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

脆弱なコードをめぐる問題

- 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。プログラマがなんとかしないといけない。

安全でない(insecure)コードの定義

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

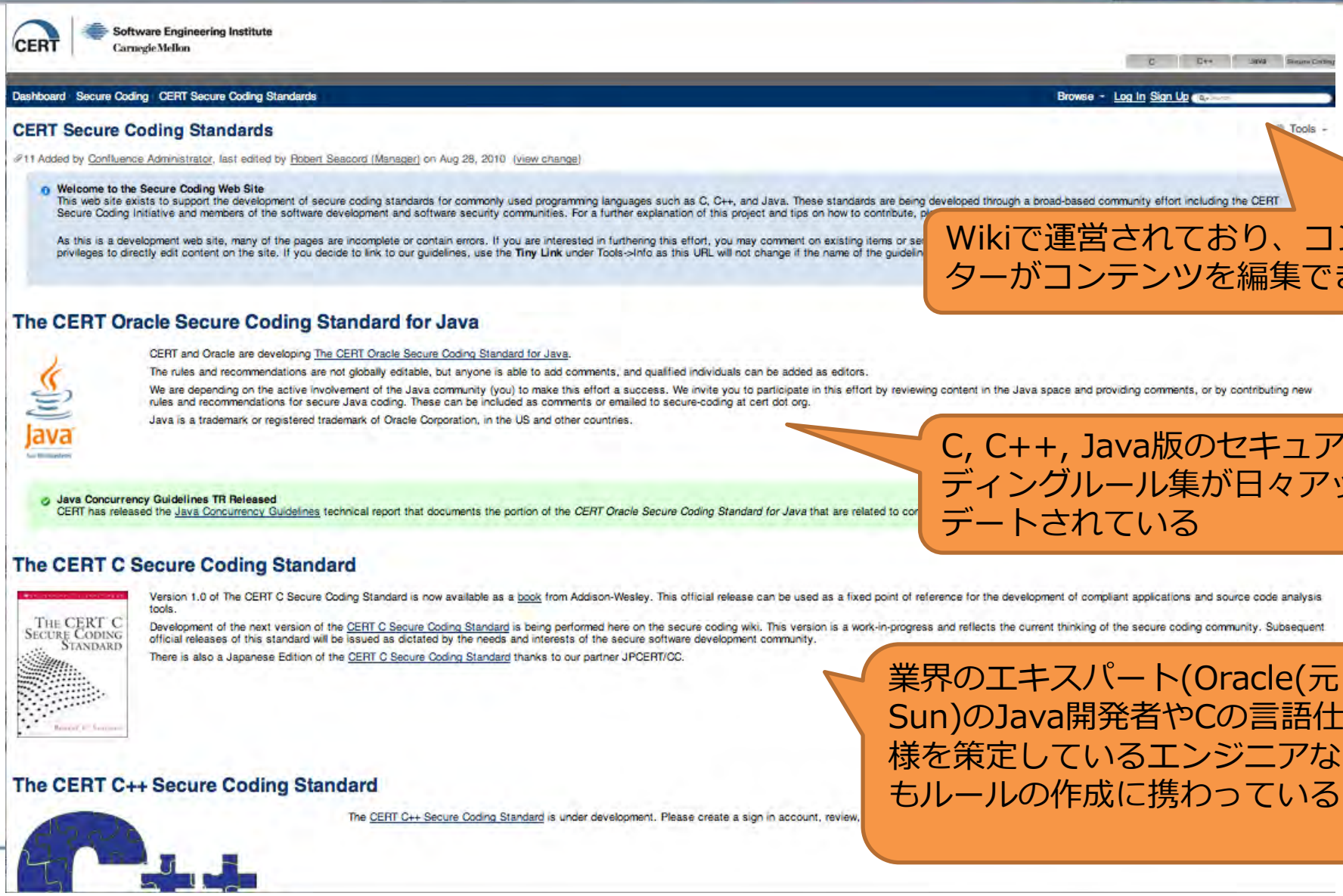
方法

- 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®



CERT | Software Engineering Institute Carnegie Mellon

Dashboard · Secure Coding · CERT Secure Coding Standards

Browse · Log In Sign Up

CERT Secure Coding Standards

Tools

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, pl

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 se 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 guidelin

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 at cert dot org.

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

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 cor

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.

The CERT C++ Secure Coding Standard

The CERT C++ Secure Coding Standard is under development. Please create a sign in account, review,

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

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

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ルールの作成に携わっている

JPCERT/CC セキュアコーディング

JPCERT CC®
Japan Computer Emergency Response Team Coordination Center
JPCERTコーディネーションセンター

安全・安心なIT社会のための、国内・国際連携を支援する

► お問い合わせ ► サイトマップ ► English

Home > ラーニング > セキュアコーディング

最新情報を取得 (RSS | メーリングリスト)
HTTPS

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

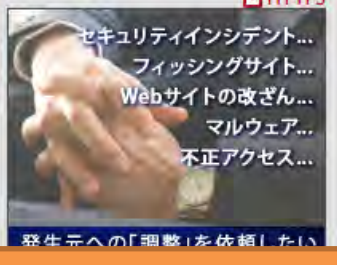
セキュアコーディング

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、安全なソフトウエ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エア開発を支援する資料**
  - OWASP「ソフトウエアセキュリティ保証成熟度モデル」

Topへ



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

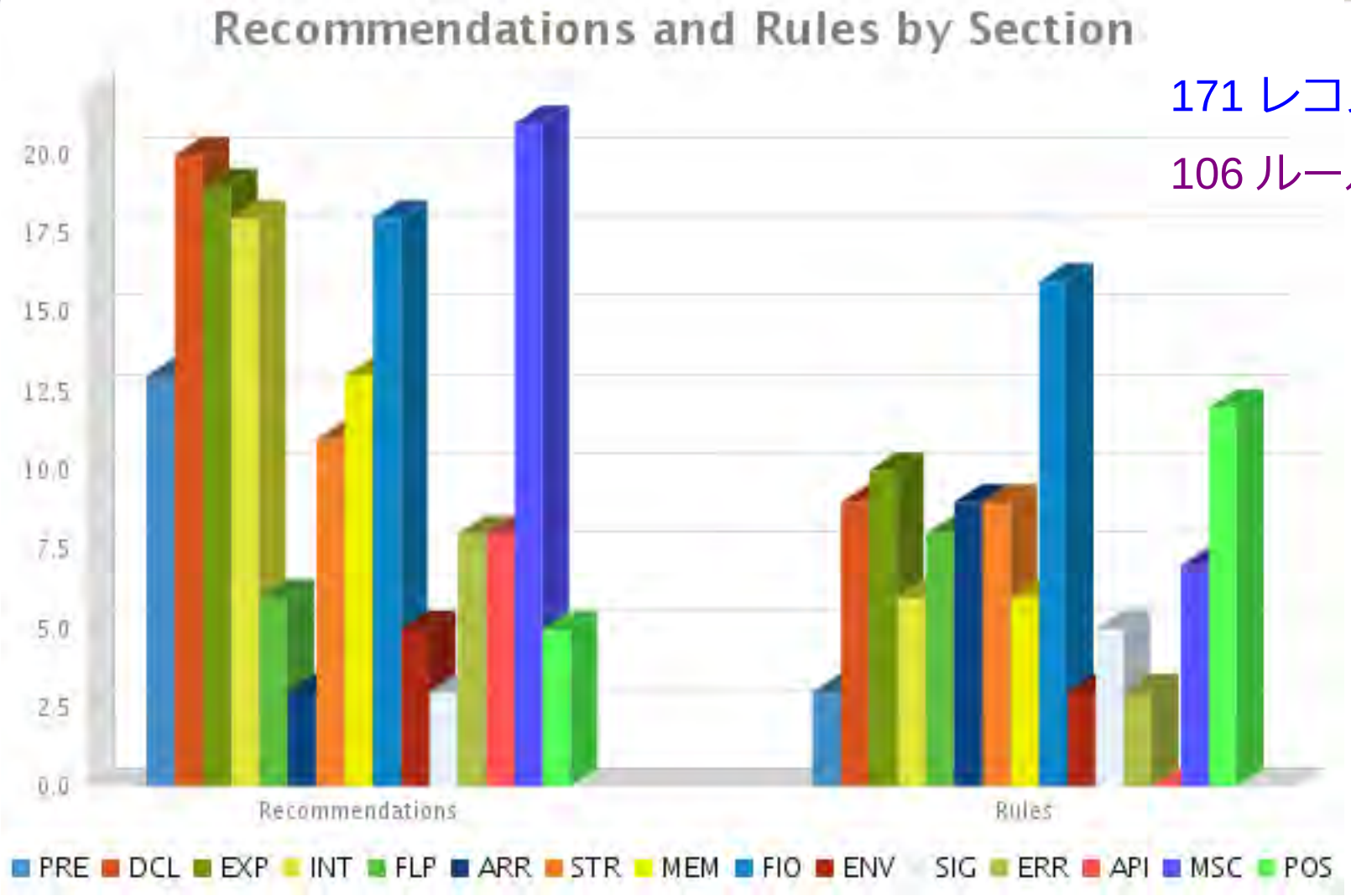
2010-12-16
冬期の長期休暇を控えて
2010/12

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

C ルール/レコメンデーションの分布



171 レコメンデーション

106 ルール

JPCERT CC®

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.

静的解析いろいろ

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)

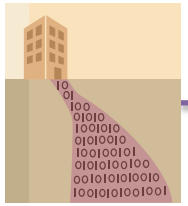
プログラム理解 (Program Understanding)
- IDEに組み込まれている(「このメソッド呼出を全て検索」とか)
- Fujaba (www.uni-paderborn.de/cs/fujaba)

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

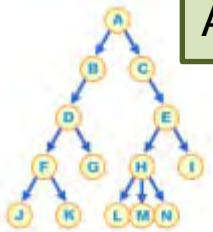
セキュリティチェック
- Codesurfer
- coverity
- fortify
- LDRA
- klocworks
- **Rosecheckers**

静的コード解析の概念図



ソースコード

ROSEの仕事



モデルの作成

AST

rosecheckers

分析

結果のレポート

rosecheckers
の出力

セキュリティの知識

ROSEのAPIを使い
分析エンジンを実装

今日のセミナーはここ！

CERT C/C++ Secure Coding Standards

抽象構文木 (AST)

