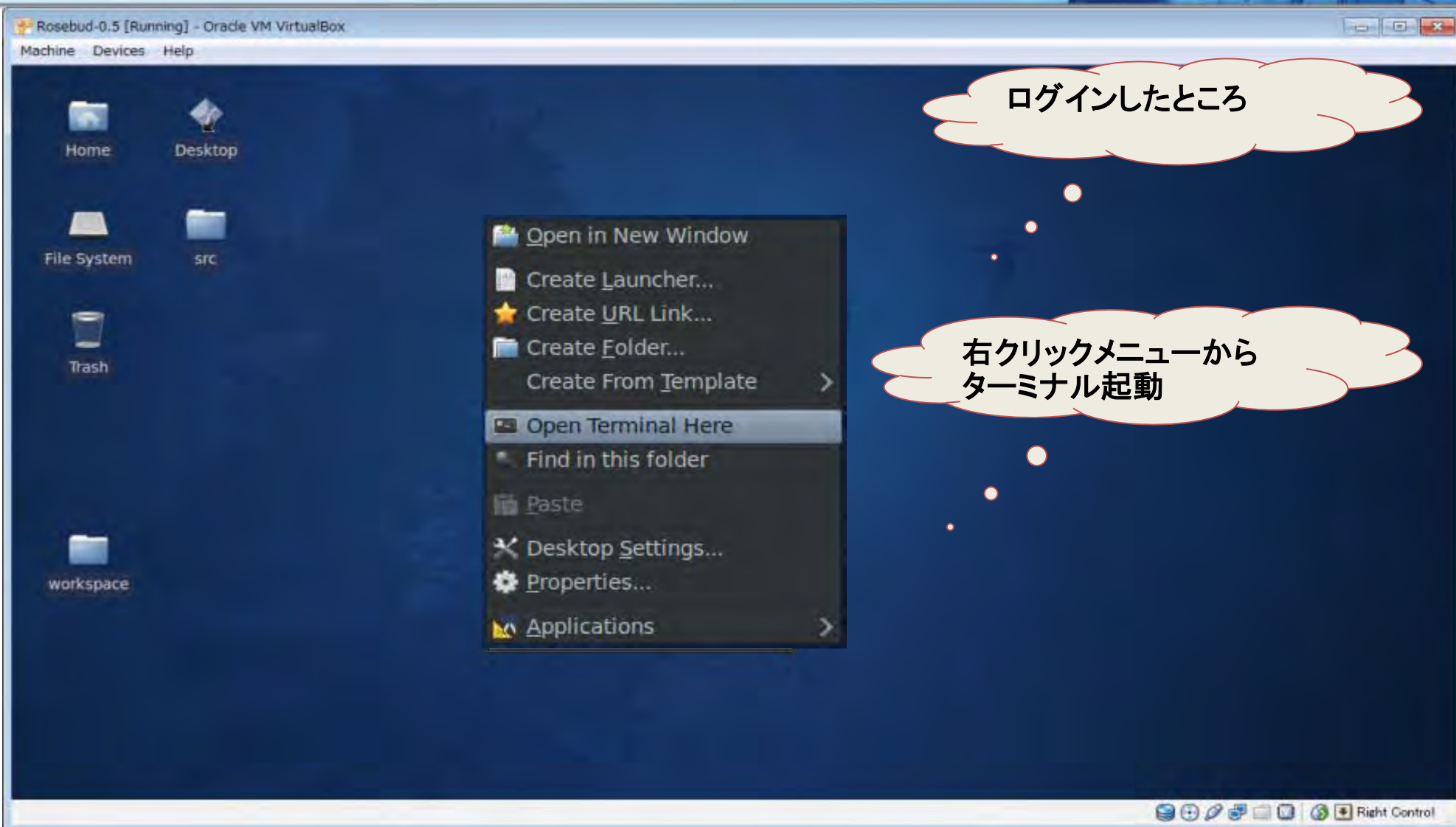
右クリックでプログラムメニューを出し、ターミナルを起動。
以降の作業は `~/workspace/` の下で行う。

- `~/src/` : boostライブラリや ROSE を置いてある
- `~/workspace/` : 作業用ディレクトリとして用意されたもの
- `~/workspace/rosecheckers/` : rosecheckers のリポジトリをチェックアウトしたもの

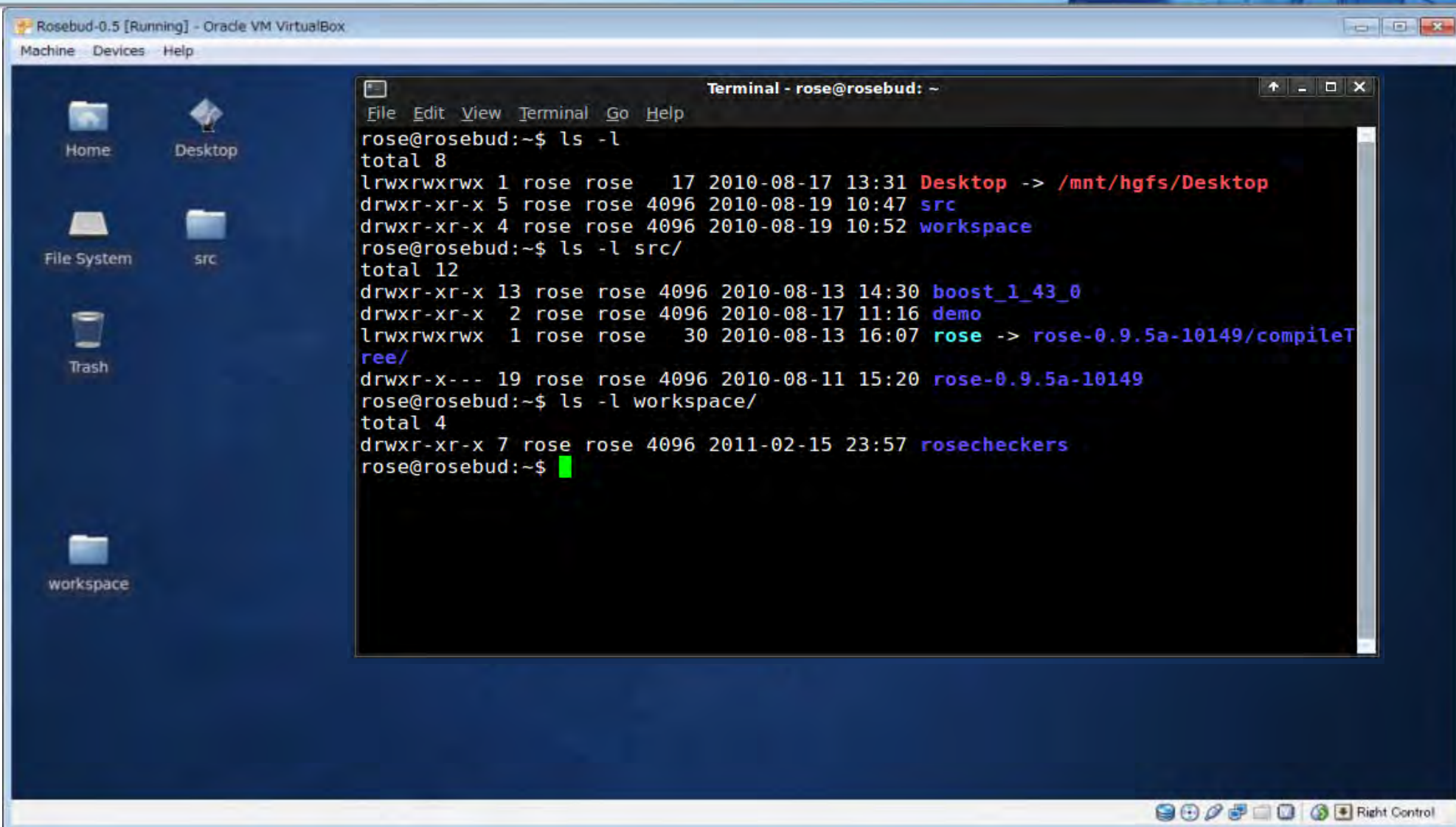
Rosebud-0.5 スクリーンショット： ログイン画面



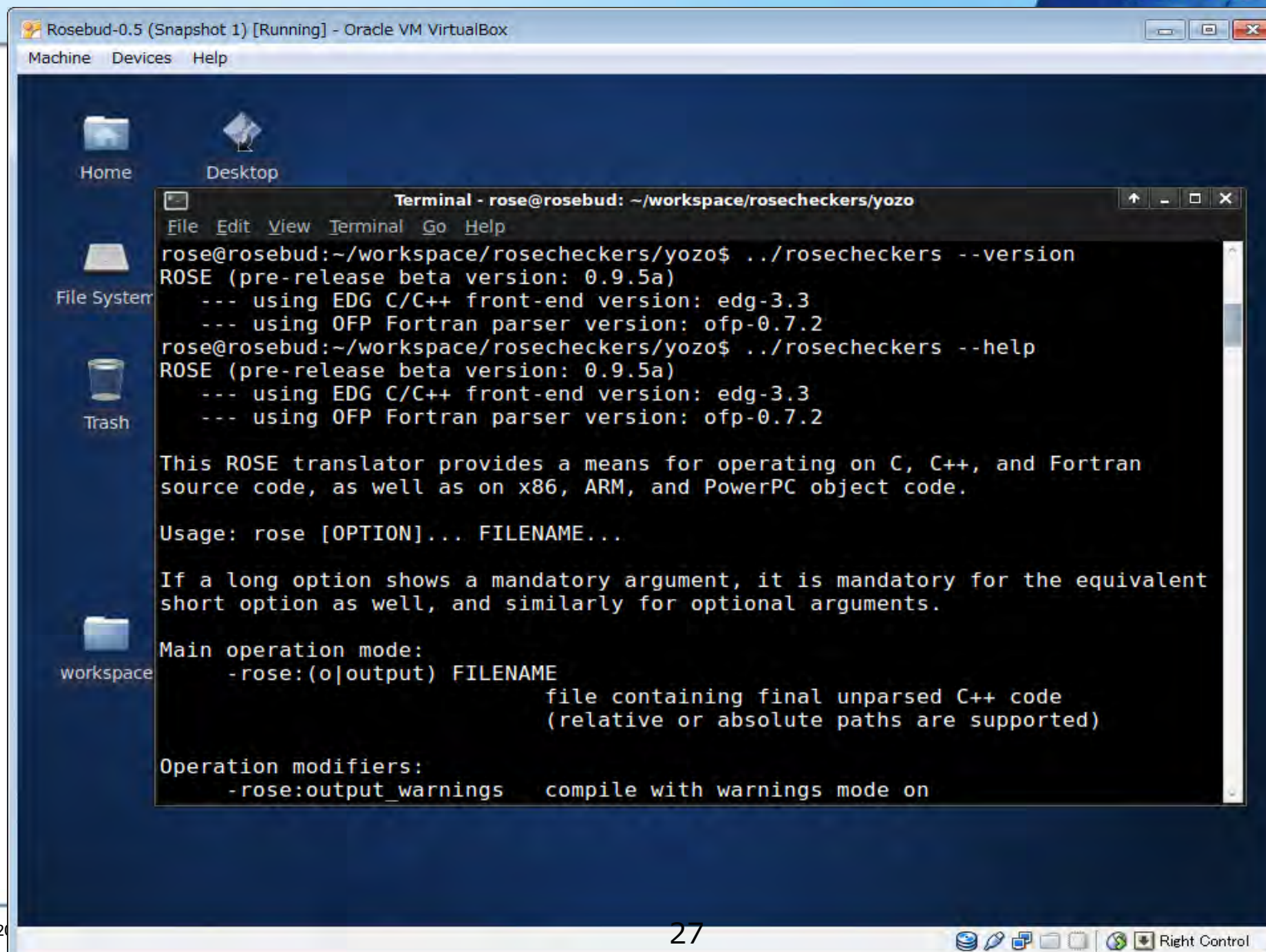
Rosebud-0.5 スクリーンショット： 右クリックのメニュー



Rosebud-0.5 スクリーンショット: ターミナルで作業



rosecheckers のヘルプメッセージで動作確認



The screenshot shows a virtual machine window titled "Rosebud-0.5 (Snapshot 1) [Running] - Oracle VM VirtualBox". The desktop environment includes icons for Home, Desktop, File System, Trash, and workspace. A terminal window is open, displaying the following output:

```
Terminal - rose@rosebud: ~/workspace/rosecheckers/yozo
File Edit View Terminal Go Help
rose@rosebud:~/workspace/rosecheckers/yozo$ ../rosecheckers --version
ROSE (pre-release beta version: 0.9.5a)
--- using EDG C/C++ front-end version: edg-3.3
--- using OFP Fortran parser version: ofp-0.7.2
rose@rosebud:~/workspace/rosecheckers/yozo$ ../rosecheckers --help
ROSE (pre-release beta version: 0.9.5a)
--- using EDG C/C++ front-end version: edg-3.3
--- using OFP Fortran parser version: ofp-0.7.2

This ROSE translator provides a means for operating on C, C++, and Fortran
source code, as well as on x86, ARM, and PowerPC object code.

Usage: rose [OPTION]... FILENAME...

If a long option shows a mandatory argument, it is mandatory for the equivalent
short option as well, and similarly for optional arguments.

Main operation mode:
-rose:(o|output) FILENAME
                                file containing final unparsed C++ code
                                (relative or absolute paths are supported)

Operation modifiers:
-rose:output_warnings  compile with warnings mode on
```