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

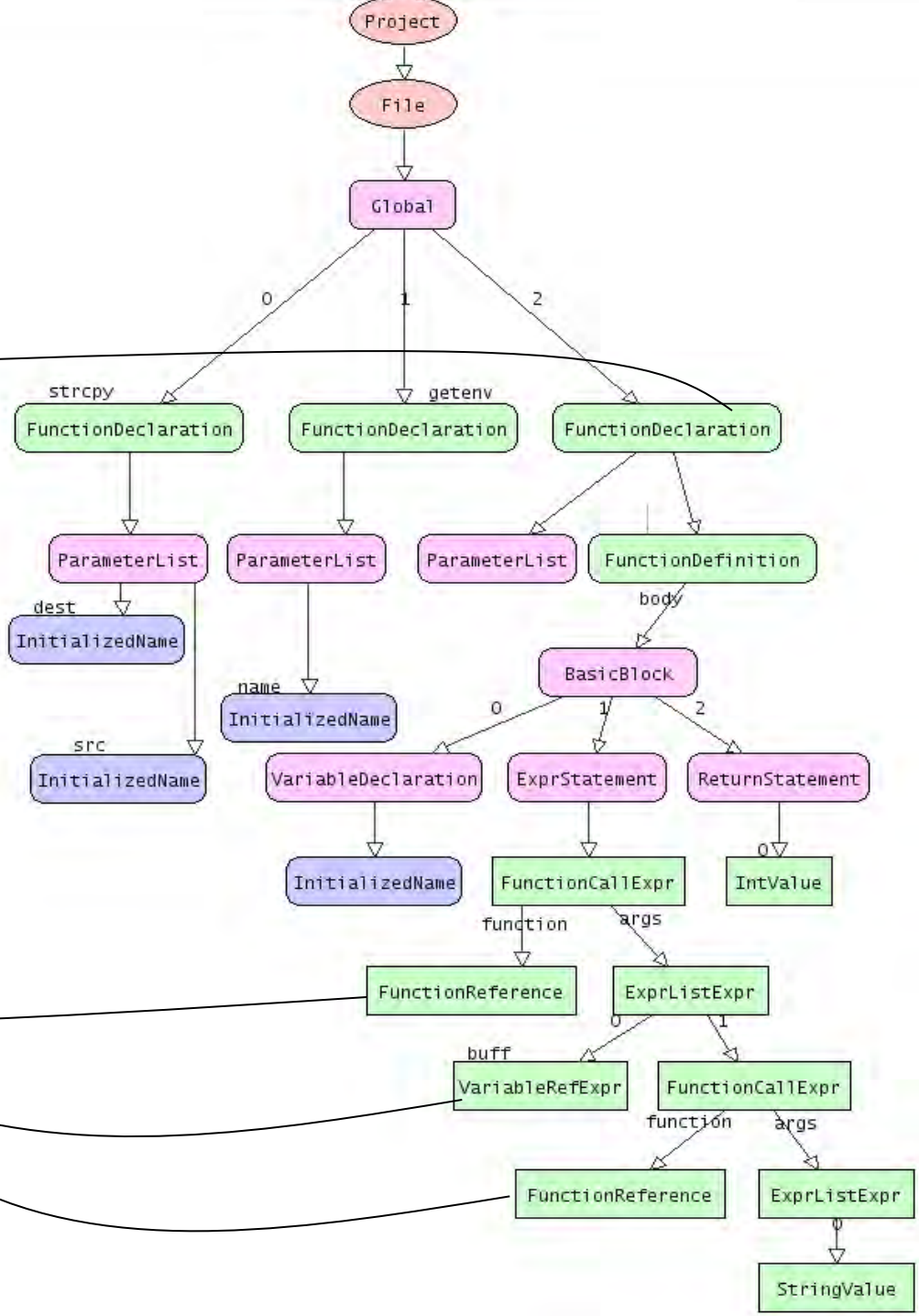
静的コード解析ツールもASTを利用

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;
}
```



ROSEとは?

JPCERT CC®

LAWRENCE LIVERMORE NATIONAL LABORATORY
Science and Technology in the National Interest

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/

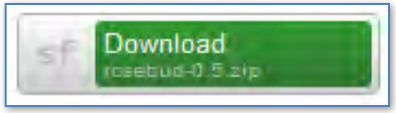
rosecheckers

- 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/

Rosebud

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



JTC1/SC22/WG14 - C

C言語仕様のメジャーアップデート

- 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

NIST SAMATE Project の SATE

JPCERT CC®

Static Analysis Tool Exposition (SATE) 2010

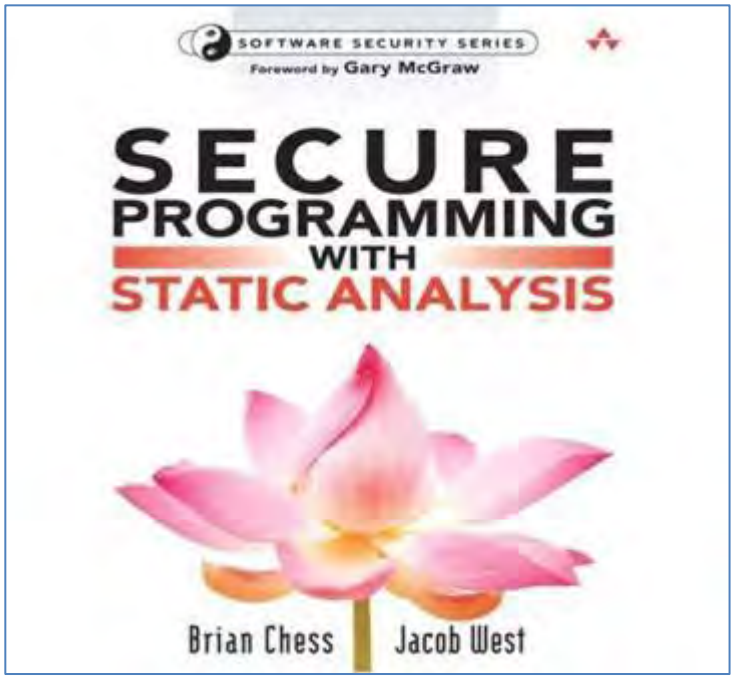
- 静的解析ツールベンダが参加
- 課題のコードを自社ツールで解析し，結果をNISTにレポート．NISTの研究者が結果を分析してレポートにまとめる．
- 2009年のレポート

  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

参考図書