Rosebud-0.5: 追加の設定

```
export LD_LIBRARY_PATH=¥  
/usr/lib/jvm/java-6-openjdk/jre/lib/i386/server:$LD_LIBRARY_PATH
```

```
--- cpp2ps.orig 2010-08-12 16:43:18.027702841 -0400  
+++ cpp2ps 2011-02-16 03:08:40.058132022 -0500  
@@ -95,6 +95,9 @@
```

```
    if($opt_postorder) {  
        $genfile="$infilebasename.Postorder.dot";  
    }  
+ else {  
+   $genfile="$infilebasename.dot";  
+ }
```

```
$outfile=@ARGV[1];  
system("dot2ps $genfile $outfile");
```

~/src/rose/bin/cpp2ps
(perlスクリプト) にパッチ

cpp2psやcpp2pdf
を動作させるため
の修正です

ROSEに関するドキュメント

ROSEのドキュメントは英語しかありません



ROSE に関するドキュメント

[チュートリアル \(ROSE-Tutorial.pdf\)](#)

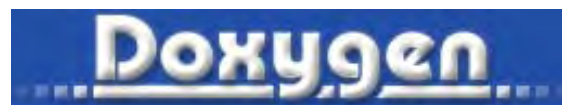
ROSEの各種機能に関するチュートリアル

[ユーザマニュアル \(ROSE-userManual.pdf\)](#)

ROSEの機能と使い方, インストールや関連ツール
に関するドキュメント

[Rose Web Reference](#)

ROSEの各クラスとメソッドのHTMLドキュメント
(Doxygenで生成)



ROSE Homepage

このリンクを辿ってドキュメント一覧ページへ

Firefox

ROSE

http://www.rosecompiler.org/

Google

Feedback

ROSE

[Privacy & Legal Notice](#)

[Home](#)

[News](#)

[Download Software](#)

[Documents](#)

[Doxygen Documentation](#)

[Qt GUI Support](#)

[Internal Projects](#)

[FAQ](#)

[Regression Tests](#)

[Report Bugs](#)

[Publications](#)

[ROSE Mailing List](#)

[Nightly External NMI Tests](#)

[News: Haskell bindings have been released.](#)

[News: OpenMP 3.0 implementation has been released.](#)

[News: Autotuning draft tutorial has been released.](#)

[News: ROSE wins R&D 100 Award!](#)

ROSE is an open source compiler infrastructure to build source-to-source program transformation and analysis tools for large-scale Fortran 77/95/2003, C, C++, OpenMP, and UPC applications. The intended users of ROSE could be either experienced compiler researchers or library and tool developers who may have minimal compiler experience. ROSE is particularly well suited for building custom tools for static analysis, program optimization, arbitrary program transformation, domain-specific optimizations, complex loop optimizations, performance analysis, and cyber-security.

Like other compiler infrastructures, ROSE consists of front-ends, a midend, and backends, but ROSE backends generate (unparse) source code. Thus ROSE is a source-to-source compiler infrastructure. The intermediate representation (IR) used in ROSE is high level to build an abstract syntax tree (AST) that is well suited to source-to-source (so ROSE does not lose any information about the structure of the original source code). The midend contains an evolving set of analyzes and optimizations. The Edison Design Group (EDG) front-end is used to parse C and C++ applications. Although the IR, source code and interfaces are protected, they may

Firefox

ROSE ROSE Documents ROSE

http://www.rosecompiler.org/documents.html

Google

Feedback

クラスやメソッドのドキュメントはこのリンクを辿る

ROSE

[Privacy & Legal Notice](#)

Please find various documents of ROSE in this page:

- [Installation Guide \(pdf\)](#)
- [Developer's Guide \(pdf\)](#)
- [User Manual \(pdf\)](#)
- [Tutorials \(pdf\)](#)
- [Doxygen Documentation ROSE \(html\)](#)
- [Doxygen Documentation for ROSE Qt Widgets \(html\)](#)
- [Doxygen Documentation for ROSE Haskell Binding API \(html\)](#)
- [Compass Manual \(pdf\)](#)
- [Autotuning Tutorials \(pdf\)](#)
- [QROSE Manual \(pdf\)](#)

[Home](#)

[News](#)

[Download Software](#)

[Documents](#)

[Doxygen Documentation](#)

[Qt GUI Support](#)

[Internal Projects](#)

[FAQ](#)

[Regression Tests](#)

[Report Bugs](#)

[Publications](#)

[ROSE Mailing List](#)

ROSE Web Reference(1)

The screenshot shows a Firefox browser window displaying the ROSE Web Reference website. The address bar shows the URL http://www.rosecompiler.org/ROSE_HTML_Reference/index.l. The page features a navigation menu on the left with items like "ROSE Web Reference", "Modules", "Class List", "Class Hierarchy", "Class Members", "Graphical Class Hierarchy", "Namespace List", "Namespace Members", "File List", "Examples", "File Members", and "Related Pages". The main content area has tabs for "Main Page", "Modules", "Namespaces", "Classes", "Files", "Related Pages", and "Examples". The title "ROSE Web Reference" is prominently displayed, followed by the version "0.9.5a". Below the title is a logo featuring a compass rose with a rose in the center and the letters "ROSE" in a stylized font. The "Authors:" section lists LLNL Staff (Dan Quinlan, Thomas Panas, Chunhua Liao), former LLNL Post-docs (Jeremiah Willcock, Markus Schordan, Qing Yi, Rich Vuduc), and Student Interns at LLNL (Gergo Barany, Michael Byrd, Gabriel Coutinho, Peter Collingbourne, Valentin David, Jochen Haerdlein, Vera Hauge, Christian Iwainsky, Lingxiao Jiang, Alin Jula, Han Kim, Milind Kulkarni, Markus Kowarschik, Gary Lee, Chunhua Liao, Peter Pirkelbauer, Bobby Philip, Radu Popovici, Robert Preissl, Andreas Saebjornsen, Sunjeev Sikand, Andy Stone, Danny Thorne, Nils Thuerey, Ramakrishna Upadrasta, Christian Wiess, and Jeremiah).

Firefox

ROSE

ROSE Documents

ROSE

http://www.rosecompiler.org/ROSE_HTML_Reference/index.l

Google

ROSE

Main Page Modules Namespaces Classes Files Related Pages Examples

Search for

ROSE Web Reference

0.9.5a

Authors:

LLNL Staff: **Dan Quinlan, Thomas Panas, Chunhua Liao**
Former LLNL Post-docs (most of them are still active on ROSE project): **Jeremiah Willcock, Markus Schordan, Qing Yi, and Rich Vuduc**
Student Interns at LLNL: Gergo Barany (Technical University of Vienna), Michael Byrd (University of California at Davis), Gabriel Coutinho (Imperial College London), Peter Collingbourne (Imperial College London), Valentin David (University of Bergen, Norway), Jochen Haerdlein (University of Erlanger, Germany), Vera Hauge (University of Oslo, Norway), Christian Iwainsky (University of Erlanger, Germany), Lingxiao Jiang (University of California at Davis), Alin Jula (Texas A&M), Han Kim (University of California at San Diego), Milind Kulkarni (Cornell University), Markus Kowarschik (University of Erlanger, Germany), Gary Lee (University of California at Berkeley and Purdue University), Chunhua Liao (University of Houston), Ghassan Mishergahi. (University of California at Davis), Peter Pirkelbauer (Texas A&M), Bobby Philip (University of Colorado), Radu Popovici (Cornell University), Robert Preissl (Austria), Andreas Saebjornsen (University of Oslo, Norway), Sunjeev Sikand (University of California at San Diego), Andy Stone (Colorado State University at Fort Collins), Danny Thorne (University of Kentucky), Nils Thuerey (University of Erlanger, Germany), Ramakrishna Upadrasta (Colorado State University at Fort Collins), Christian Wiess(Munich University of Technology, Germany), Jeremiah

Scripts Currently Forbidden | <SCRIPT>: 1 | <OBJECT>: 0

Options... x

ROSE Web Reference (2)

File Edit View History Bookmarks Tools Help

file:///home/svoboda/Desktop/Dc Google

[Main Page](#) | [Modules](#) | [Namespace List](#) | [Class Hierarchy](#) | [Class List](#) | [File List](#) | [Namespace Members](#) | [Class Members](#) | [File Members](#) | [Related Pages](#)

SgIfStmt Class Reference

```
#include <Cxx_Grammar.h>
```

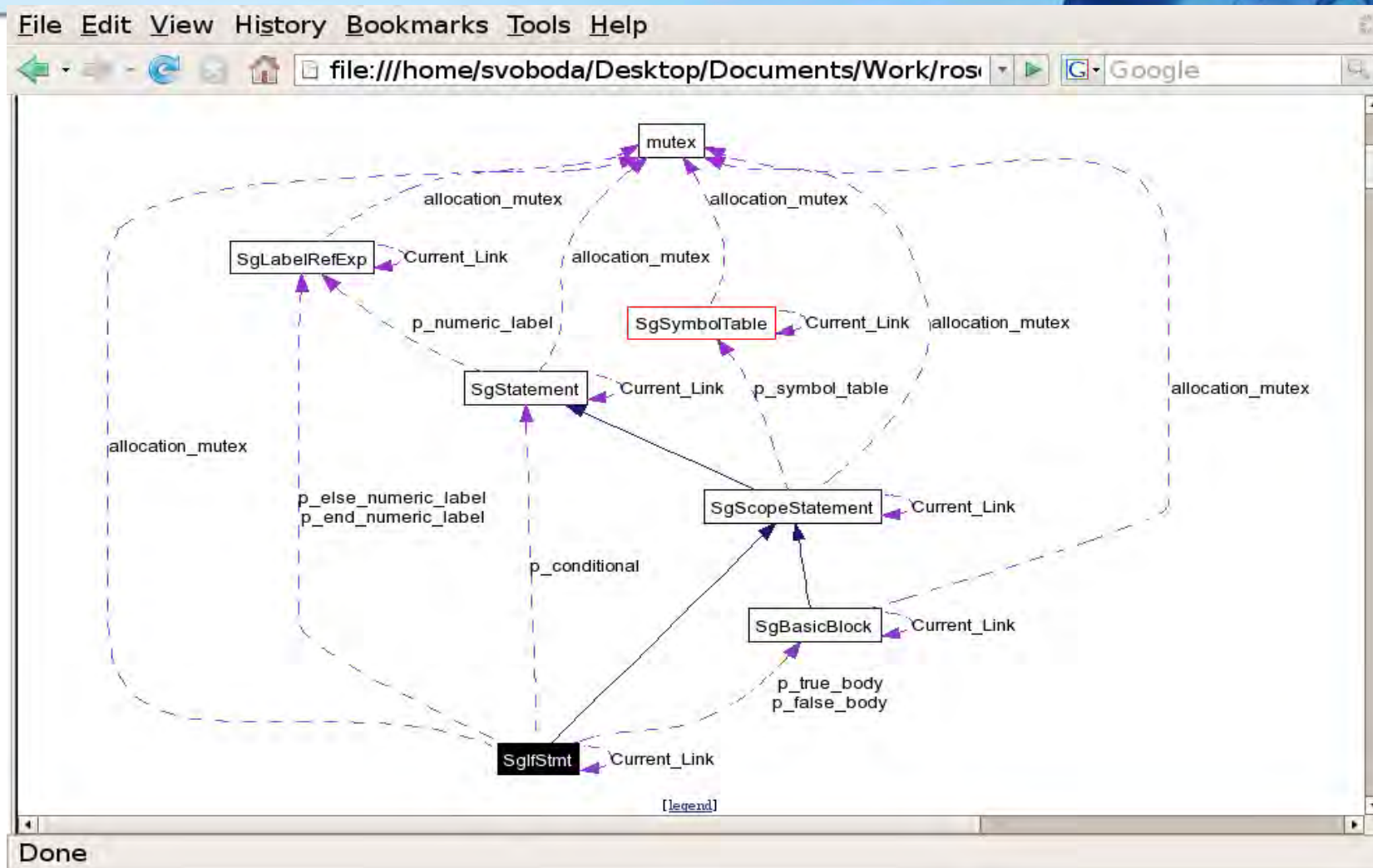
Inheritance diagram for SgIfStmt:

```
graph BT; SgNode --> SgLocatedNode; SgLocatedNode --> SgStatement; SgStatement --> SgScopeStatement; SgScopeStatement --> SgIfStmt;
```