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

アジェンダ

- 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を試してみましょう.

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とおり体験しましょう.

JPCERT CC®

rosebud のセットアップ

自分のPCに rosebud をインストールしよう

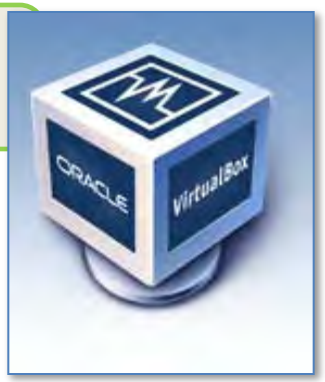
Rosebud-0.5のダウンロードと展開

rosebud を使うためには, VMWareプレーヤーが必要. 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

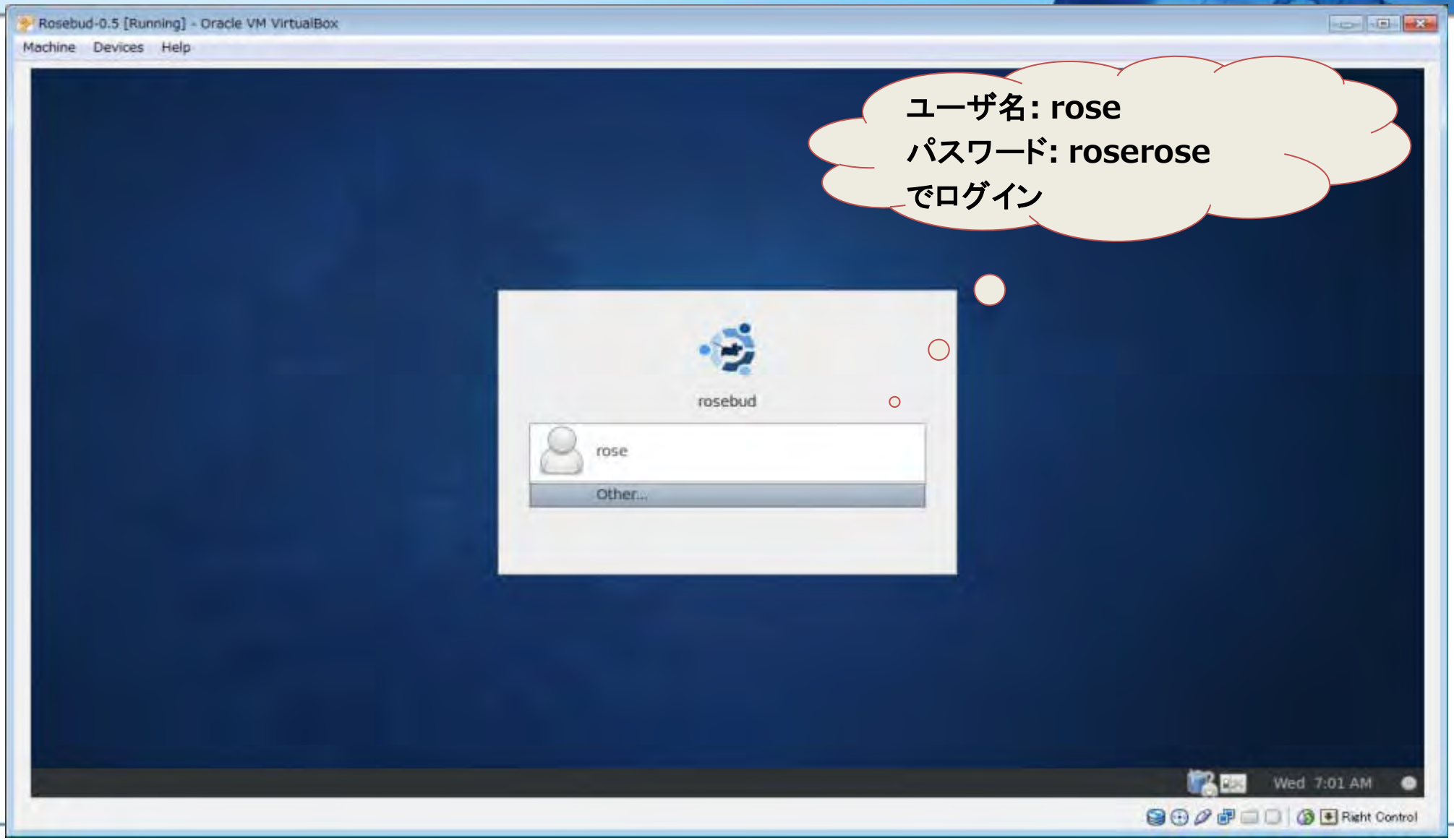
- Rosebudをダウンロードしたら7zipで解凍
- VirtualBoxで新たに仮想マシンを作成
- ハードディスクの設定で **rosebud.vmdk** を選択.
- Rosebud 仮想イメージが起動して数秒するとログインプロンプトが現れる.
- ユーザ: **rose**　パスワード: **roserose**　でログイン.
- Xfceのデスクトップ環境が立ち上がる.
- Guest Additions をインストールしておくと, X window system がホストOS上のGUI窓のサイズ変更に追従してくれて便利

Rosebud-0.5: ディレクトリ構成

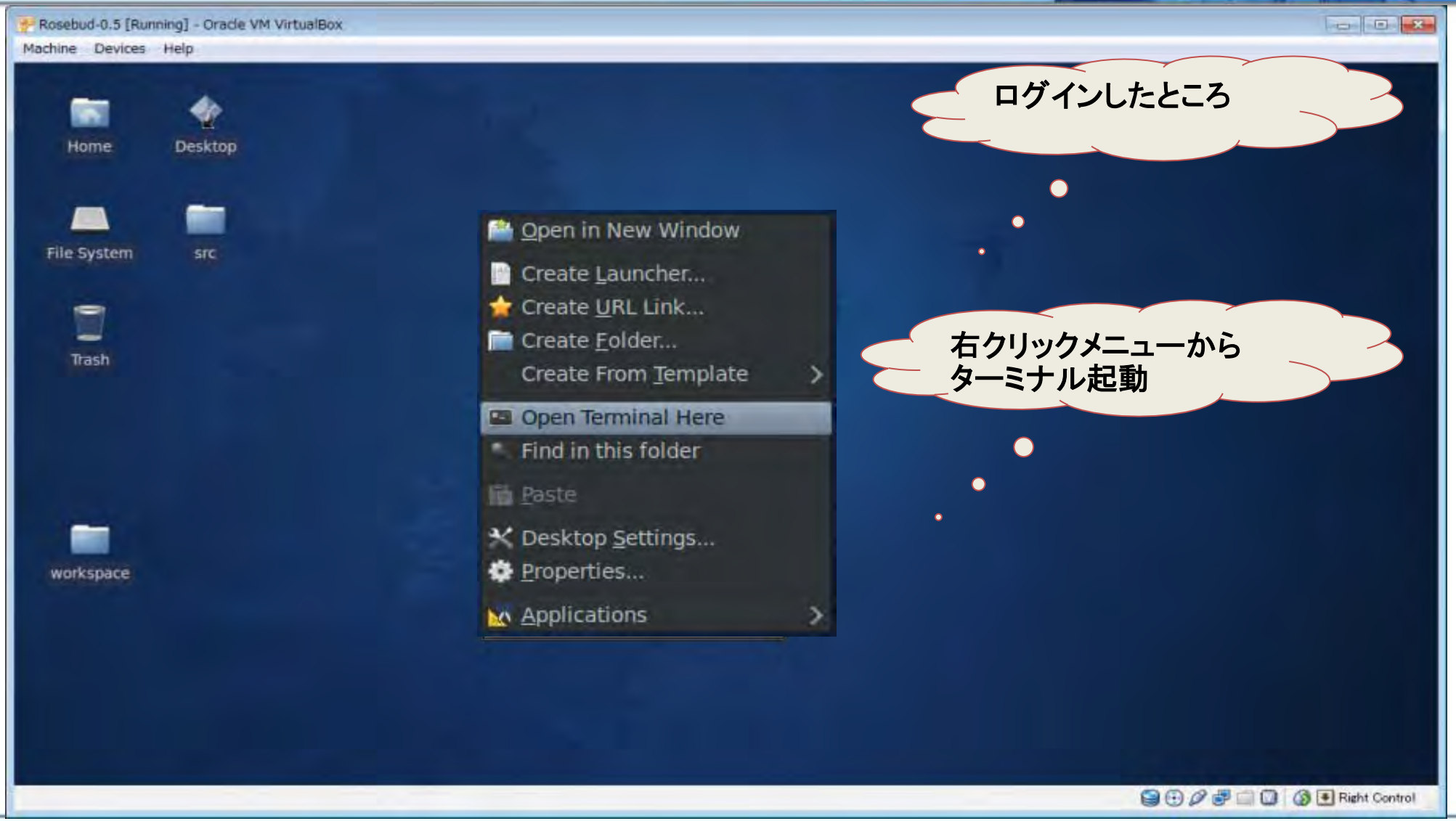
右クリックでプログラムメニューを出し, ターミナルを起動.
以降の作業は ~/workspace/ の下で行う.

- ~/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

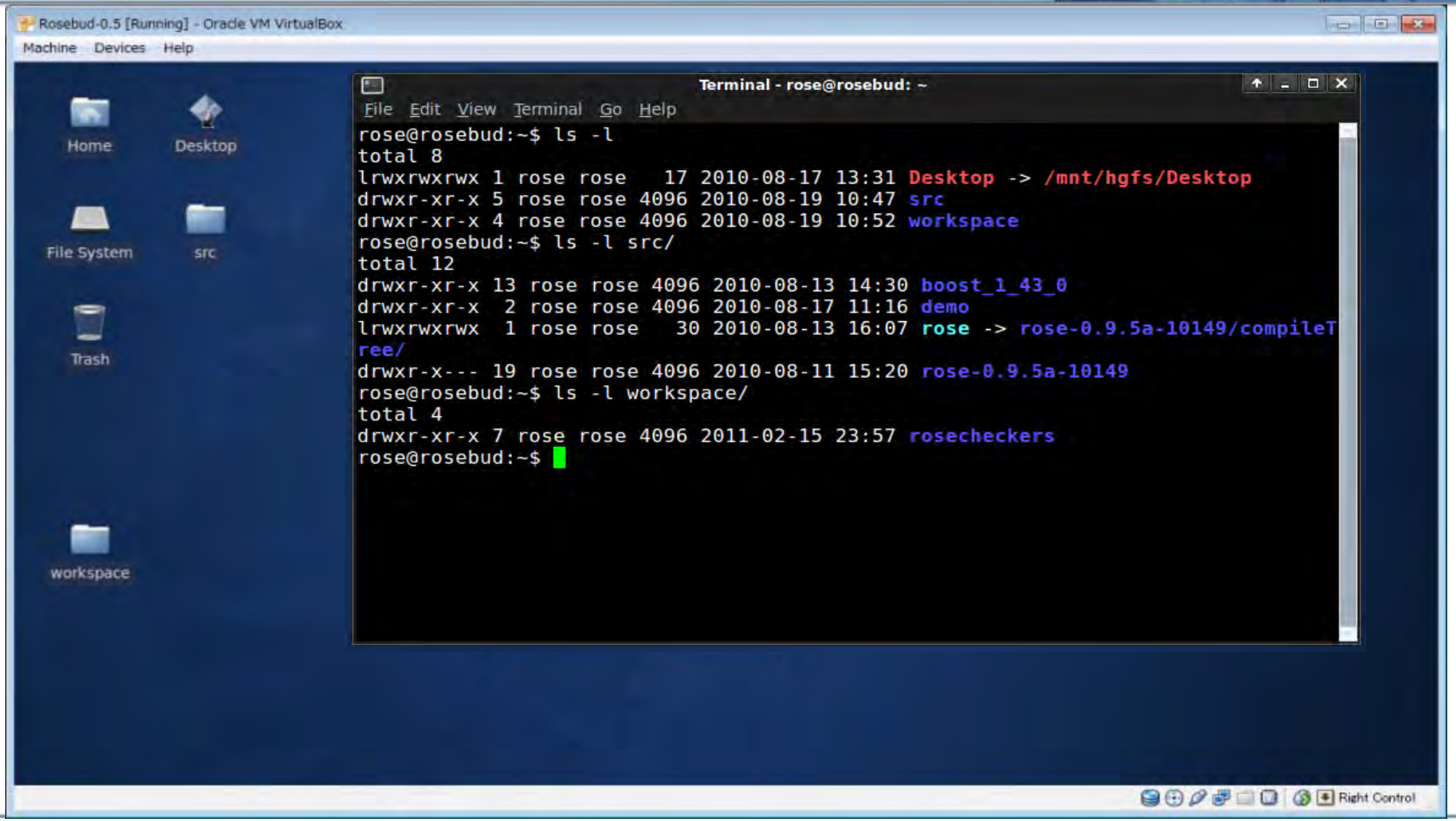