Collaboration diagram for SgIfStmt: [\[Legend\]](#)

Done

ROSE Web Reference (3)



Done

ROSE Web Reference (4)

The screenshot shows a web browser window with the address bar containing the file path `file:///home/svoboda/Desktop/Dc`. The browser's address bar also includes a search engine icon and the text "Google". The main content area displays the documentation for the `SgBasicBlock` class, which is part of the `SgBasicBlock` namespace. The left sidebar shows a list of classes, with `SgBasicBlock` selected. The main content area shows the following code snippets:

```
void set_conditional (SgStatement *conditional)
Access function for p_conditional. See *conditional) condition
for documentation.

SgBasicBlock * get_true_body () const
Access function for p_true_body. See const true_body for
documentation.

void set_true_body (SgBasicBlock *true_body)
Access function for p_true_body. See *true_body) true_body
documentation.

SgBasicBlock * get_false_body () const
Access function for p_false_body. See const false_body for
documentation.

void set_false_body (SgBasicBlock *false_body)
Access function for p_false_body. See *false_body) false_body
documentation.

std::string get_string_label () const
void set_string_label (std::string string_label)

SgLabelRefExp * get_end_numeric_label () const

virtual ~SgIfStmt ()
This is the destructor. There are a lot of things to delete, but
nothing is deleted in this destructor.

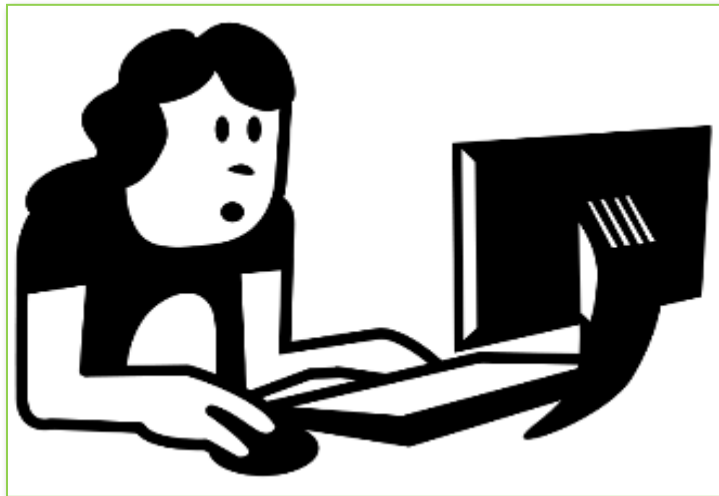
SgIfStmt (Sg_File_Info *startOfConstruct, SgStatement
*conditional=NULL, SgBasicBlock *true_body=NULL,
SgBasicBlock *false_body=NULL)

SgIfStmt (SgStatement *conditional, SgBasicBlock
```

The browser's status bar at the bottom shows the word "Done".

rosecheckersの実行

Emacsのflymakeモード使うと便利, かも



rosecheckersの実行

チェッカーの使い方はコンパイラや lint などと同じ。

違反コード例に対してチェッカー (**rosecheckers**) が出力する
エラーメッセージ例:

```
% ./rosecheckers test/c.ncce.wiki.STR.c
c.ncce.wiki.STR.c:7: error: STR31-C: String copy
destination must contain sufficient storage
%
```

適合コード例に対してはチェッカーはなにも出力しない:

```
% ./rosecheckers test/c.cce.wiki.STR.c
%
```

テストコード

チェッカーの動作確認に使うため、
~/workspace/rosecheckers/test/ ディレクトリに
適合コード例と違反コード例が置いてある。

- 適合コード例(**c**ompliant **c**ode **e**xample):
 - セキュアコーディングルールに適合したコード
- 違反コード例(**n**on **c**ompliant **c**ode **e**xample):
 - セキュアコーディングルールに違反したコード

コーディングルールのページに掲載されているコード例
だが、最新版に追従してはるわけではない……

Emacs から rosecheckers を使う (1)

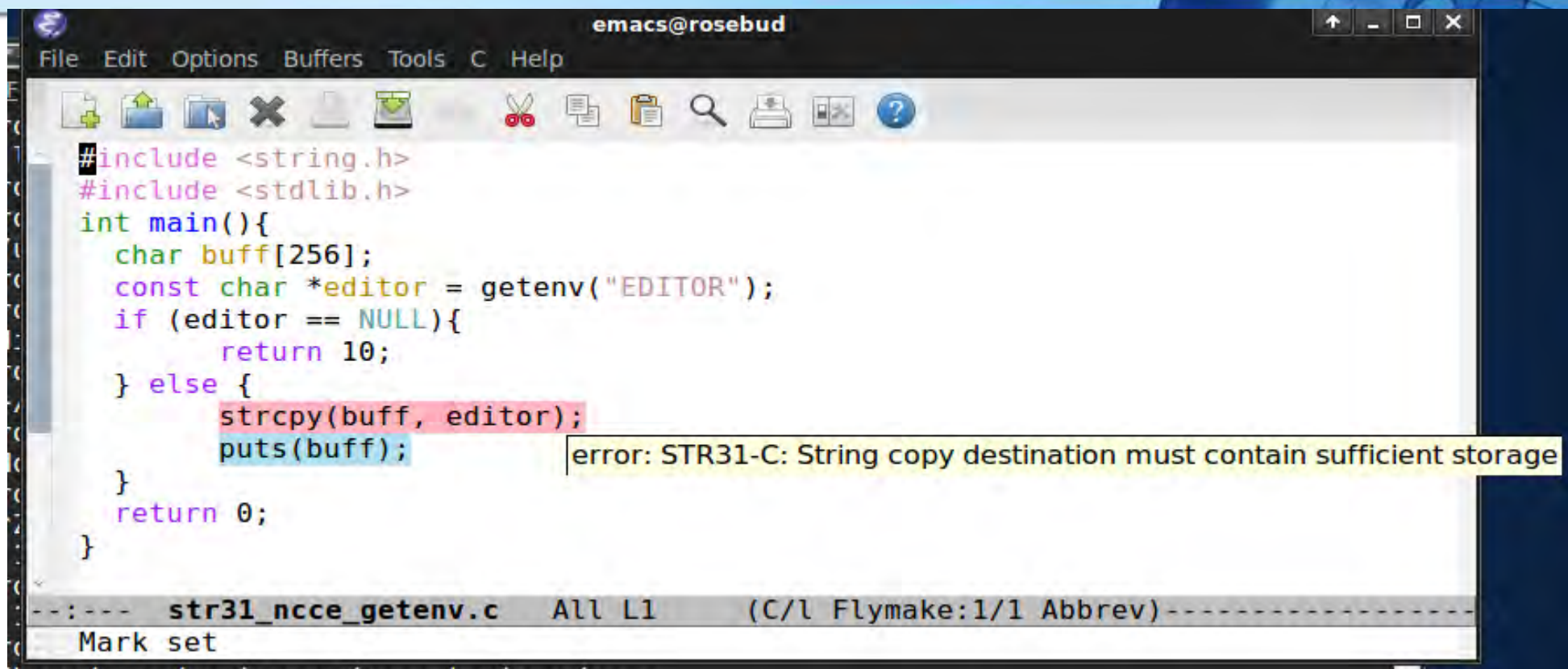
flymake モードを使って、編集作業しながら rosecheckers によるコードチェックを行わせることができる

```
int main() {  
    /* ... */  
    char buff[256];  
    strcpy(buff, getenv("EDITOR"));  
    /* ... */  
}
```

error: STR31-C: String copy destination must contain sufficient storage

```
-- STR31_C_getenv.c Bot (12,0) (C/1 Flymake:1/0 Abbrev) --  
STR31_C_getenv.c: 1 error(s), 0 warning(s) in 0.27 second(s)
```

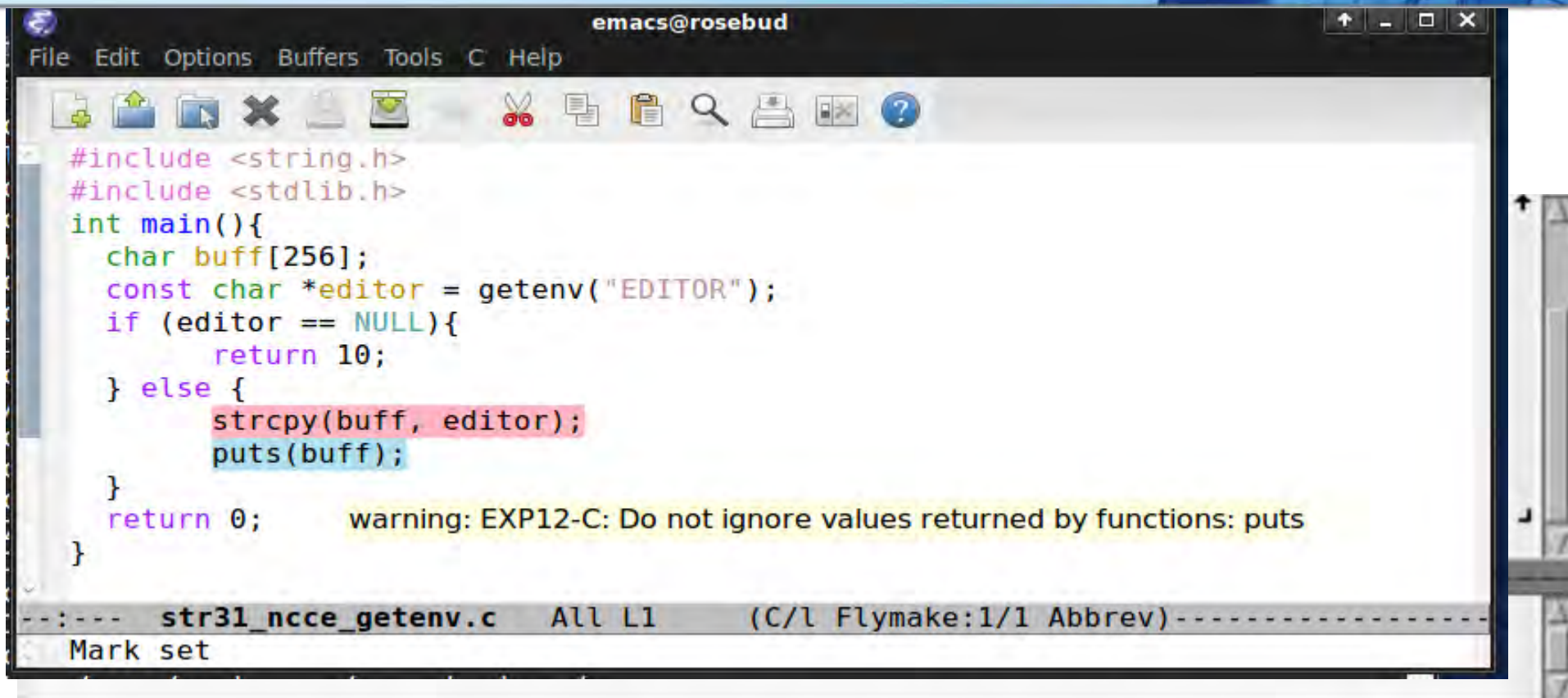
Emacs から rosecheckers を使う (2)



```
emacs@rosebud
File Edit Options Buffers Tools C Help
#include <string.h>
#include <stdlib.h>
int main(){
    char buff[256];
    const char *editor = getenv("EDITOR");
    if (editor == NULL){
        return 10;
    } else {
        strcpy(buff, editor);
        puts(buff);
    }
    return 0;
}
error: STR31-C: String copy destination must contain sufficient storage
str31_ncce_getenv.c All L1 (C/l Flymake:1/1 Abbrev)
Mark set
```