Rosebud-0.5 スクリーンショット: ログイン画面



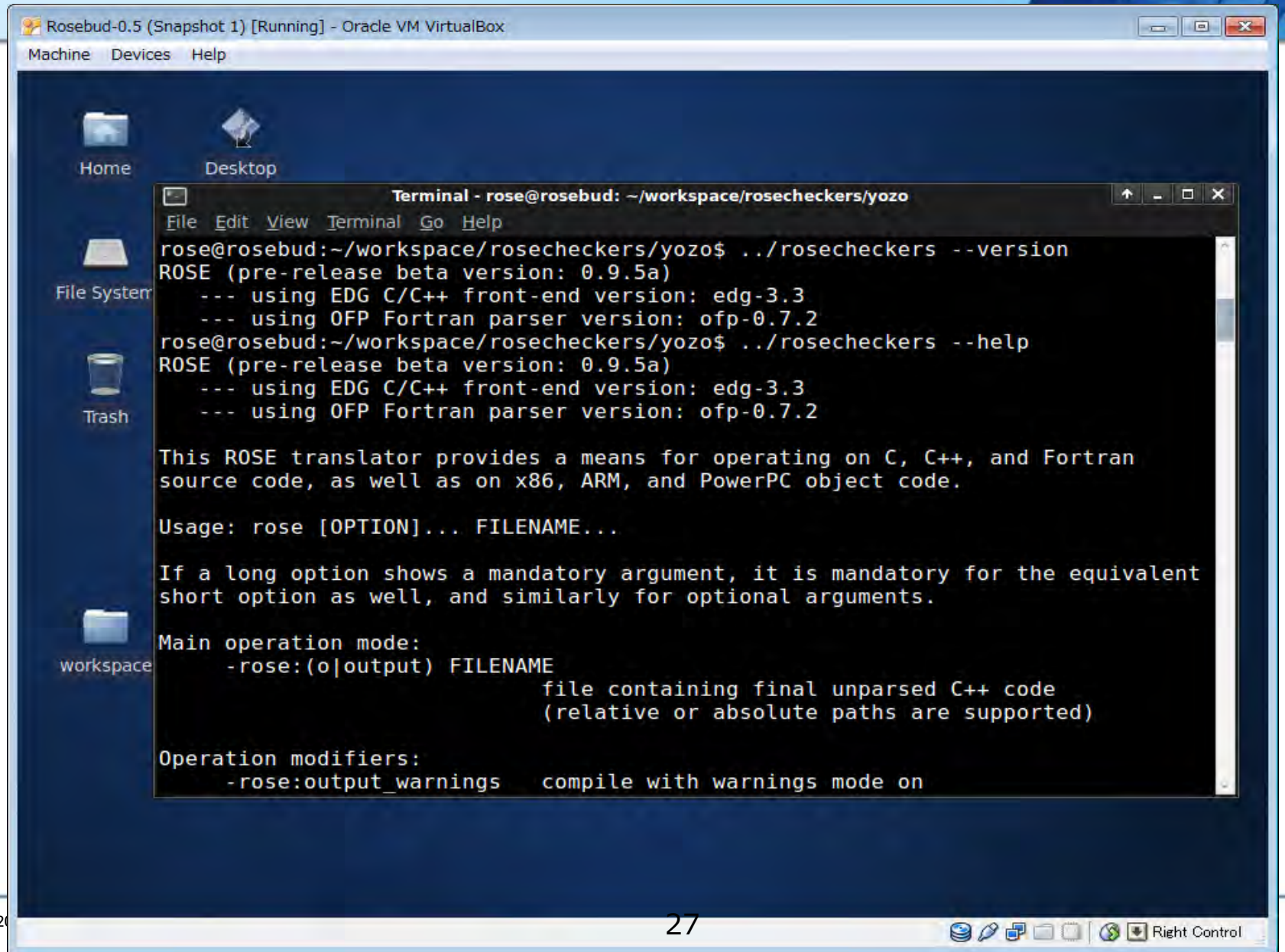
Rosebud-0.5 スクリーンショット: 右クリックのメニュー



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



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

JPCERT CC®

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

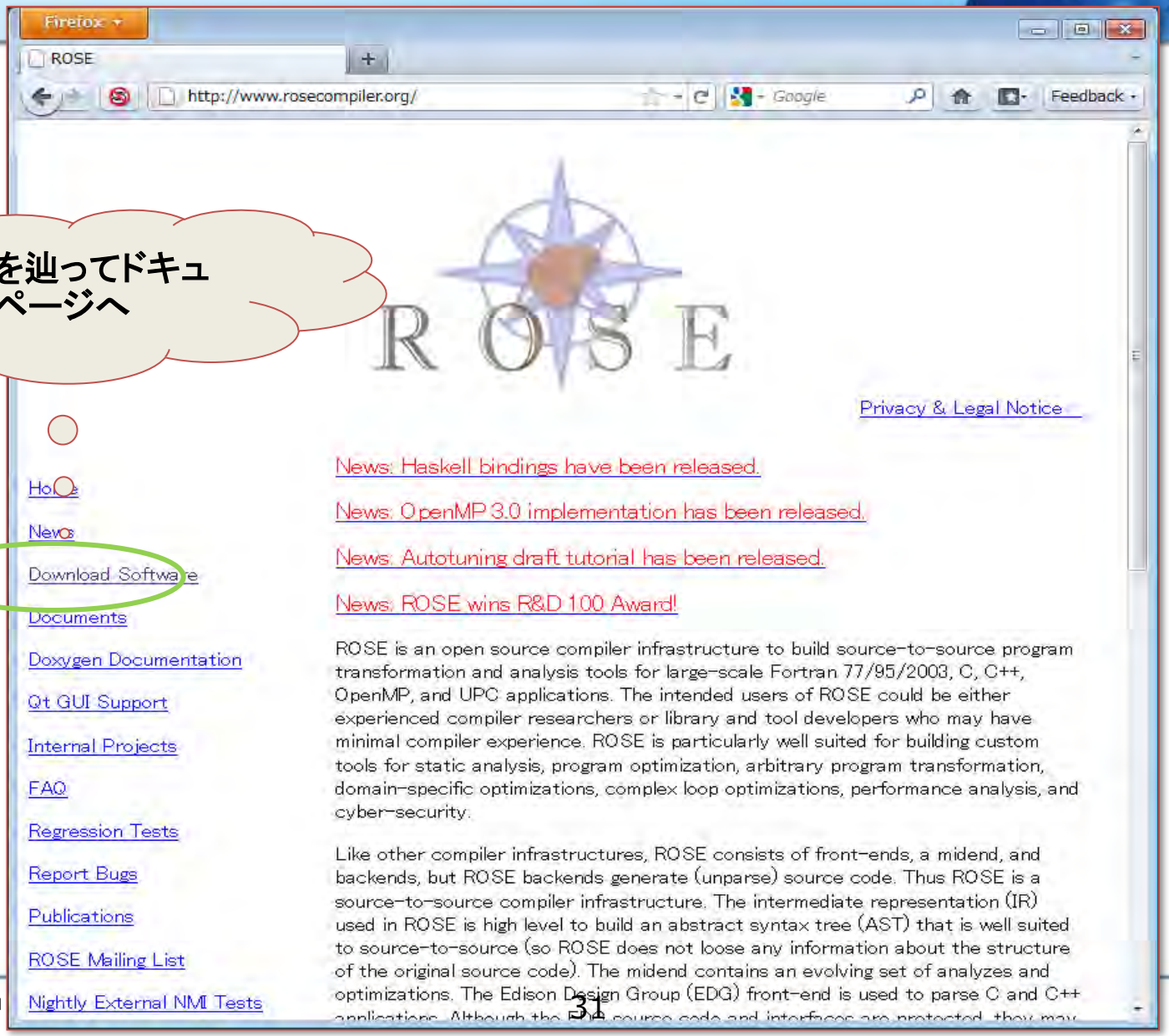
ROSEの各クラスとメソッドのHTMLドキュメント

(Doxygenで生成)

ROSEの
ドキュメント

ROSE Homepage

このリンクを辿ってドキュメント一覧ページへ



ROSE Documents

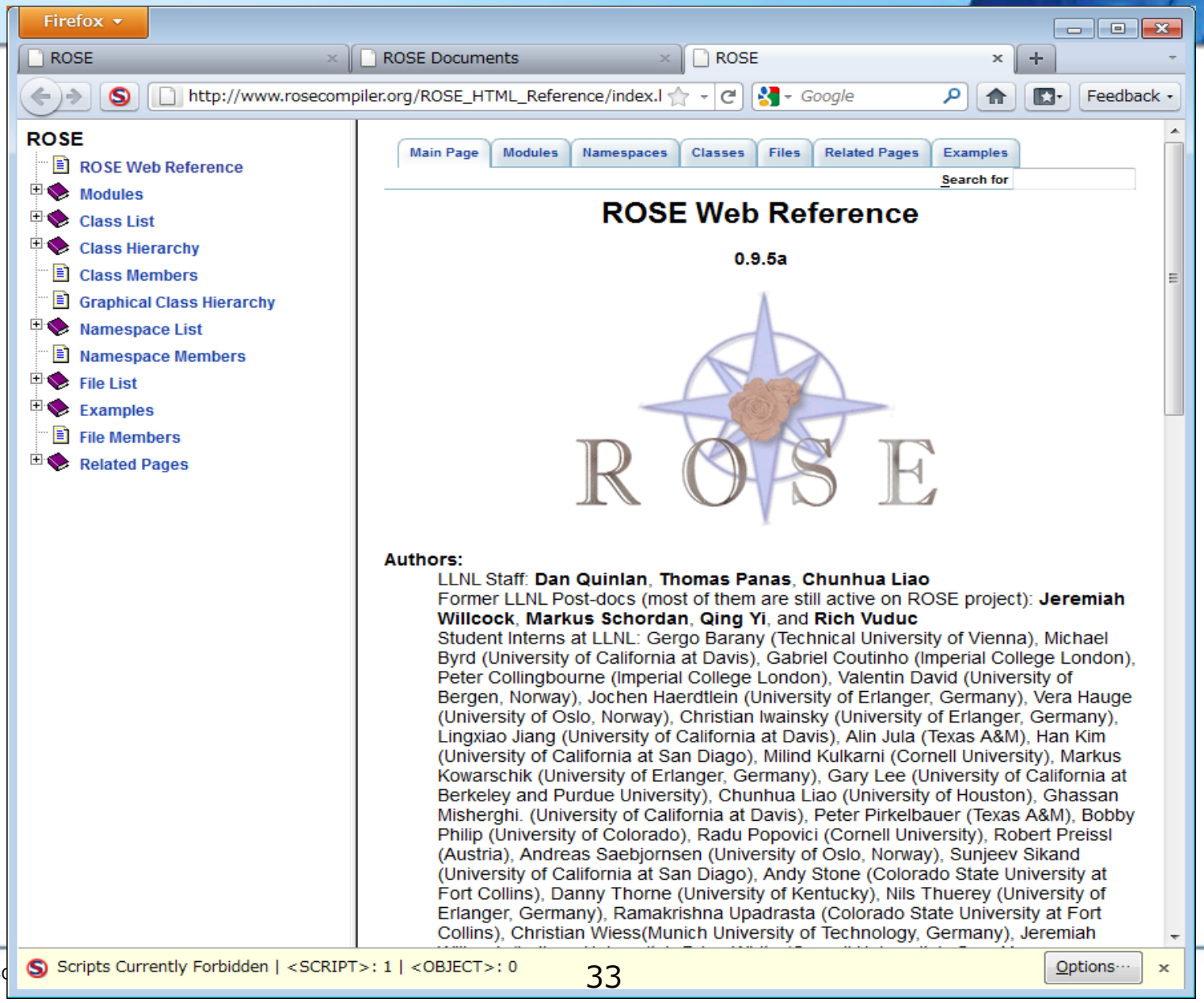
ROSEの
ドキュメント

JPCERT CC®

クラスやメソッドのドキュメントはこのリンクを辿る

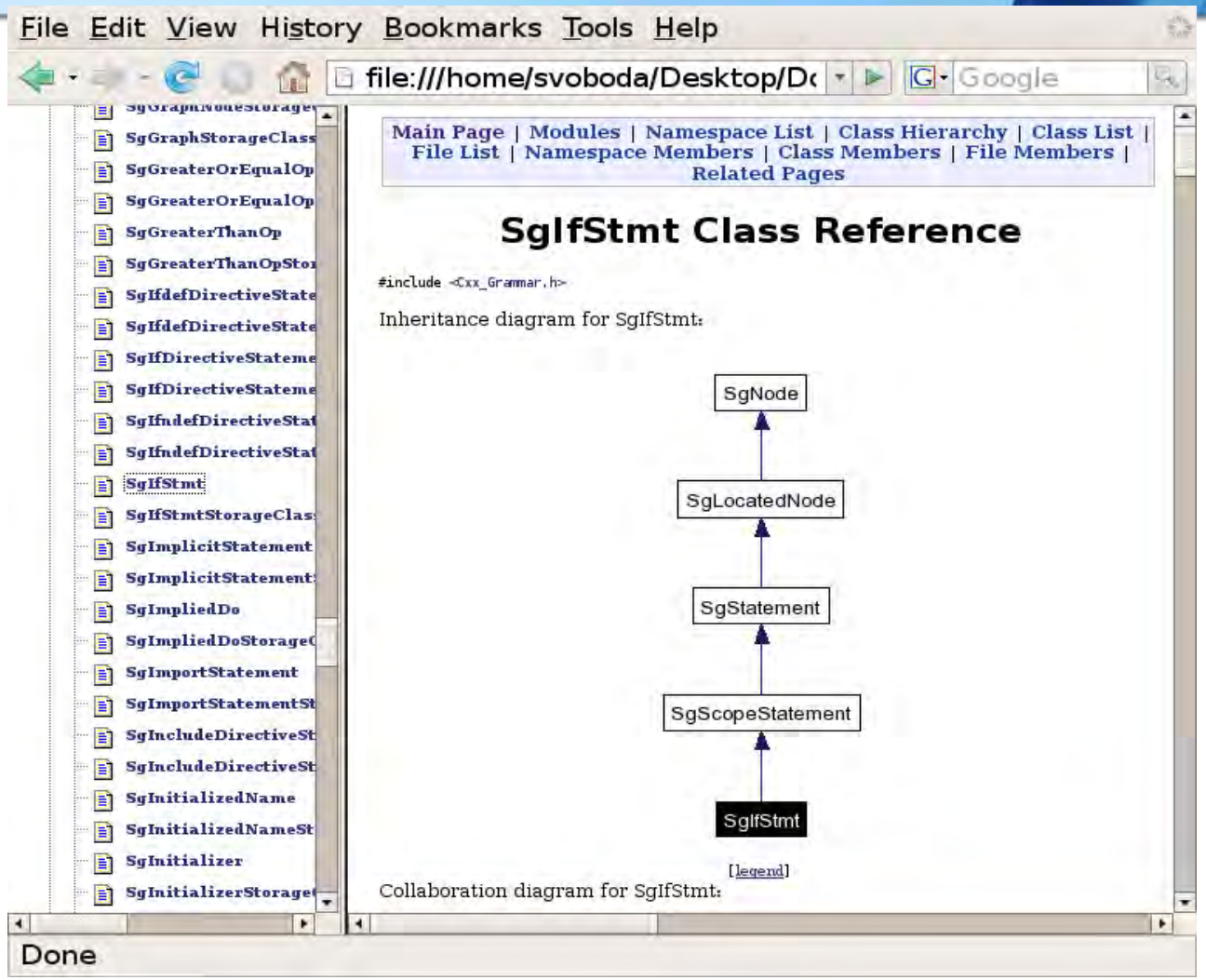
ROSE Web Reference(1)

ROSEの
ドキュメント



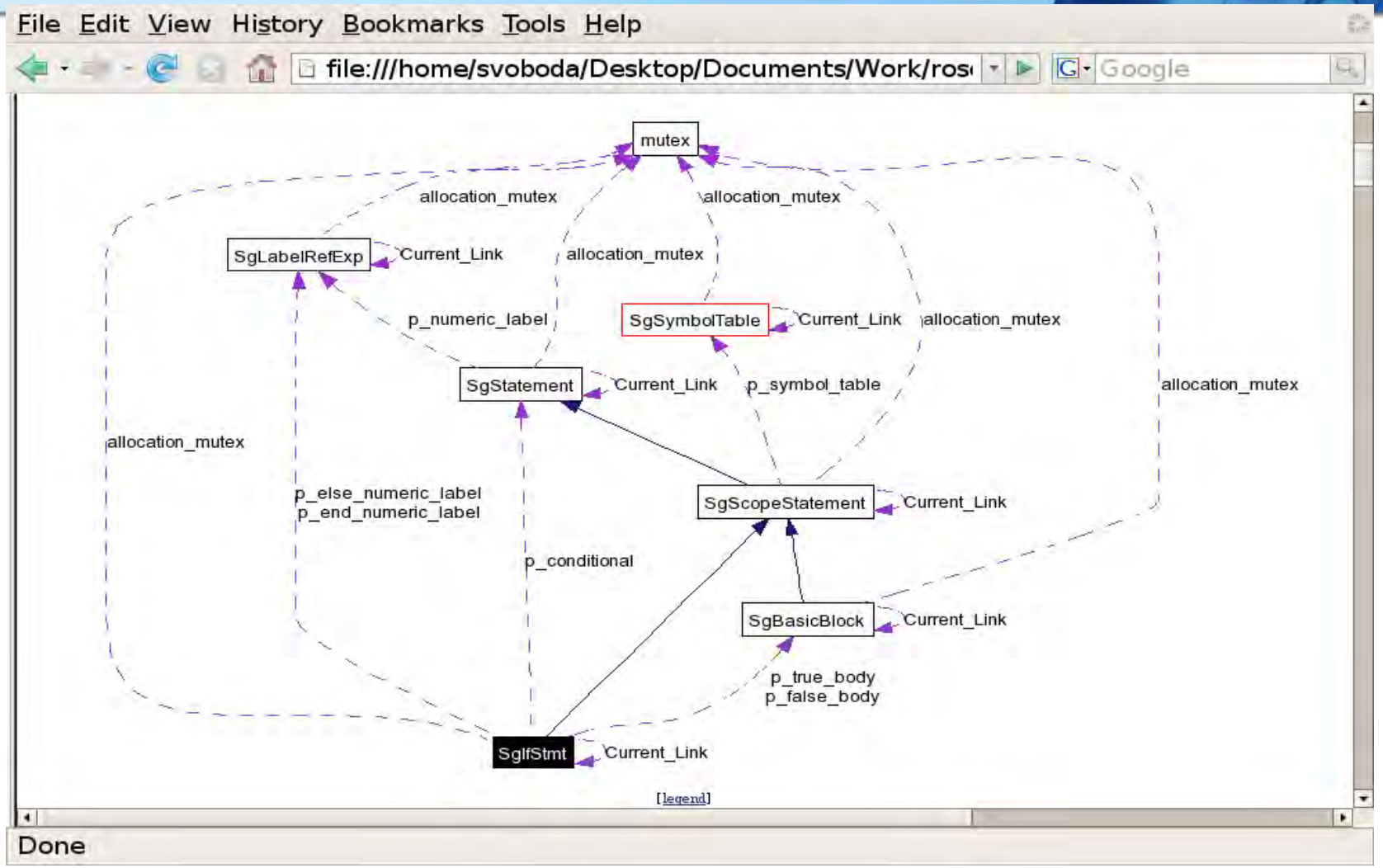
ROSE Web Reference(2)

ROSEの
ドキュメント



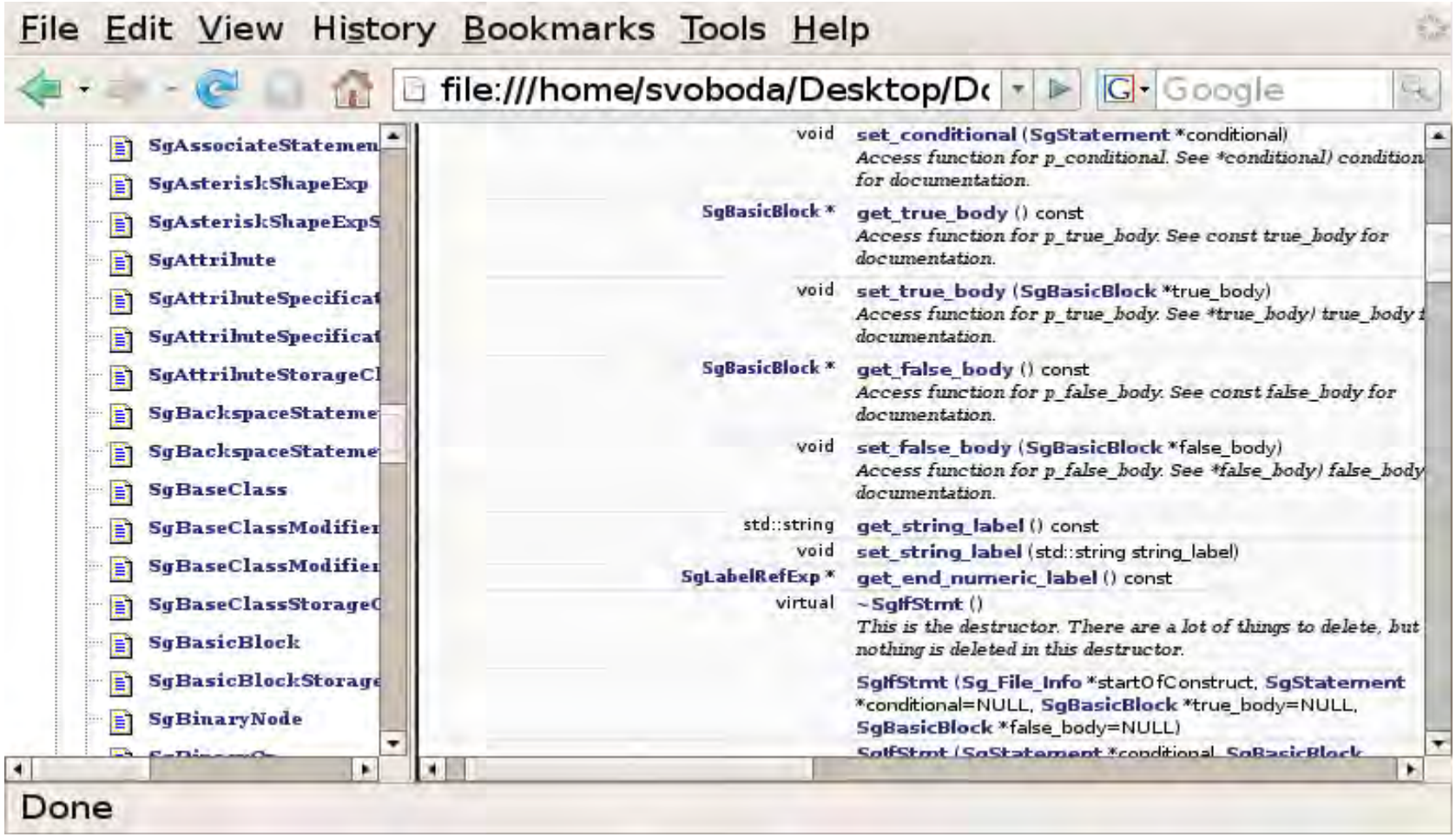
ROSEの
ドキュメント

ROSE Web Reference(3)



ROSE Web Reference(4)

ROSEの
ドキュメント



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

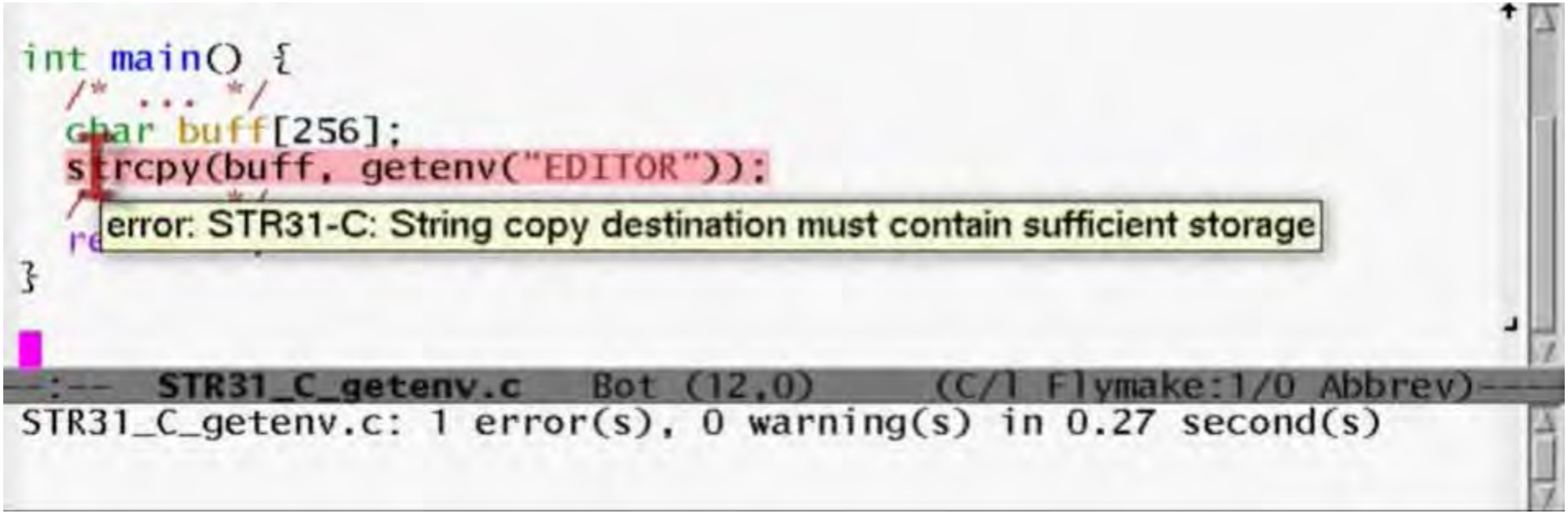
チェッカーの動作確認に使うため,
~/workspace/rosecheckers/test/ ディレクトリに
適合コード例と違反コード例が置いてある.

- 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

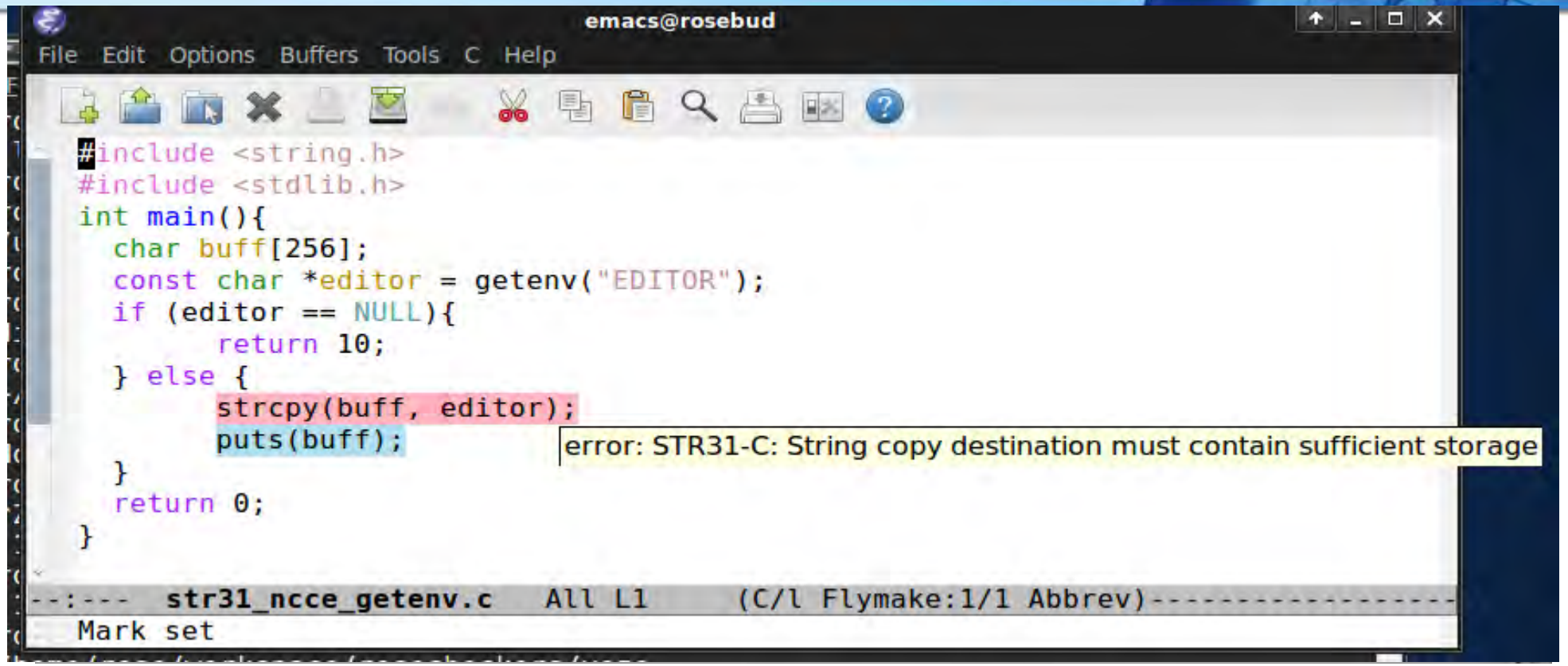
コーディングルールのページに掲載されているコード例だが, 最新版に追従してるわけではない……

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

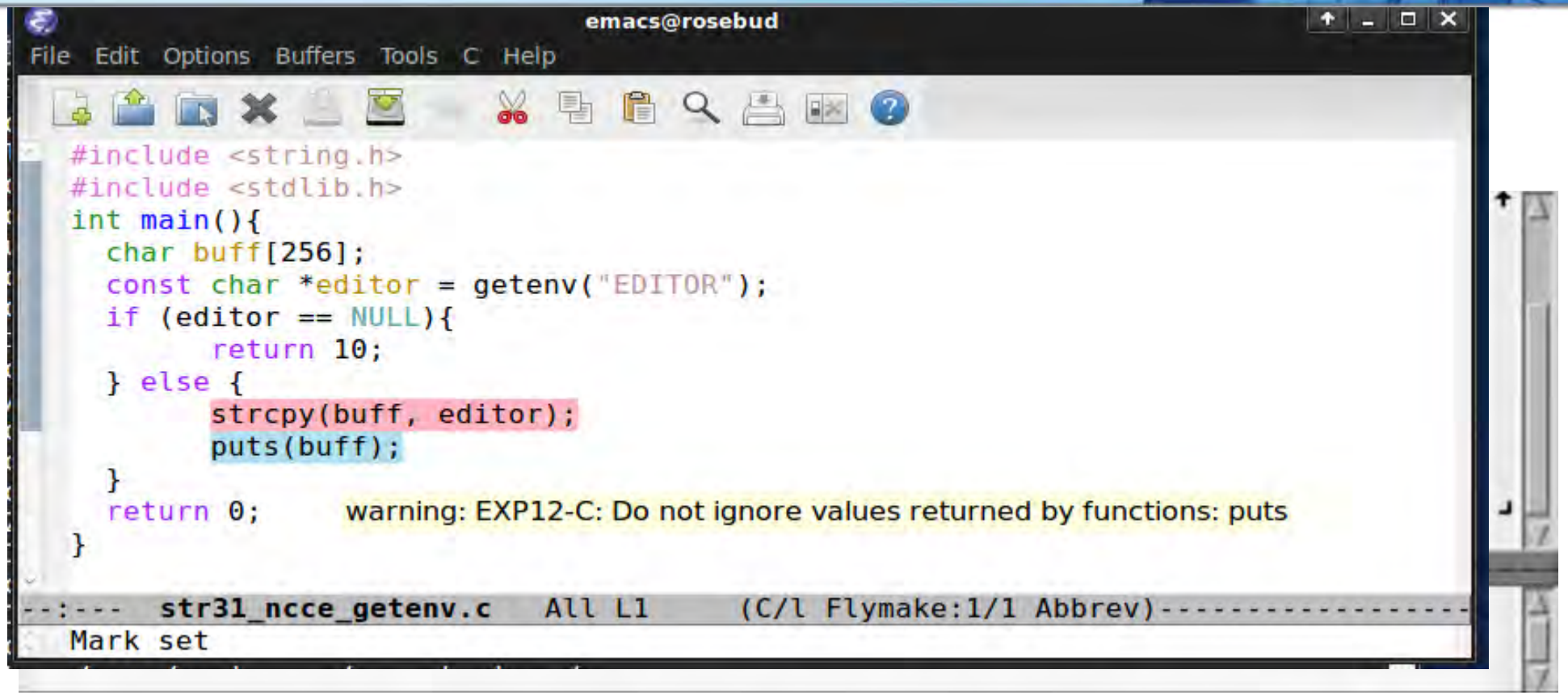


Emacs から rosecheckers を使う(2)



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)



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.

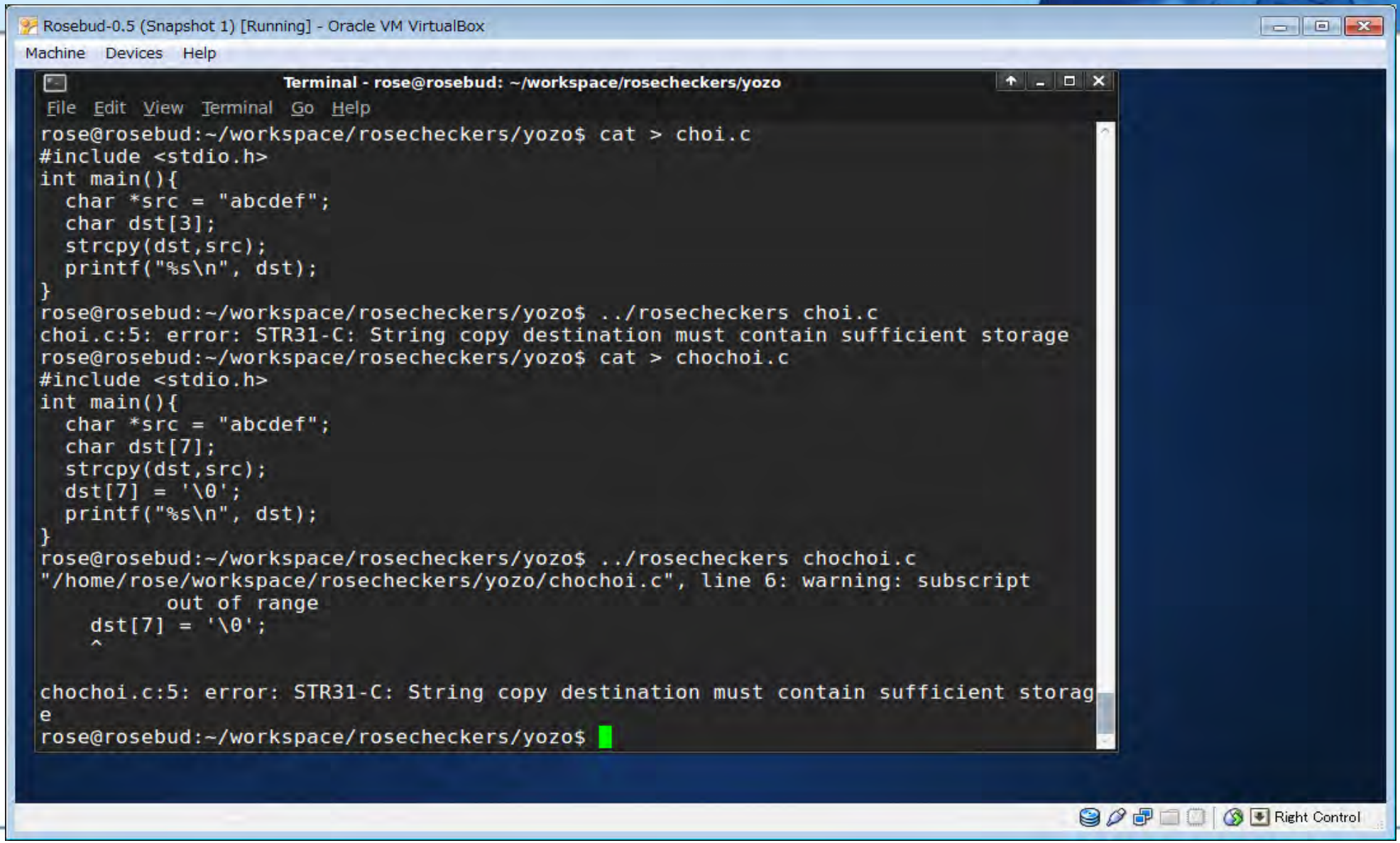
Emacs から rosecheckers を使う(4)

JPCERT CC®

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



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を使ってエクササイズ

JPCERT CC®

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

rosecheckersを使ってエクササイズ

JPCERT CC®

Hello world プログラム

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

rosecheckersを使ってエクササイズ

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”);
}
```