エラーの部分は赤くなり、マウスカーソルを持っていくとメッセージが表示される。

Emacs から rosecheckers を使う (3)



```
emacs@rosebud
File Edit Options Buffers Tools C Help
#include <string.h>
#include <stdlib.h>
int main(){
    char buff[256];
    const char *editor = getenv("EDITOR");
    if (editor == NULL){
        return 10;
    } else {
        strcpy(buff, editor);
        puts(buff);
    }
    return 0;
}
warning: EXP12-C: Do not ignore values returned by functions: puts
str31_nce_getenv.c All L1 (C/l Flymake:1/1 Abbrev)
Mark set
```

warningの部分は青くなり、マウスカーソルを持っていくとメッセージが表示される。

Emacs から rosecheckers を使う (4)

flymake の info §4.2.1 Example にある perl の例を下敷きに設定

```
(defun flymake-rosecheckers-init ()
  (let* ((temp-file (flymake-init-create-temp-buffer-copy
                     'flymake-create-temp-inplace))
         (local-file (file-relative-name
                      temp-file
                      (file-name-directory buffer-file-name))))
    (list "/home/rose/workspace/rosecheckers/rosecheckers" (list "" local-file))))
(setq flymake-allowed-file-name-masks
      (cons ('(.+¥¥.c$"
              flymake-rosecheckers-init
              flymake-simple-cleanup
              flymake-get-real-file-name)
            flymake-allowed-file-name-masks))
(setq flymake-err-line-patterns
      (cons ('("¥¥(. *¥¥) at ¥¥([ ^ ¥n]+¥¥) line ¥¥([0-9]+¥¥)[, .¥n]"
              2 3 nil 1)
            flymake-err-line-patterns))
```

簡単なコードでrosecheckersを実行

```

Rosebud-0.5 (Snapshot 1) [Running] - Oracle VM VirtualBox
Machine  Devices  Help

Terminal - rose@rosebud: ~/workspace/rosecheckers/yozo
File Edit View Terminal Go Help
rose@rosebud:~/workspace/rosecheckers/yozo$ cat > choi.c
#include <stdio.h>
int main(){
    char *src = "abcdef";
    char dst[3];
    strcpy(dst,src);
    printf("%s\n", dst);
}
rose@rosebud:~/workspace/rosecheckers/yozo$ ../rosecheckers choi.c
choi.c:5: error: STR31-C: String copy destination must contain sufficient storage
rose@rosebud:~/workspace/rosecheckers/yozo$ cat > chochoi.c
#include <stdio.h>
int main(){
    char *src = "abcdef";
    char dst[7];
    strcpy(dst,src);
    dst[7] = '\0';
    printf("%s\n", dst);
}
rose@rosebud:~/workspace/rosecheckers/yozo$ ../rosecheckers chochoi.c
"/home/rose/workspace/rosecheckers/yozo/chochoi.c", line 6: warning: subscript
    out of range
        dst[7] = '\0';
        ^
chochoi.c:5: error: STR31-C: String copy destination must contain sufficient storag
e
rose@rosebud:~/workspace/rosecheckers/yozo$ █
    
```

ここまででできたこと

- Rosebudをインストールしてログイン
- rosecheckers によるコードチェック
- Emacsのflymakeモードの活用



rosecheckersを使ってエクササイズ

rosecheckersに親しみましょう



rosecheckersの使い心地を確かめるため、簡単な課題をいくつか用意しました。

- コマンドラインからの使用
- どのような出力が出るのか
- 検知できる違反コード例
- False positive, false negative

Emacs からの使用

Vim や eclipse からの使用

Hello world プログラム

```
#include <stdio.h>
int main(){
    printf("hello!¥n");
}
```

文字配列の宣言で, サイズ指定と文字列リテラルによる初期化の両方を行っているコード例(STR36-C)

```
#include <stdio.h>
int main(){
    char a[4] = "abc";
    printf("hello!¥n");
}
```

変数を宣言するだけで使っていないコード例

```
#include <stdio.h>
int main(){
    int i=0;
    printf("hello!¥n");
}
```

オーバーフローが発生するコード例(STR31-C)

```
#include <string.h>
int main(){
    char src[]="abcdef";
    char dst[6];
    strcpy(dst,src);
}
```

オーバーフローが発生するコード例その2(STR31-C)

```
#include <string.h>
int main(){
    char src[]="abcdef";
    char dst[6];
    for(i=0;i<sizeof(src);i++){
        dst[i]=src[i];
    }
    printf("%s\n", dst);
}
```

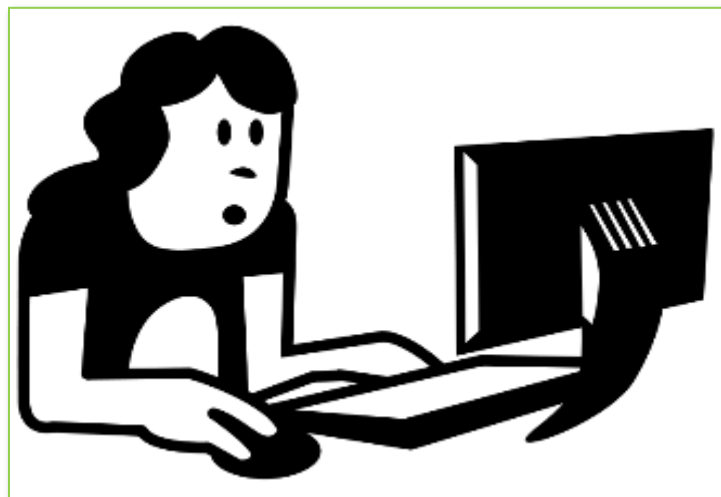
free() の使い方に関するコード例(MEM34)

```
struct list {
    int I;
    struct list *next;
};
int testalloc(struct list *head){
    if(!head) return(-1);
    struct list *lhead;
    struct list *lnext;
    for(lhead=head; lhead!=NULL; lhead=lnext){
        lnext=lhead->next;
        free(lhead);
    }
    return(0);
}
```

- gccと同じように使う
- コンパイルできる状態のコードでないと使えない
- ルールチェッカーは違反コードを検知してメッセージを出力
- ルールチェッカーの前にパーザによるチェックがある
- セキュアコーディングスタンダードの違反コード例でも, 検知できるものとできないものがある(false negative)
- 適合コード例で違反として検知することもあるかも(false positive)

ASTを調べよう

rosecheckersが扱うASTってどんなもの？



ソースコード `foo.c` の `Abstract Syntax Tree` (AST) を
PostScript形式のファイル `foo.ps` へ出力

```
cpp2ps foo.c foo.ps
```

ps, pdf は evince で表示できます

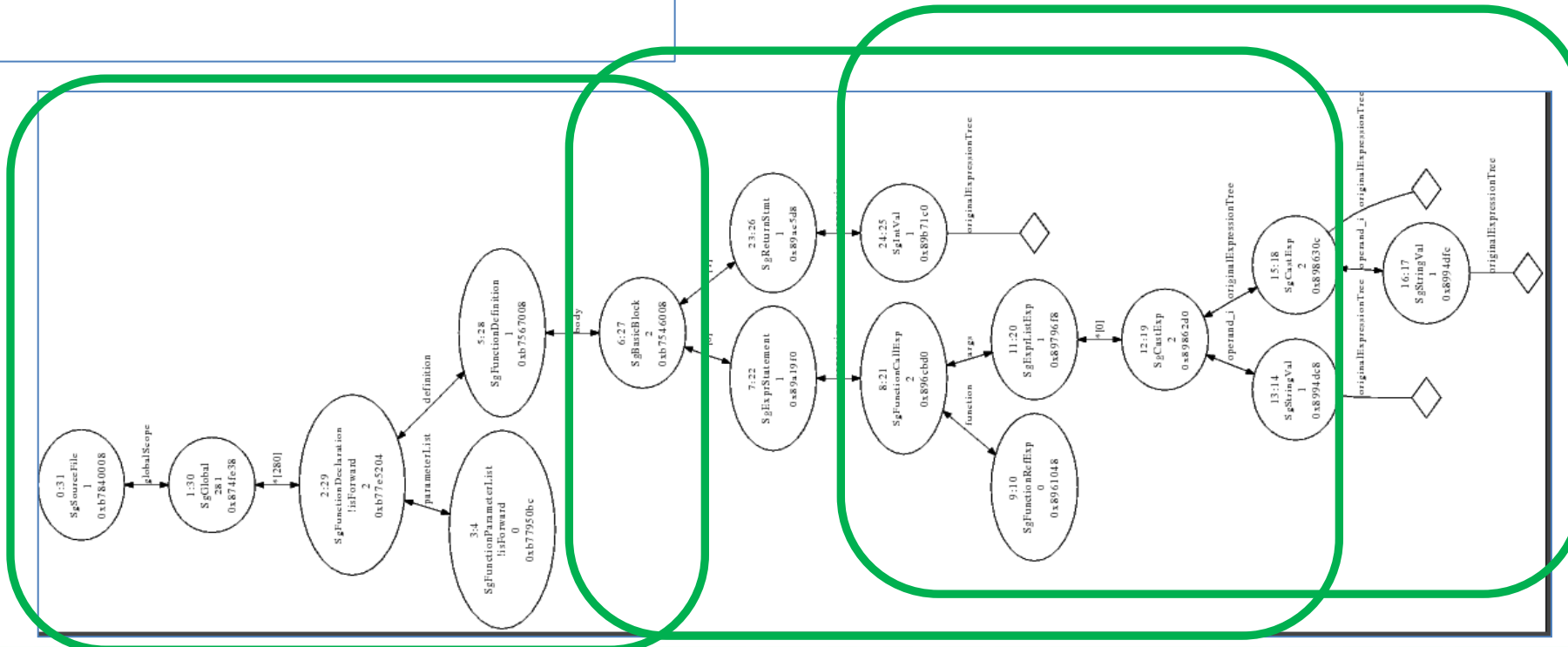


ASTの構造(hello.c その1)

コード例

```
#include <stdio.h>
int main(){
    printf("hello!¥n");
}
```

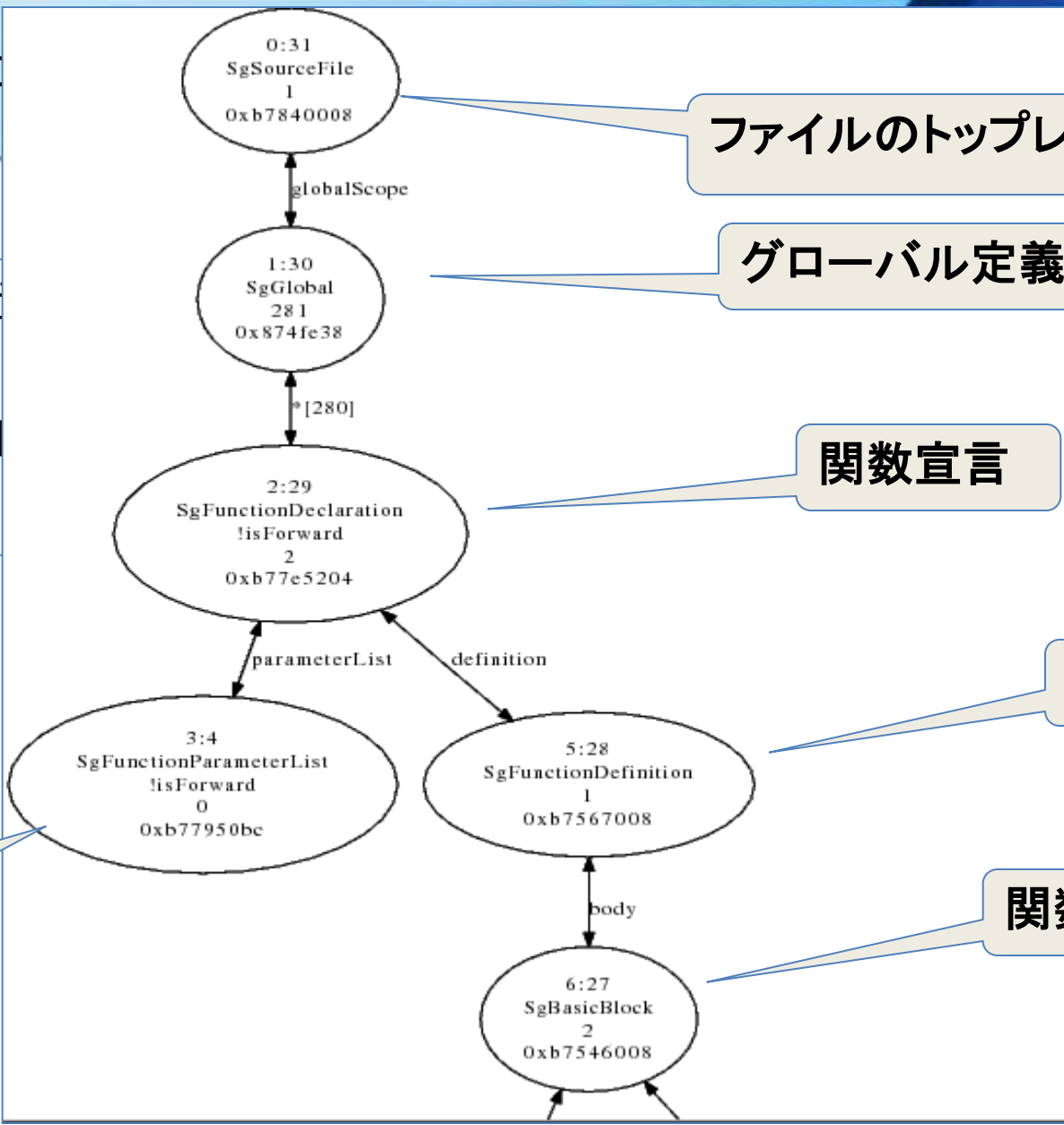
ファイルのなかに
main関数の定義



ASTの構造(hello)