rosecheckersを使ってエクササイズ

JPCERT CC®

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”);
}
```

rosecheckersを使ってエクササイズ

JPCERT CC®

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);
}
```

rosecheckersを使ってエクササイズ

JPCERT CC®

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);
}
```

rosecheckersを使ってエクササイズ

JPCERT CC®

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);
}
```

rosecheckersを使ってエクササイズ

- 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?

ASTを図示する

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

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

ps, pdf は evince で表示できます

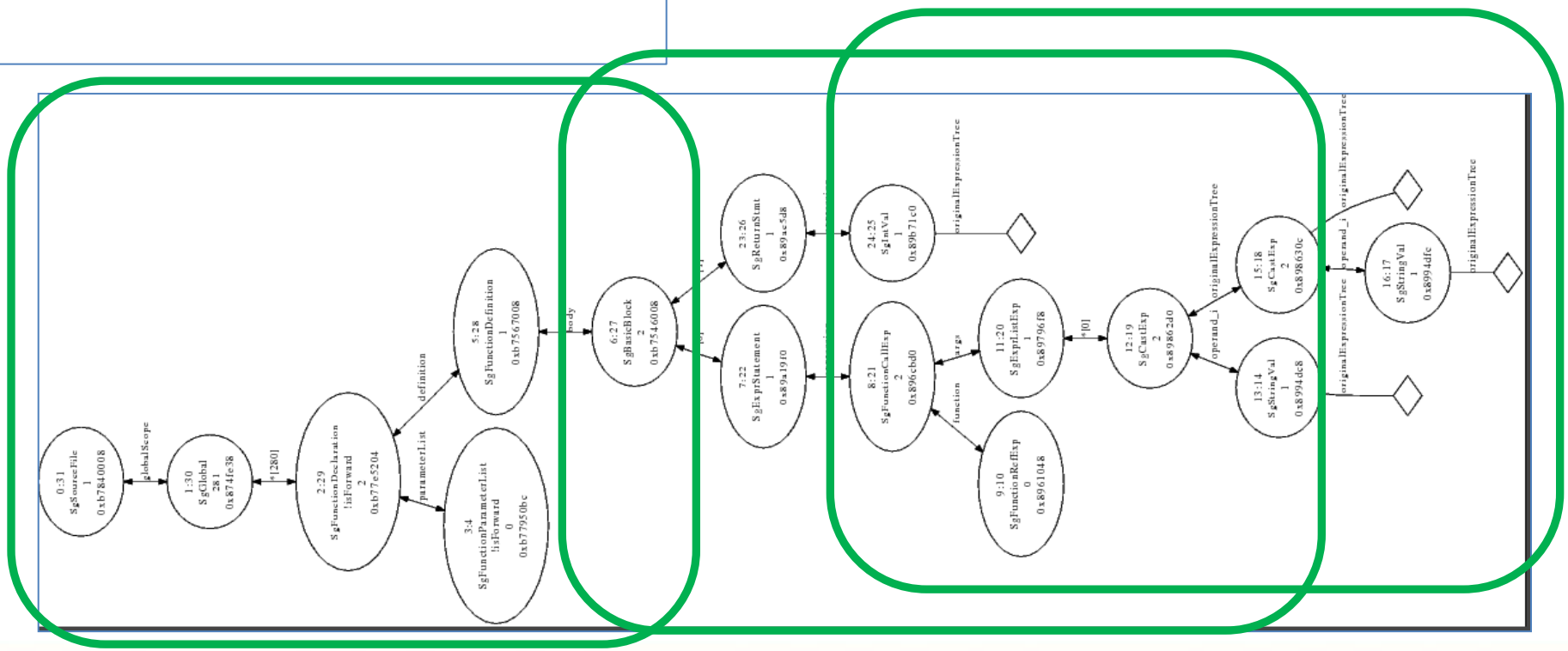
ASTの構造(hello.c その1)