コード例

```
#include <stdio.h>
int main(){
    printf("hello");
}
```



ファイルのトップレベル

グローバル定義

関数宣言

関数定義

関数本体

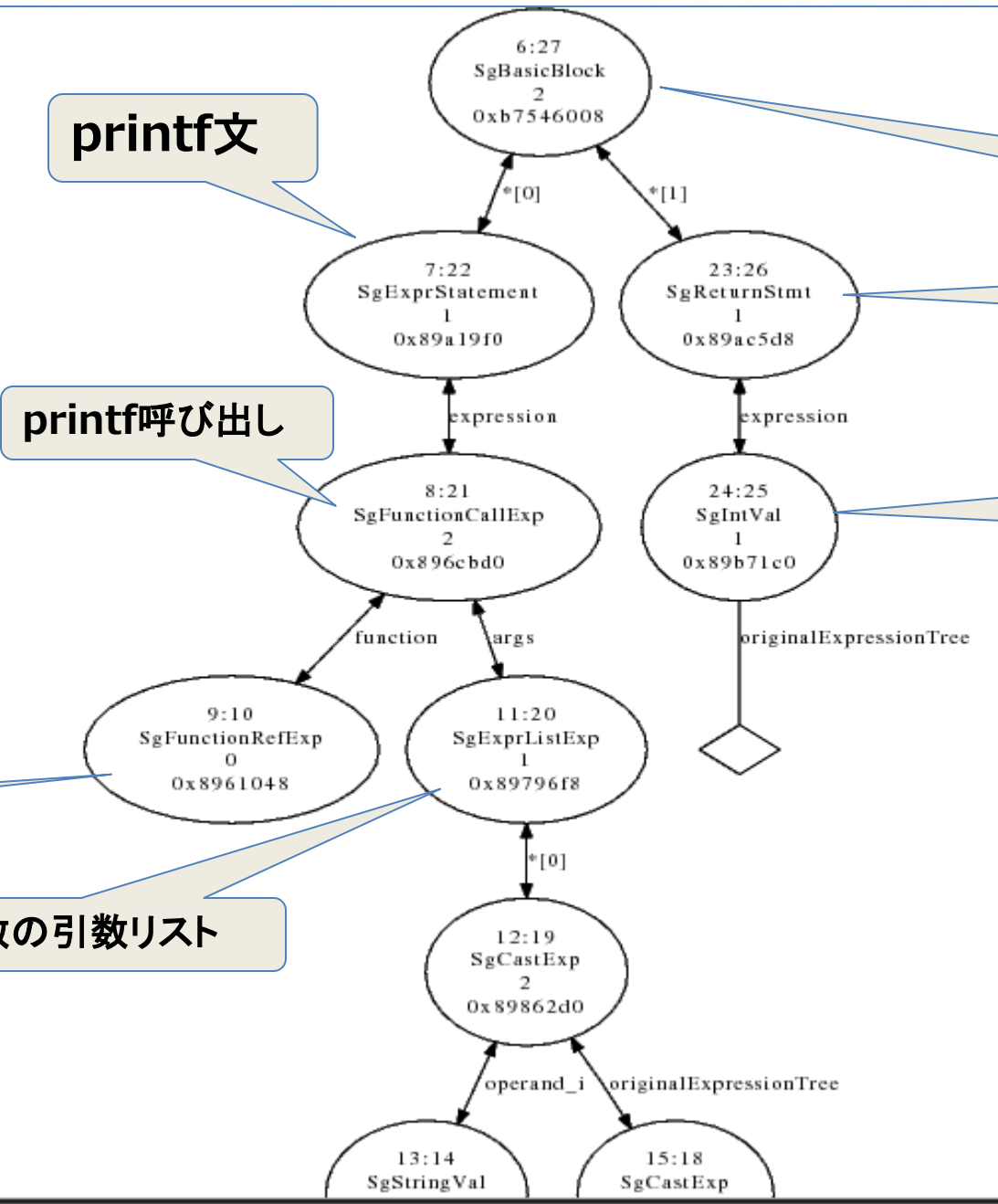
関数の引数リスト

ASTを調べよう

ASTの構造(hello)

コード例

```
#include <stdio.h>
int main(){
    printf("hello!¥n");
}
```



printf文

関数

return文

printf呼び出し

return文の引数

printf関数への参照

printf関数の引数リスト

ASTを調べよう

ASTの構造(hello.c)

printf呼び出し

return
引数

コード例

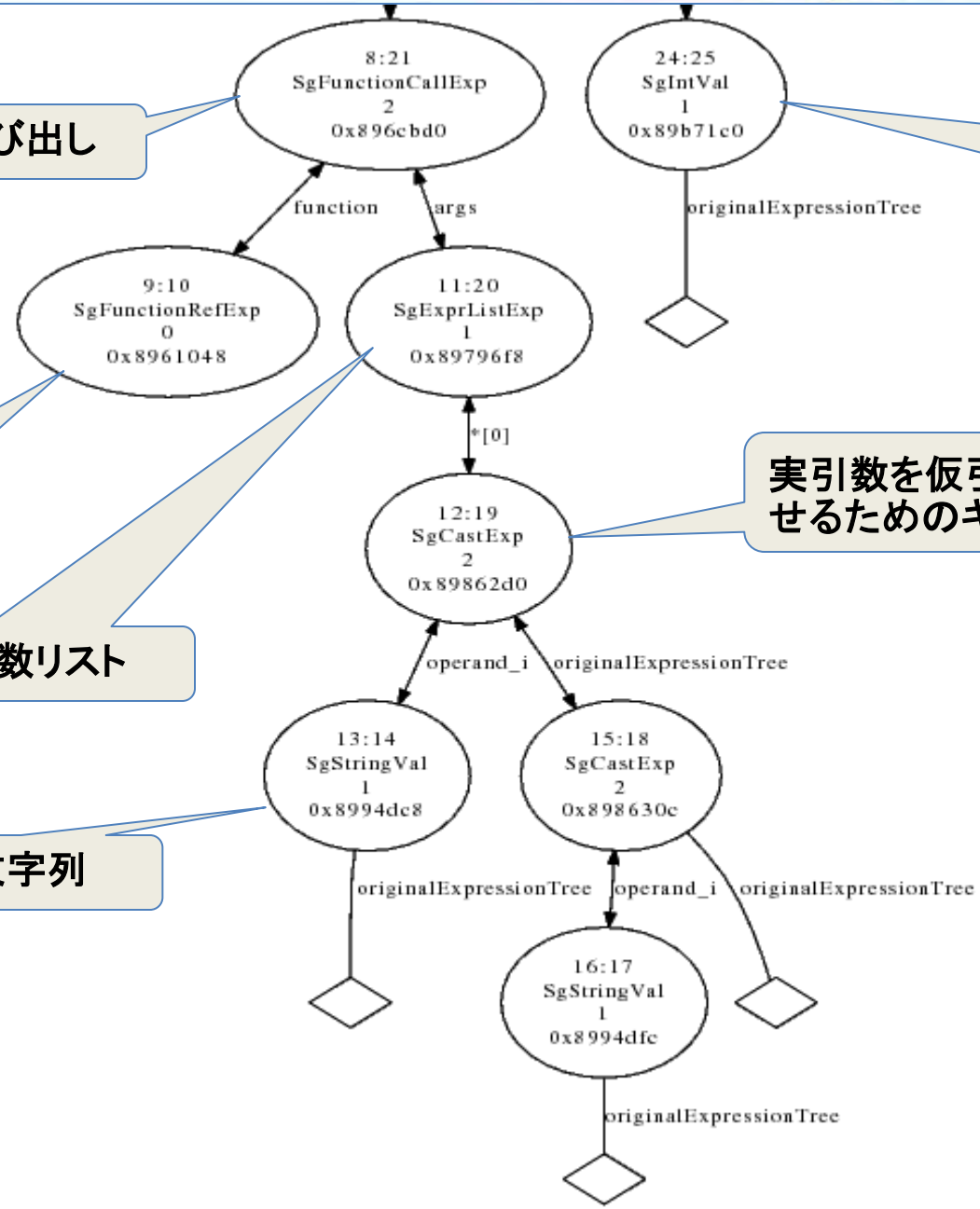
```
#include <stdio.h>
int main() {
    printf("hello\n");
}
```

printf関数への参照

printf関数の引数リスト

引数の文字列

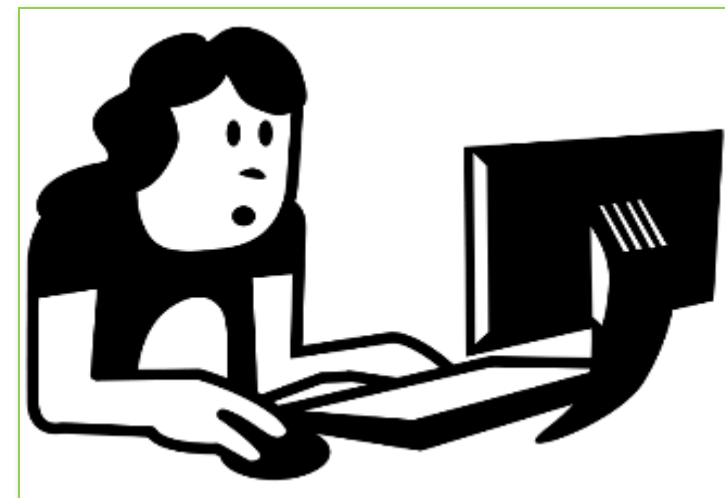
実引数を仮引数の型に合わせるためのキャスト式



エクササイズ: ASTの構造を調べる(1)

以下のコード例では, どのようなASTになるか?

- 関数呼び出し
- 算術式
- 代入文
- 変数宣言
- キャスト式
- `if else`文
- `for`文
- `while`文
- `do while` 文



エクササイズ: ASTの構造を調べる(2)

- ひとつのファイルに複数の関数定義がある場合, どのようなASTになるか
- 変数宣言のノード `SgInitializedName` の `p_name` 属性の値を調べよ.



ノードの属性値を調べる

ASTを
調べよう

ソースコード `foo.c` の AST の各ノードの属性値を
PDF形式のファイル `foo.pdf` へ出力

```
cpp2pdf foo.c foo.pdf
```

ps, pdf は evince で表示できます



ルールチェッカーの作成

自分でチェッカーを作ってみよう



ルールチェッカーの作成

STR31-C のチェッカーを作成しよう!