ASTを調べよう

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

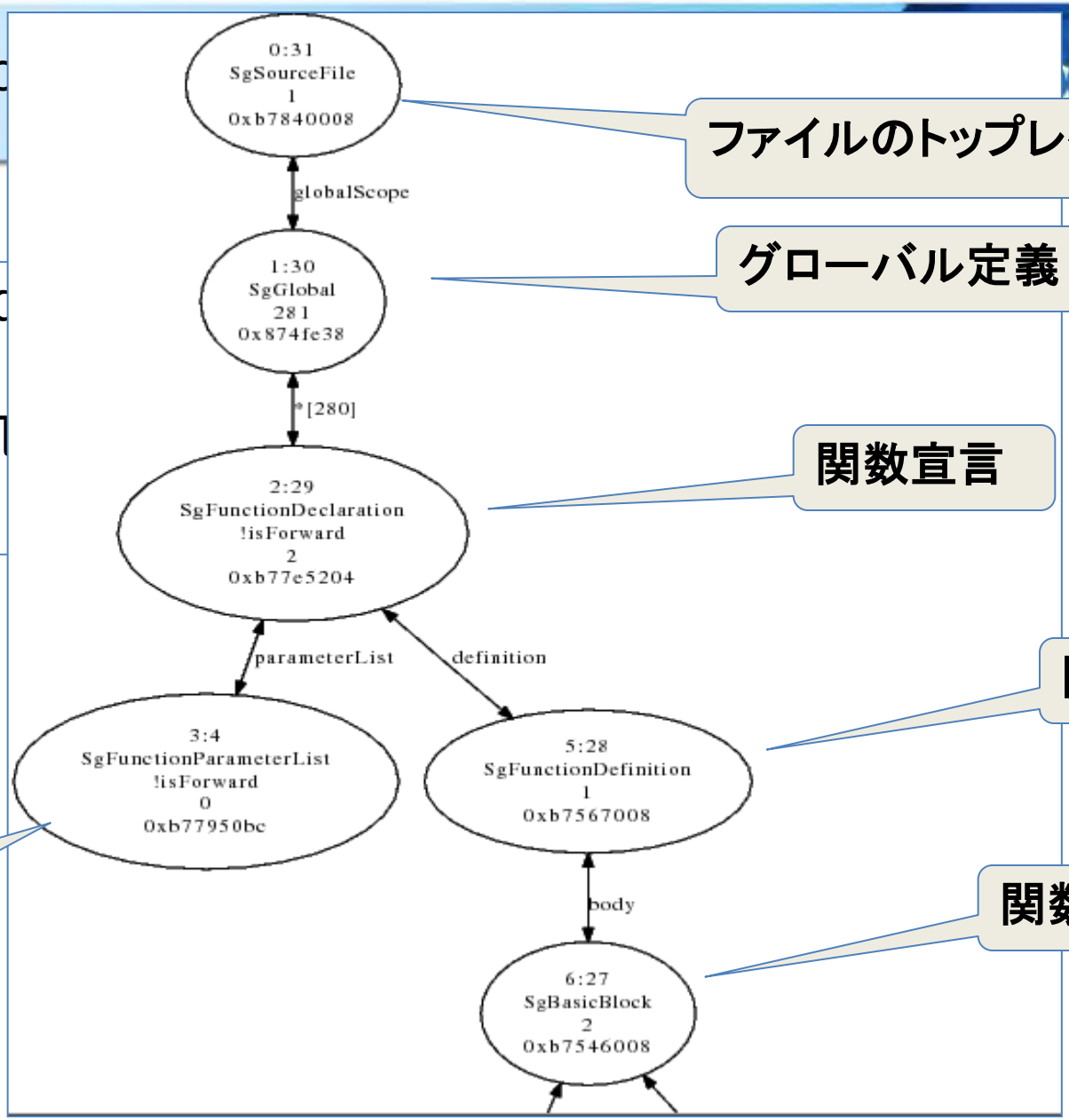


ASTの構造(hello

ASTを調べよう

コード例

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



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

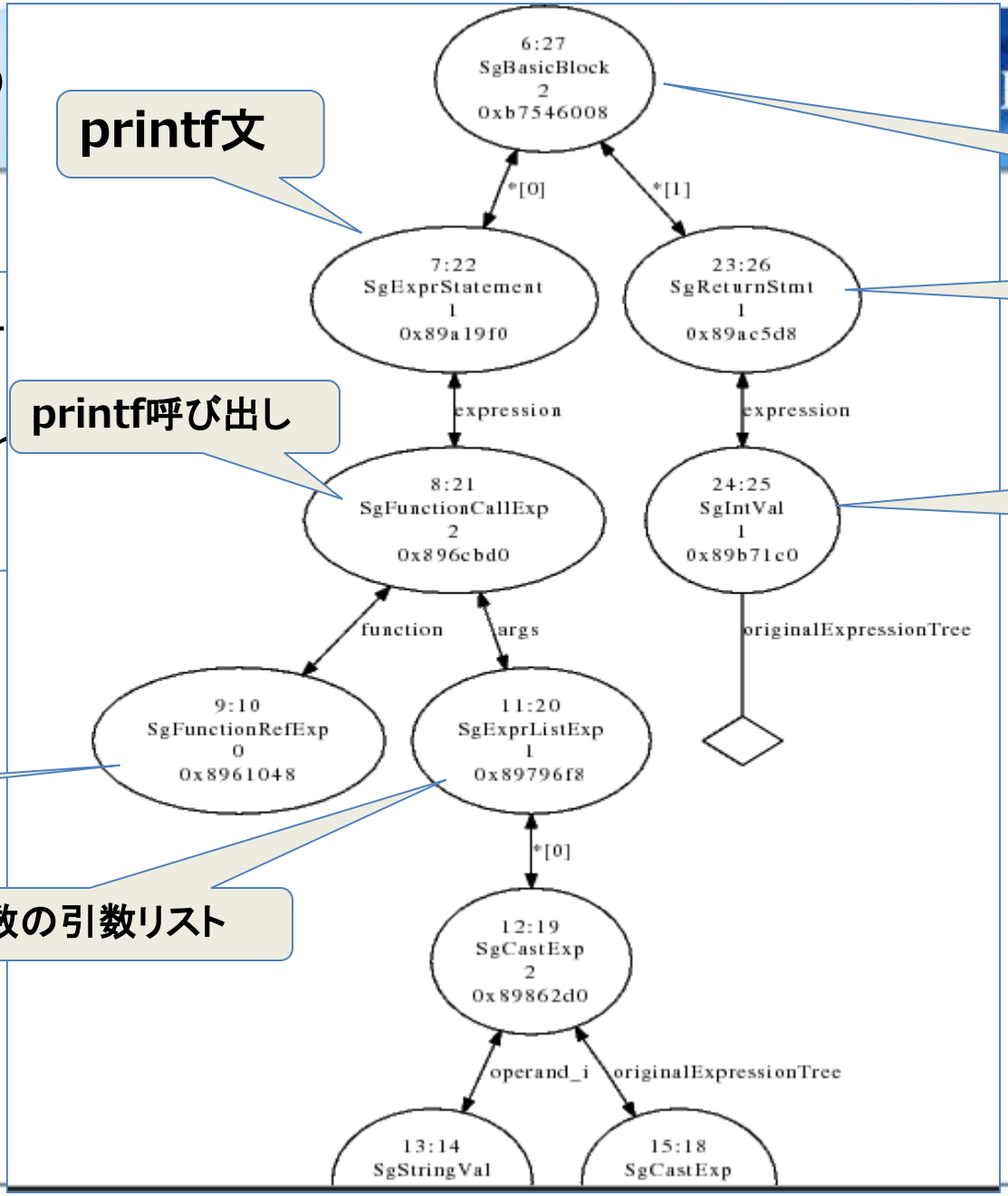
関数本体

関数の引数リスト

ASTの構造(hello

コード例

```
#include <stdi
int main(){
  printf("hello!¥r
}
```



printf文

printf呼び出し

printf関数への参照

printf関数の引数リスト

関数

return文

return文の引数

ASTを調べよう

ASTの構造(hello.c

コード例

```
#include <stdio
int main(){
   printf("hel
}
```

printf呼び出し

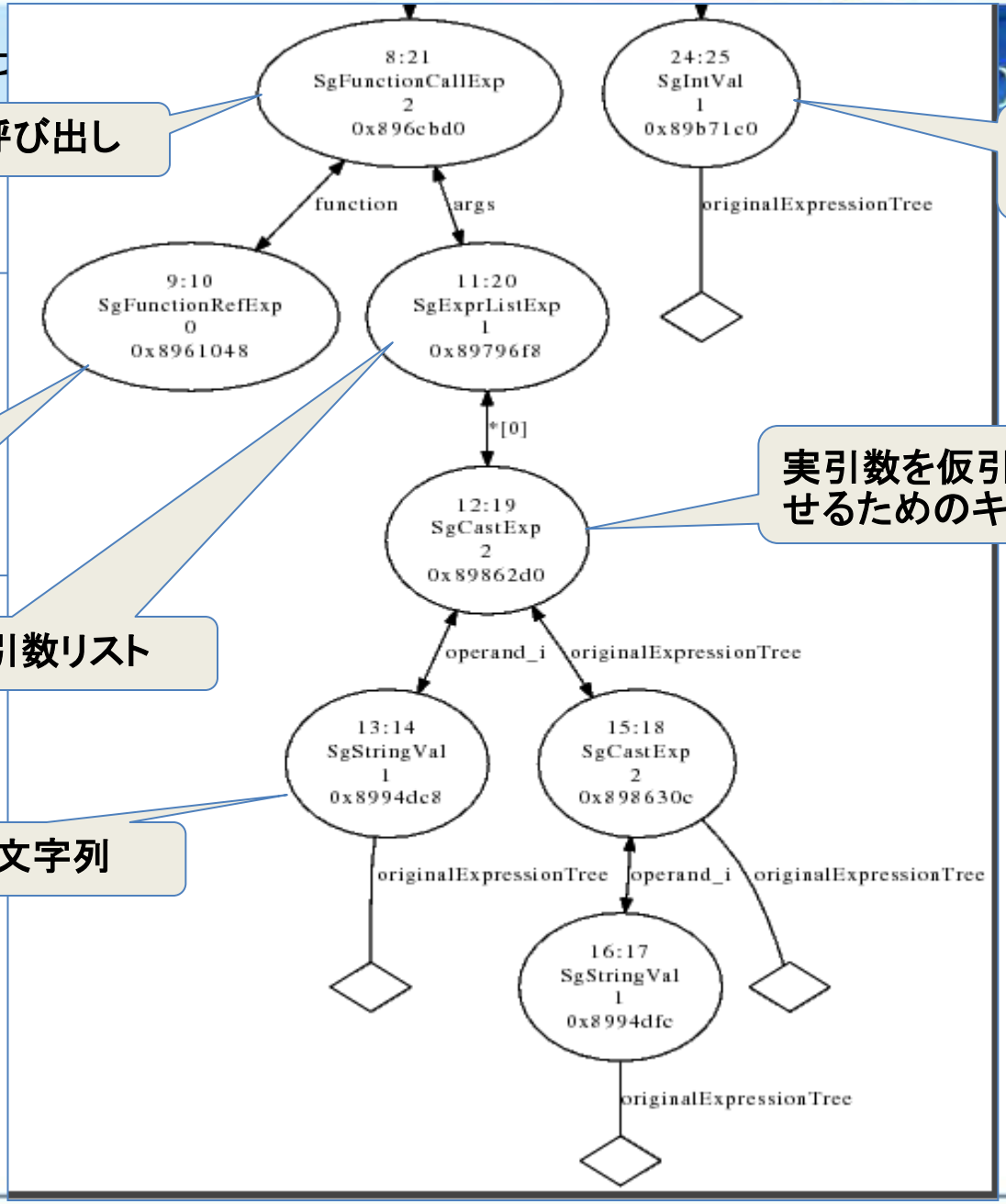
printf関数への参照

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

ASTを調べよう



エクササイズ: ASTの構造を調べる(1)

ASTを調べよう

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)