ルール STR31-C のチェッカーは
rosecheckers/STR.C のなかの STR31_C() で実装されています。

この関数の実装過程を見ていきます。

07. 文字と文字列 (STR) 最終更新: 2010-09-30

検索

トップページ

情報提供

- 注意喚起
- 早期警戒
- 脆弱性対策情報
- Weekly Report
- インターネット 定点観測

インシデントの報告

各種登録

制御システムセキュリティ

STR31-C. 文字データと null 終端文字を格納するために十分な領域を確保する

データをバッファにコピーする際、データを保持する十分な大きさがバッファにないと、バッファオーバーフローが発生する。バッファオーバーフローは、null 終端バイト文字列(NTBS)の入力時ばかりでなく、NTBS データを加工する際に生じることもある。このようなエラーを防ぐには、コピーする文字列を切り捨てるか、あるいは可能であればコピーする文字データと null 終端文字を保持できる十分なサイズをコピー先に確保する(「STR03-C. null 終端バイト文字列を不注意に切り捨てない」)。

違反コード (オフバイワンエラー)

以下のコード例は、一般にオフバイワンエラーと呼ばれる例を示している[Dowd 06]。ループ

違反コード例(Non-Compliant Code Example)

STR31-C(ncce:getenv())

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

int main() {
    /* ... */
    char buff[256];
    char *editor = getenv("EDITOR");
    if (editor == NULL){
        return 10; /* 環境変数EDITORが設定されていない場合エラー */
    } else {
        strcpy(buff, editor);
    }
    return 0;
}
```

適合コード例 (Compliant Code Example)

STR31-C(`cce: getenv()`)

```
#include <string.h>
#include <stdlib.h>
int main() {
    /* ... */
    char *buff;
    char *editor = getenv("EDITOR");
    if (editor == NULL){
        return 10; /* 環境変数EDITORが設定されていない場合エラー */
    } else {
        size_t len = strlen(editor)+1;
        buff = (char*)malloc(len);
        if (buff==NULL){ return 20; /* malloc() エラー */ }
        strcpy(buff, editor);
    }
    return 0;
}
```

チェッカーはどのような処理を行うべきか？

```
#include <string.h>
#include <stdlib.h>
int main() {
    /* ... */
    char buff[256];
    char *editor = getenv("EDITOR");
    if (editor == NULL){
        return 10; /* 環境変数EDITORが設定されていない場合エラー */
    } else {
        strcpy(buff, editor);
    }
    return 0;
}
```

環境変数 EDITOR に256文字を超える長さの文字列を設定することにより、攻撃できる。

`getenv()` がどのくらいの長さの文字列を返すかは分からない

`strcpy()` の第1引数はローカル変数 (`char[]`)

`strcpy()` の第2引数は (`char*`)

`strcpy()` 呼び出しについて、第一引数が固定長配列のローカル変数、第二引数がポインタになっているものを違反コードとして検知する

他のテストケース

STR31-C では違反コード例と適合コード例が複数挙げられている。
チェッカーの動作確認ではそれら複数のコード例でチェックするとよい。

- 全てのコード例について正しく動作するか?
- `strcpy()`呼び出しの引数を判別するというやり方で正しくコードを検査できるか?
- もし正しく動作しないなら, どのような処理にするべきか?

違反コード例: (off-by-one)

```
char dest[ARRAY_SIZE];
char src[ARRAY_SIZE];
size_t i;
/* ... */
for(i=0; src[i] && (i<sizeof(dest)); i++){
    dest[i] = src[i];
}
dest[i] = '¥0';
/* ... */
```

違反コード例: (argv, strcpy)

```
int main(int argc, char *argv[]) {  
    /* ... */  
    char prog_name[128];  
    strcpy(prog_name, argv[0]);  
    /* ... */  
}
```


適合コード例: (argv, strcpy_s)

```
int main(int argc, char *argv[]) {
    /* Be prepared for argv[0] to be null */
    const char *const name = argv[0] ? argv[0] : "";
    char * prog_name;
    size_t prog_size;
    prog_size = strlen(name)+1;
    prog_name = (char *)malloc(prog_size);
    if (prog_name != NULL) {
        if (strcpy_s(prog_name, prog_size, name)) {
            /* strcpy_s() エラーの処理 */
        }
    } else {
        /* メモリ不足時のエラー処理 */
    }
}
```

以上のコード例による動作チェックから分かること

(off-by-1)コード例を違反コードと判別できない

⇒ `strcpy()`呼び出ししか見てないから

(argc, strcpy)コード例は違反コードとして検知する

⇒ 想定どおり!

`strcpy_s()` のコード例では何も検査していない

⇒ `strcpy()`呼び出ししか見てないから

`strcpy_s()`の場合は `strcpy()`と同様な
処理を追加することにより対応できる。

STR31-Cチェッカーのデザイン

1. ASTのノードを順に辿りながら
 2. `strcpy()`呼び出しだったら
 1. `strcpy()` の二つの引数について以下を調べる
 2. 第一引数は変数
 3. 第一引数の型は固定長配列
 4. 第二引数の型は **ポインター型である**
- 以上の条件が全て満たされたらSTR31-C違反とする!**

STR31-Cチェッカーの限界

- `strcpy(char[], char*)` という形のコードは全て(コーディングルールに違反してなくても)違反コードとして検知してしまう。
- 上記以外の `strcpy()` 呼び出しは(コーディングルールに違反していても)違反コードとして検知しない。
- `strcpy()` 以外の文字列コピー処理を行う関数の違反コードは検知できない。例えば `strncpy()`, `strcpy_s()`, `memcpy()` など。
- `off-by-1`コード例のように、文字列を直接コピーする違反コードは検知できない。

チェッカー実装にあたって心がけるべきこと

- false positive と false negative に注意
- チェッカーは100%完璧でなくても十分役に立つ
- ひとつのコーディングルールに対してチェッカーは複数あってよい。
- 「病的」なコード例は気にしなくてよい。まずは現実が発生しやすいタイプのコード例に注力しよう

チェッカーを実装してみることで、セキュアコーディングルールの内容を別の視点で見ることができ、理解が深まる

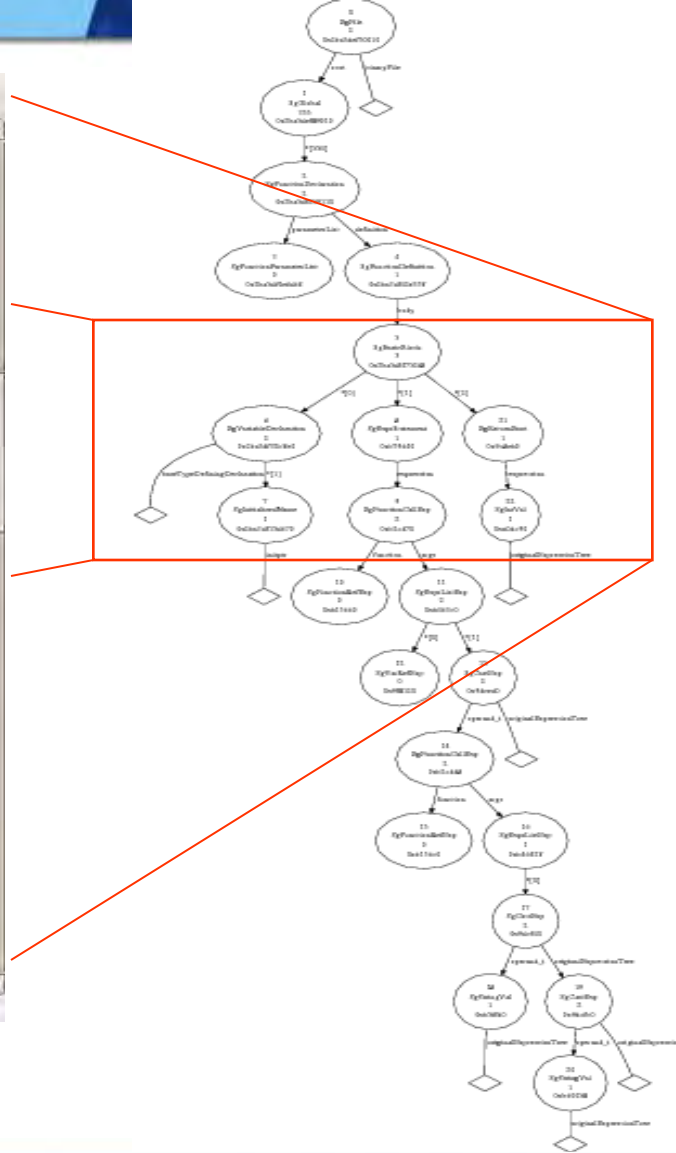
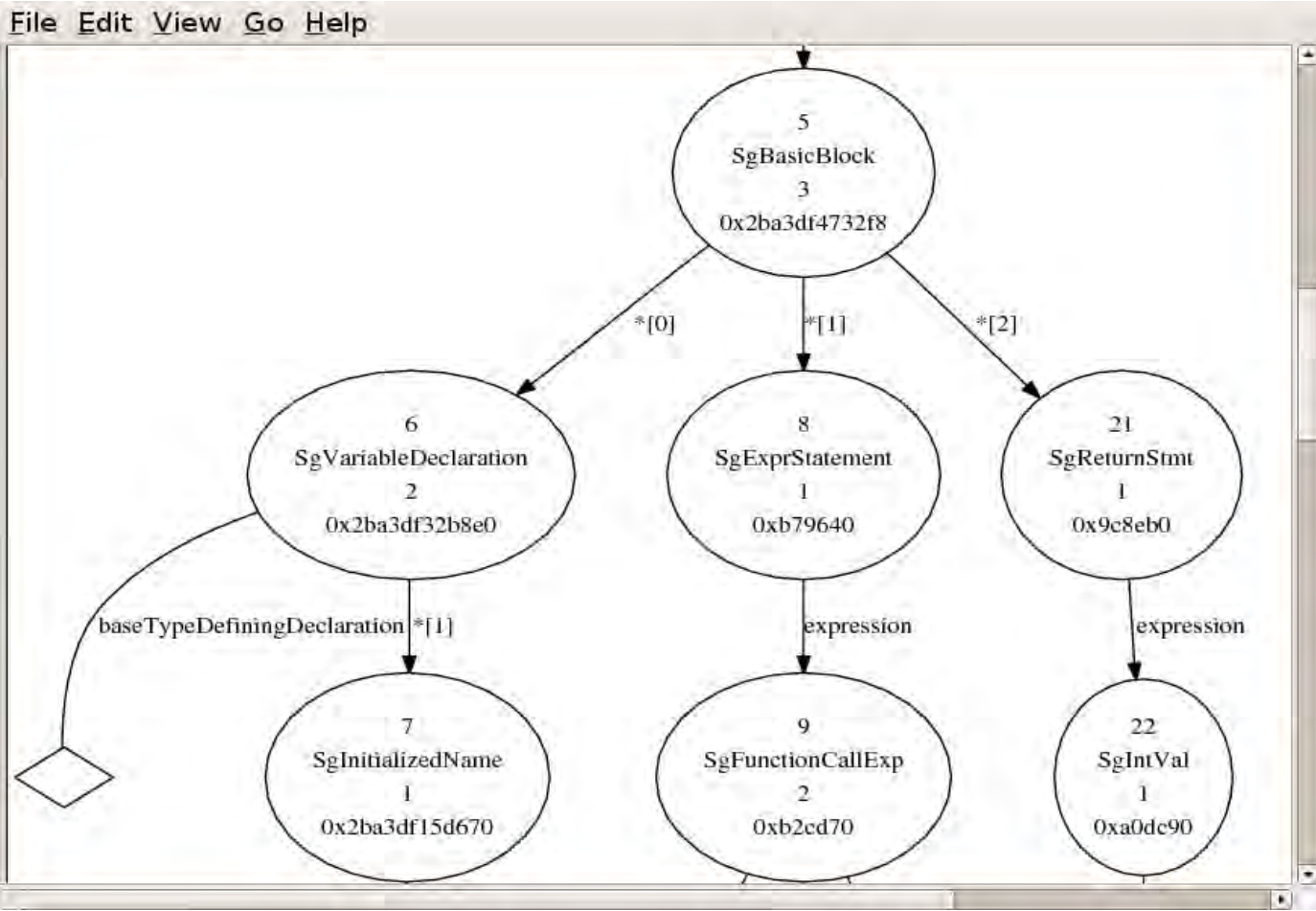
Abstract Syntax Tree 1

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

int main() {
    char buff[256];
    strcpy(buff,
getenv("EDITOR"));
    return 0;
}
```



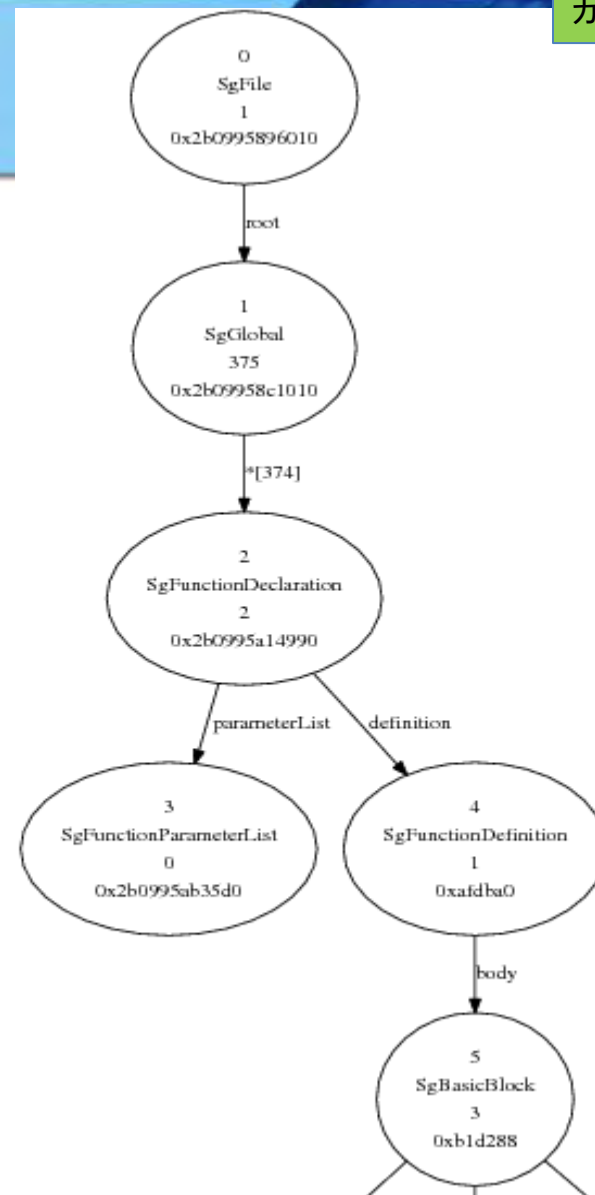
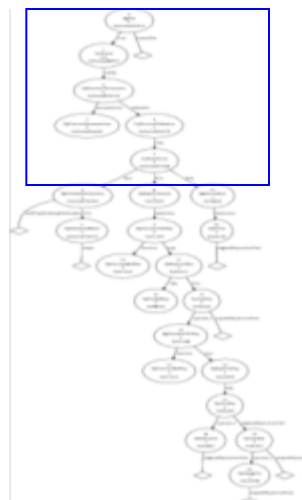
Abstract Syntax Tree 2



Abstract Syntax Tree 3

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

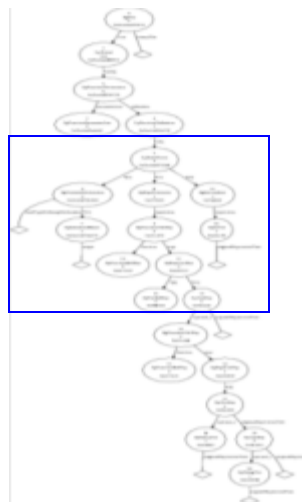
```
int main() {
    char buff[256];
    strcpy(buff, getenv("EDITOR"));
    return 0;
}
```



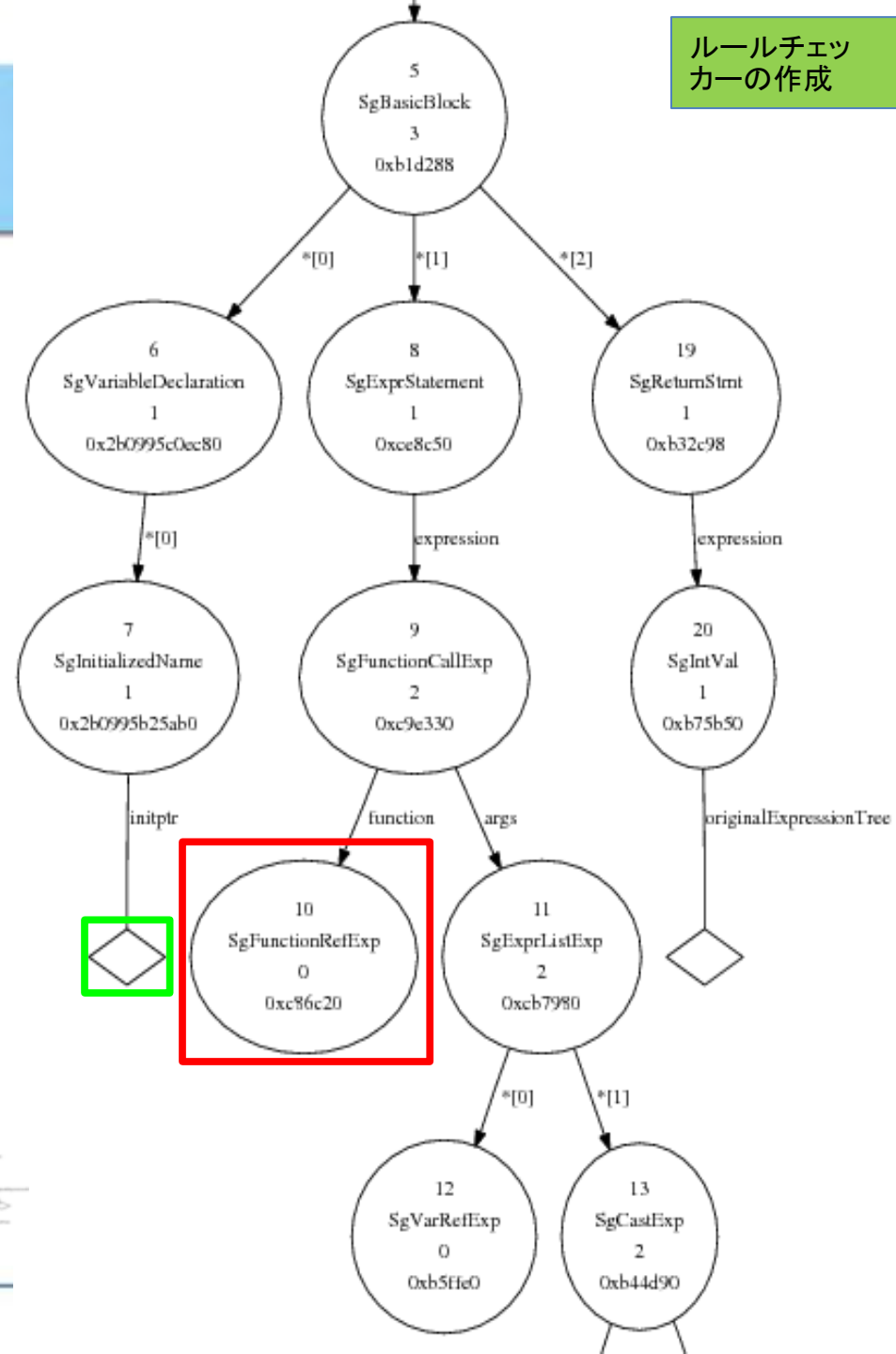
Abstract Syntax Tree 4

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

int main() {
    char buff[256];
    strcpy(buff, getenv("EDITOR"));
    return 0;
}
```



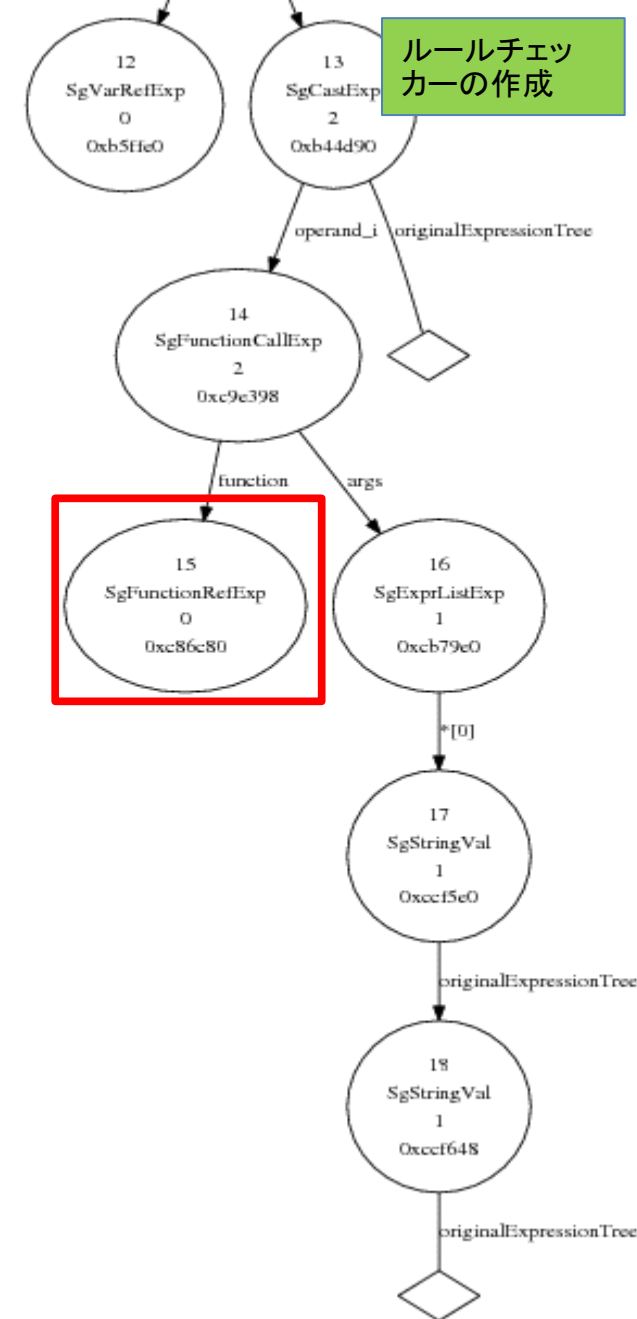
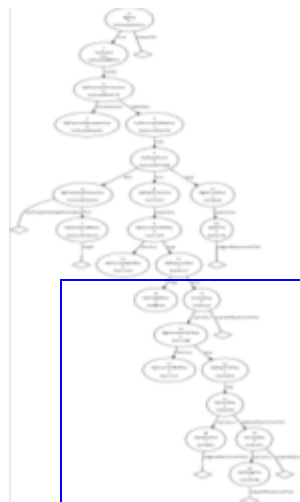
ルールチェッカーの作成



Abstract Syntax Tree 5

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

```
int main() {
    /* ... */
    char buff[256];
    strcpy(buff, getenv("EDITOR"));
    /* ... */
    return 0;
}
```



ルールチェッカーの作成

ASTノードの属性値

File Edit View Go Help

Index

- ▽ SgFile 1
- ▽ SgGlobal (compilerGenerated:...) 2
- ▽ SgFunctionDeclaration (com... 3
 - SgFunctionParameterList (... 4
- ▽ SgFunctionDefinition (com... 5
- ▽ SgBasicBlock (compilerG... 6
 - ▽ SgVariableDeclaration ... 7
 - SgInitializedName 8
 - ▽ SgExprStatement (co... 9
 - ▽ SgFunctionCallExp (c... 10
 - SgFunctionRefExp ... 11
 - ▽ SgExprListExp (co... 12
 - SgVarRefExp (co... 13
 - ▽ SgCastExp (com... 14

pointer:0xb35360

[Click here to go to the parent node](#)

```

SgNode* p_parent : 0xb4ca70
bool p_isModified : 0
$CLASSNAME* p_freepointer : 0xffffffffffffff
static SgFunctionTypeTable* p_globalFunctionTypeTable : 0xbaf0
static std::map<SgNode*,std::string> p_globalMangledNameMap :
static std::map<std::string, int> p_shortMangledNameCache : __cc
Sg_File_Info* p_startOfConstruct : 0xb1e7a0
Sg_File_Info* p_endOfConstruct : 0xb1e800
AttachedPreprocessingInfoType* p_attachedPreprocessingInfoPtr
AstAttributeMechanism* p_attributeMechanism : 0
bool p_need_paren : 0
bool p_lvalue : 0
bool p_global_qualified_name : 0
Sg_File_Info* p_operatorPosition : 0
SgFunctionSymbol* p_symbol_i : 0x7d6b80: varsym strcpy declar
SgFunctionType* p_function_type : 0
  
```

rosecheckers の中身(0): AST traversal

Rosecheckersは, ASTのノードをトップから順に辿りつつ, ルールチェッカーを適用する.

ROSEの Simple Traversal 機能を使っている

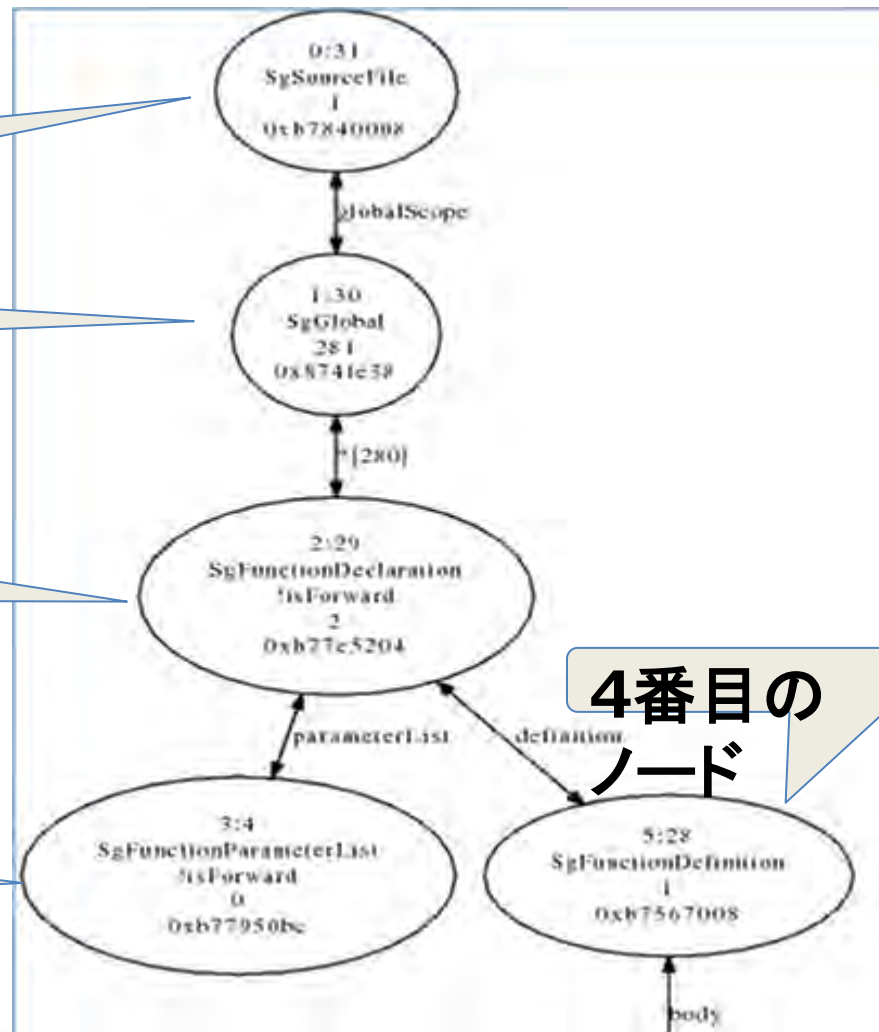
0番目のノード

1番目のノード

2番目のノード

3番目のノード

4番目のノード



rosecheckers の中身(1): main関数

```
#include "rose.h"
```

rosecheckers.C の main()関数

```
int main(int argc, char* argv[]) {
```

```
    SgProject* project = frontend(argc, argv);
```

```
    ROSE_ASSERT( project );
```

ROSE がソースコードを解析して
ASTを生成

```
    AstSimpleProcessing* visitorTraversal;
```

```
    .....
```

ASTを辿り, 各ノードを調べる

```
    visitorTraversal->traverseInputFiles(  
        project, preorder);
```

```
    delete visitorTraversal;
```

```
    return 0;
```

```
}
```

rosecheckers の中身(2): 各ルールのチェック

ASTの各ノードに対して以下の関数が呼び出される。

```
bool EXP(const SgNode *node) {  
    bool violation = false;  
    violation |= EXP01_C(node);  
    violation |= EXP04_C(node);  
    .....  
    violation |= EXP36_C(node);  
    violation |= EXP37_C(node);  
    return violation;  
}
```

EXP.Cの末尾あたり

各関数は個々のコーディングルールの検査に対応し、ルール違反を検出すると true を返す。

同様の関数が STR, MEM などについても用意されている。

チェッカーの作成(1)

```
#include "rose.h"
#include "utilities.h"

// ensure sufficient storage for strings
bool STR31_C(const SgNode *node) {
    /* ??? */
}

bool STR(const SgNode *node) {
    bool violation = false;
    /* ... */
    violation |= STR31_C(node);
    return violation;
}
```

ASTの各ノードに対してこの関数が呼び出される。違反コード例に対してエラーメッセージを出力し、true を返すようにする。

チェッカーの作成(2)

DONE

この部分をどう作るか?

```
bool STR31_C(const SgNode *node){  
    /* ??? */  
}
```

```
bool STR(const SgNode *node){  
    bool violation = false;  
    /* ... */  
    violation |= STR31_C(node);  
    return violation;  
}
```

ASTのノードを順に辿りながら
strcpy() 呼び出しだったら

1. **strcpy()** の二つの引数について
2. 第一引数は変数
3. 第一引数は固定長配列
4. 第二引数はポインタ型である

- 以上の条件が満たされればSTR31-C違反!

チェッカー開発で使える便利な関数

utilities.C, utilities_cpp.C, utilities.h, utilities_cpp.h, utilities-inline.h などに含まれている。

```
// 与えられた名前の関数呼び出しに対応するnodeであればNULL以外を返す
const SgFunctionSymbol *isCallOfFunctionNamed(
    const SgNode *node, const std::string &name);

// 関数呼び出しのi番目の引数を返す(0始まり). cast式の場合その中身を返す.
// 引数が足りない場合はNULLを返す.
const SgExpression* getFnArg(
    const SgFunctionRefExp* node, int i);

void print_error(
    const SgNode* node, const char* rule,
    const char* desc, bool warning = false);
```

チェッカーの作成(3)

```
bool STR31 C(const SgNode *node) {
    if (!isCallOfFunctionNamed(
        node, "strcpy"))
        return false;
    /* ??? */
}
```

DONE

この時点での node は
strcpy() 呼び出しを
指している

ASTのノードを順に辿りながら
strcpy() 呼び出しだったら

1. strcpy() の二つの引数について
2. 第一引数は変数
3. 第一引数は固定長配列
4. 第二引数はポインタ型である

- 以上の条件が満たされれば
STR31-C違反!

isSg 関数群

(SgNode*) 型からの型変換に使える関数群。
ノードが適切な型でない場合には NULL を返す。

```
const SgNode* node;  
const SgFunctionRefExp* sig_fn  
    = isSgFunctionRefExp(node);  
if (sig_fn == NULL) {  
    cerr << "Node is not a "  
        << "SgFunctionRefExp!" << endl;  
}
```

チェッカーの作成(4)

```
bool STR31_C(const SgNode *node) {
    if (!isCallOfFunctionNamed(
        node, "strcpy"))
        return false;
```

```
const SgVarRefExp* ref =
    isSgVarRefExp(
        getFnArg(
            isSgFunctionRefExp(node), 0));
if (ref == NULL) return false;
```

```
/* ??? */
```

```
}
```

この時点で ref は strcpy() の第一引数を指しており、それは変数である。

ASTのノードを順に辿りながら
strcpy() 呼び出しだったら

1. **strcpy()** の二つの引数について
2. 第一引数は変数 DONE
3. 第一引数は固定長配列
4. 第二引数はポインタ型である

以上の条件が満たされればSTR31-C違反!

チェッカーの作成(5)

```
bool STR31_C(const SgNode *node) {
    if (!isCallOfFunctionNamed(
        node, "strcpy"))
        return false;

    const SgVarRefExp* ref =
        isSgVarRefExp(
            getFnArg(
                isSgFunctionRefExp(node), 0));
    if (ref == NULL) return false;
    if (!isSgArrayType(
        getRefDecl(ref)->get_type()))
        return false;
    /* ??? */
}
```

DONE

ASTのノードを順に辿りながら
strcpy() 呼び出しだったら

1. **strcpy()** の二つの引数について

2. 第一引数は変数

3. 第一引数は固定長配列

4. 第二引数はポインタ型である

以上の条件が満たされれば
STR31-C違反!

チェッカーの作成(6)

```

bool STR31_C(const SgNode *node) {
    if (!isCallOfFunctionNamed(
        node, "strcpy"))
        return false;

    const SgVarRefExp* ref =
        isSgVarRefExp(
            getFnArg(
                isSgFunctionRefExp(node), 0));
    if (ref == NULL) return false;
    if (!isSgArrayType(
        getRefDecl(ref)->get_type()))
        return false;
    if (!isSgPointerType(
        getFnArg(isSgFunctionRefExp(node), 1)->get_type()))
        return false;
}

```

ASTのノードを順に辿りながら
strcpy() 呼び出しだったら

1. **strcpy()** の二つの引数について
2. 第一引数は変数
3. 第一引数は固定長配列

DONE

4.

第二引数はポインタ型である

以上の条件が満たされれば
STR31-C違反!

STR31-Cのチェッカー

```

#include "rose.h"
#include "utilities.h"
bool STR31_C(const SgNode *node) { // ensure sufficient storage for
strings
    if (!isCallOfFunctionNamed( node, "strcpy")) return false;

    const SgVarRefExp* ref =
        isSgVarRefExp( getFnArg( isSgFunctionRefExp(node), 0));
    if (ref == NULL) return false; // strcpy() のコピー先は変数でない
    if (!isSgArrayType( getRefDecl( ref)->get_type()))
        return false;
    if (!isSgPointerType( getFnArg( isSgFunctionRefExp(node), 1)->get_type()))
        return false;

    print_error( node, "STR31-C",
        "String copy destination must contain sufficient storage");
    return true;
}

```

ASTの各ノードについてSTR31_C()が呼び出される

strcpy()呼び出しが見つかった

第一引数はローカル変数で固定長配

第二引数はポインタ(配列でない)

チェッカーのコンパイルと動作チェック

コードの追加・変更など行って `rosecheckers` をコンパイルし直すには
`pgms` ターゲットを使う:

```
make pgms
```

全てのルールについて `rosecheckers` の動作をテストするには
`tests` ターゲットを使う:

```
make tests
```


チェッカーのコンパイルと動作チェック

違反コードに対してはエラーメッセージを出力し，適合コードに対しては何も出力されないのが正しい動作．

(違反コードの場合)

```
% ./rosecheckers test/c.ncce.wiki.EXP.c
```

```
EXP.c:5: error: EXP09-C: malloc called using  
something other than sizeof()
```

```
%
```

(適合コードの場合)

```
% ./rosecheckers test/c.cce.wiki.EXP.c
```

```
%
```

その他の便利関数

`isSg*** (node)`

`unparseToString()`

`querySubTree (SgNode* node, type)`

`unparseToString()`

`node` に対応するソースコードの文字列表現を返す。デバッグ時に役に立つ。

(使用例)

```
const SgNode* node;  
cout << "Node: "  
      << node->unparseToString()  
      << endl;
```

```
querySubTree(node, type)
```

`node` の下にぶら下がっているサブツリーのなかから適切な型 (`type`) を持つノードをリストにして返す (実際には `std::vector`) .

(使用例)

```
const SgNode* node;
Rose_STL_Container<SgNode *> nodes
    = NodeQuery::querySubTree(
        const_cast<SgNode*>(node), V_SgVarRefExp);
Rose_STL_Container<SgNode*>::iterator i;
for (i = nodes.begin(); i != nodes.end(); ++i) {
    cout << "A SgVarRefExp: "
        << (*i)->unparseToString() << endl;
}
```

(注) `querySubTree` の第一引数は `const` のつかない (`SgNode*`) 型のオブジェクトなので, `const_cast` している.

チャレンジ問題

rosecheckers にまだ実装されていないルールチェッカーを実装せよ。
(例) ARR36-C

ARR36-C 違反コード例

```
int nums[SIZE];
char *strings[SIZE];
int *next_num_ptr = nums;
int free_bytes;
/*
 * 配列にデータを入れるごとに next_num_ptr をインクリメント
 */
free_bytes = strings - (char **)next_num_ptr;
```

異なる配列オブジェクトを指す二つのポインタの差分をとろうとしている。

ARR36-C 適合コード例

```
int nums[SIZE];
char *strings[SIZE];
int *next_num_ptr = nums;
int free_bytes;
/*
 * 配列にデータを入れるごとに next_num_ptr をインクリメント
 */
free_bytes =
    (&(nums[SIZE]) - next_num_ptr) * sizeof(int);
```

同一配列オブジェクトを指す二つのポインタの差分としている。