ASTを調べよう

- 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

JPCER

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`** に出力

```
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

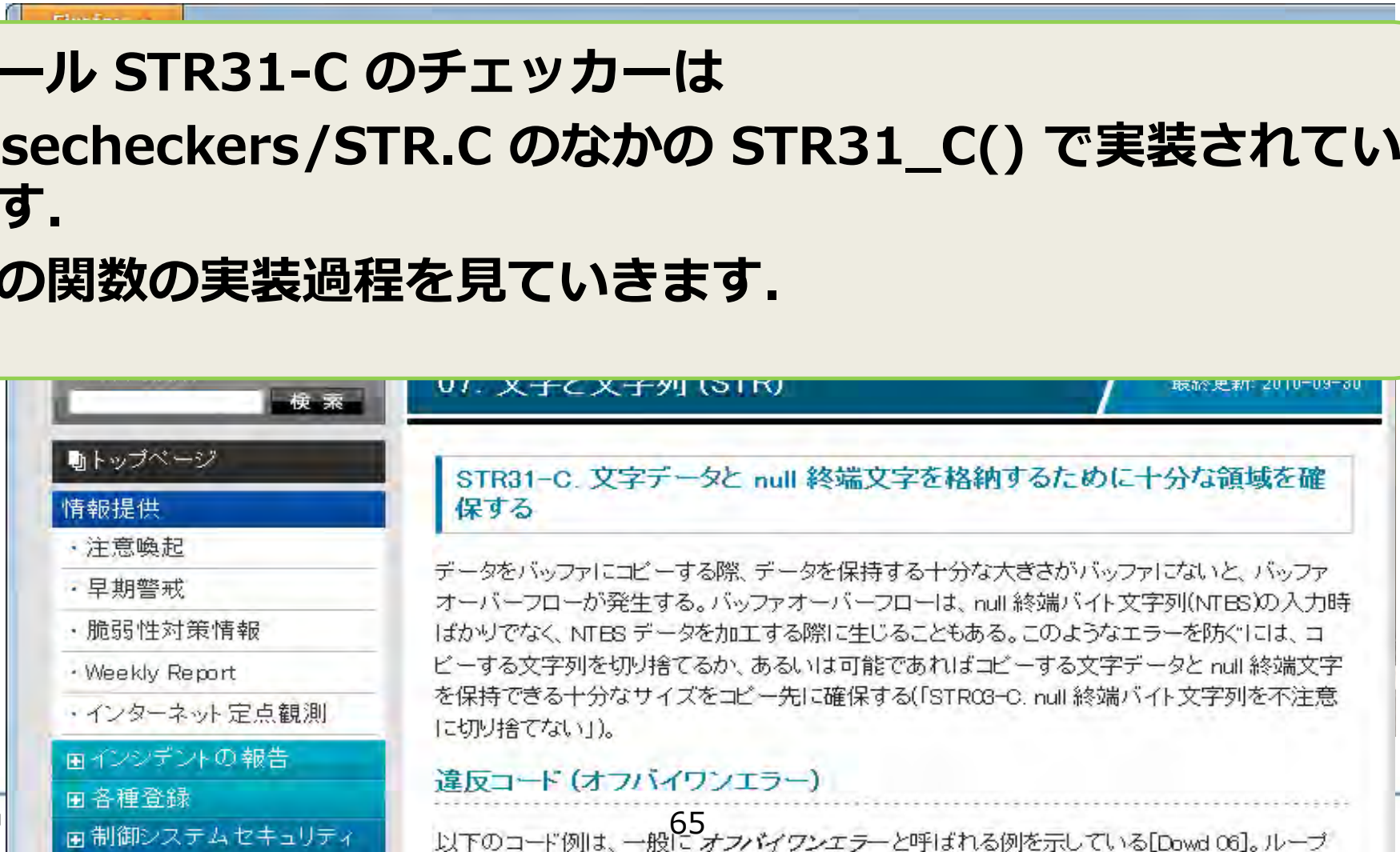
自分でチェッカーを作ってみよう

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.



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. まずは現実に発生しやすいタイプのコード例に注力しよう

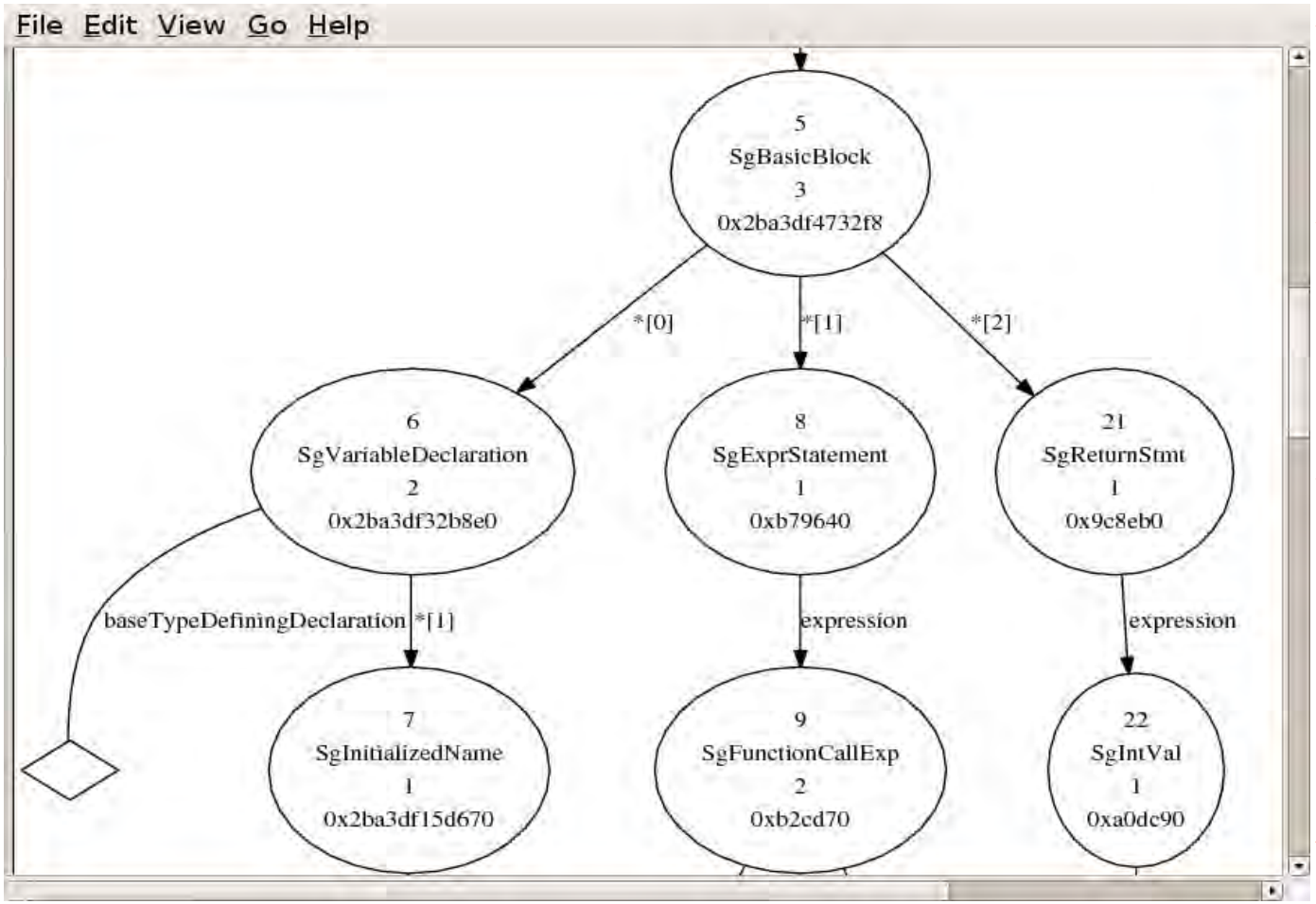
チェッカーを実装してみることにより, セキュアコーディングルールの内容を別の視点で見ることができ, 理解が深まる

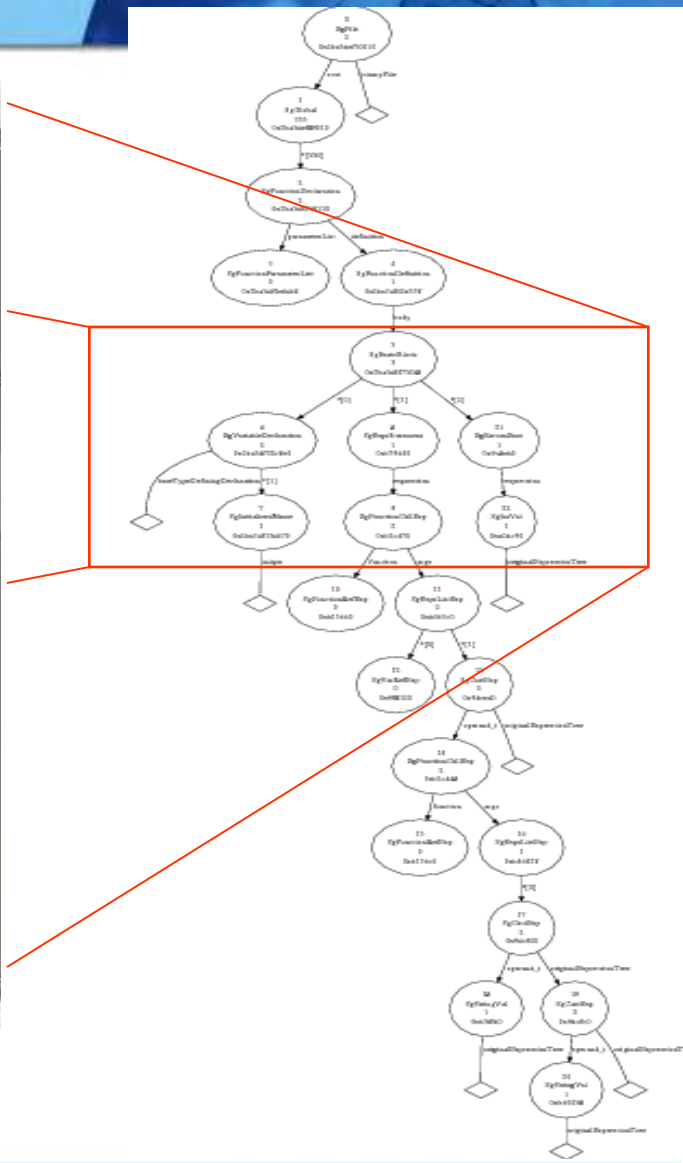
Abstract Syntax Tree 1

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

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

Abstract Syntax Tree 2

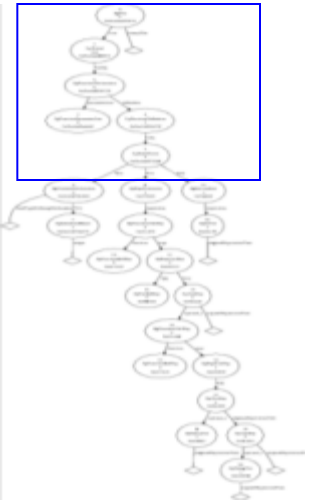


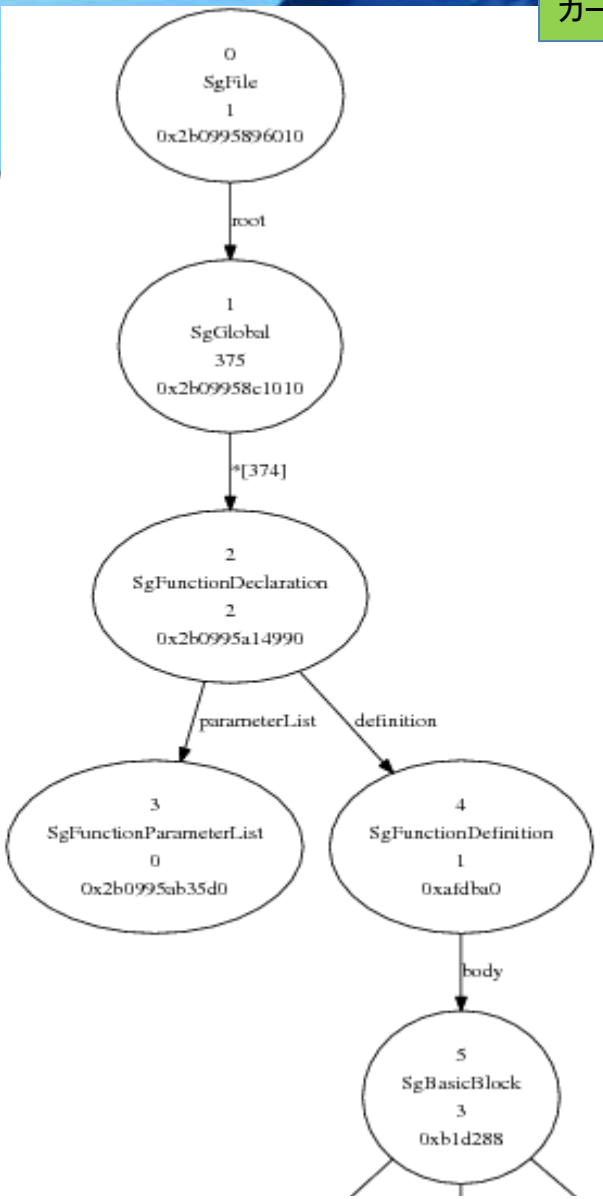


Abstract Syntax Tree 3

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

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

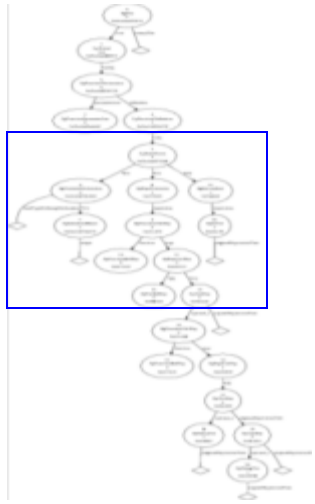




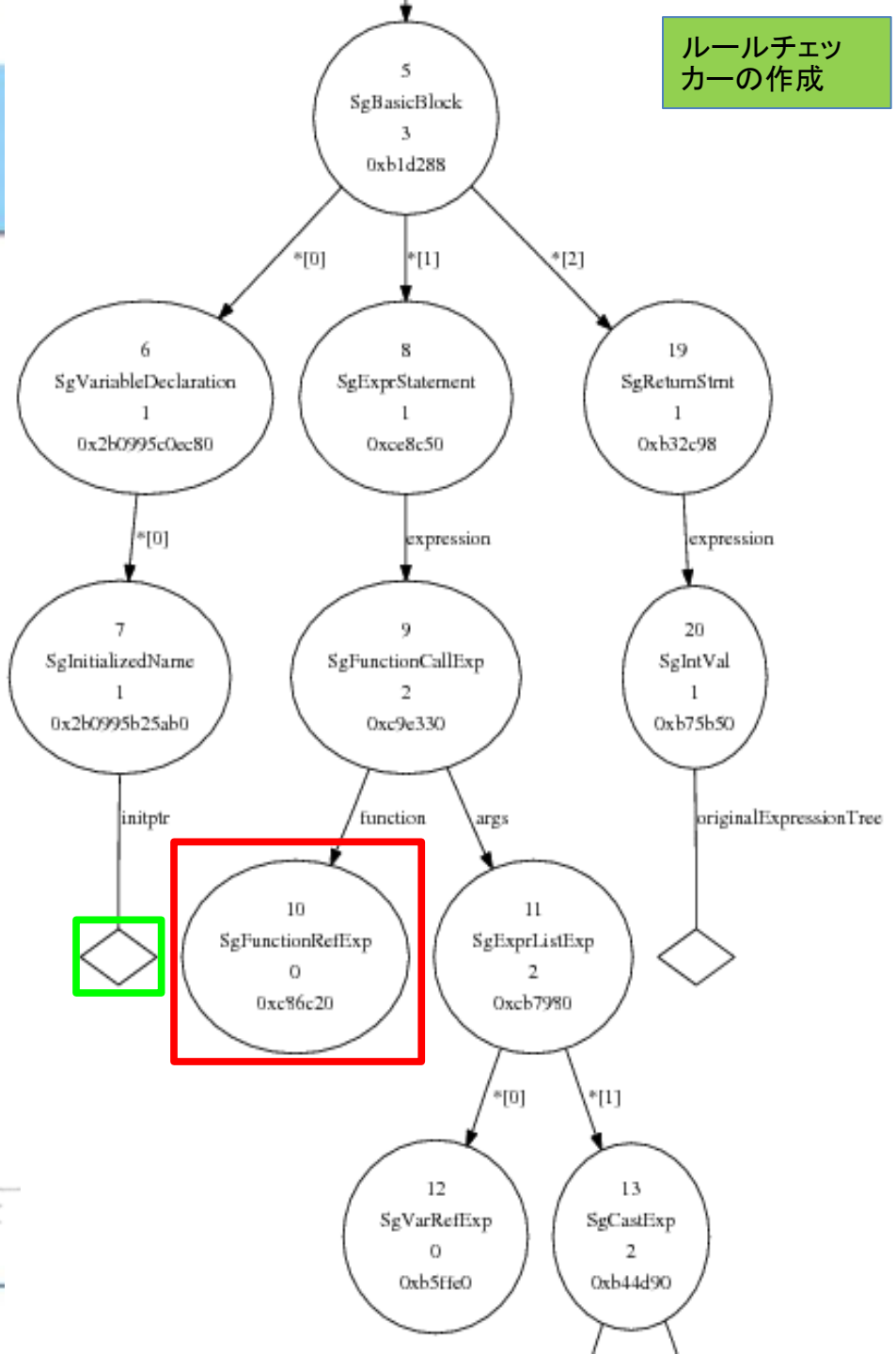
Abstract Syntax Tree 4

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

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



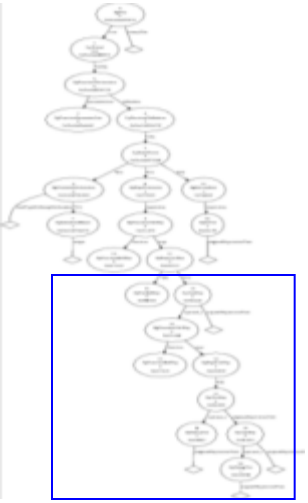
ルールチェッカーの作成

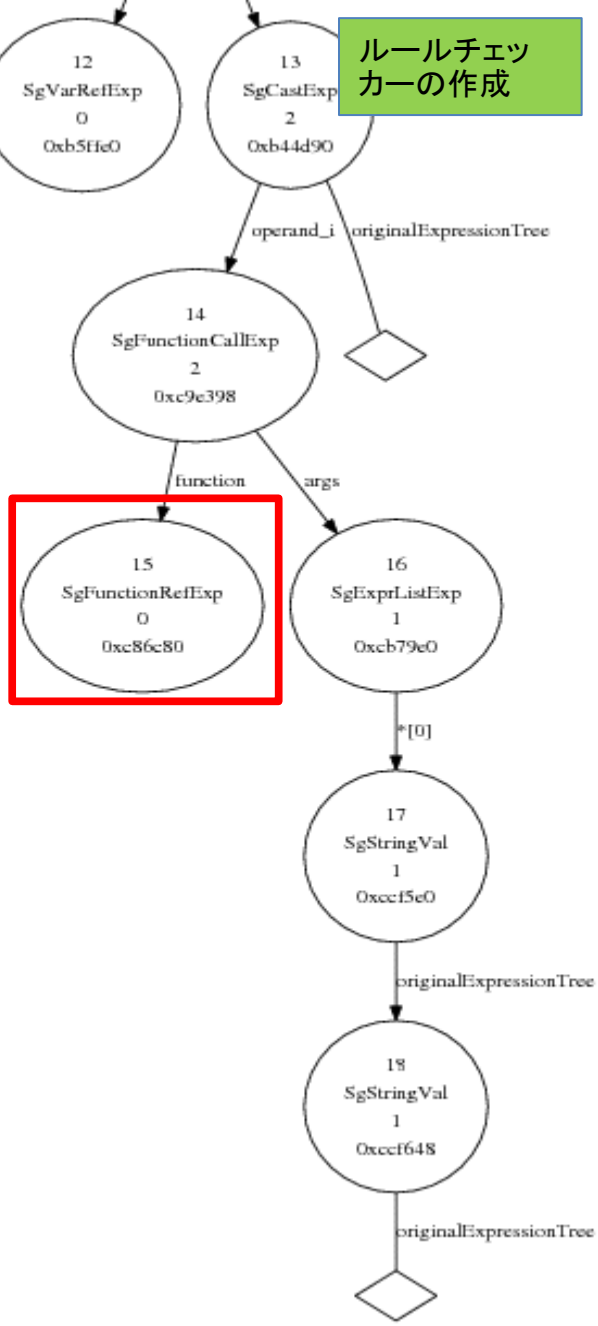


Abstract Syntax Tree 5

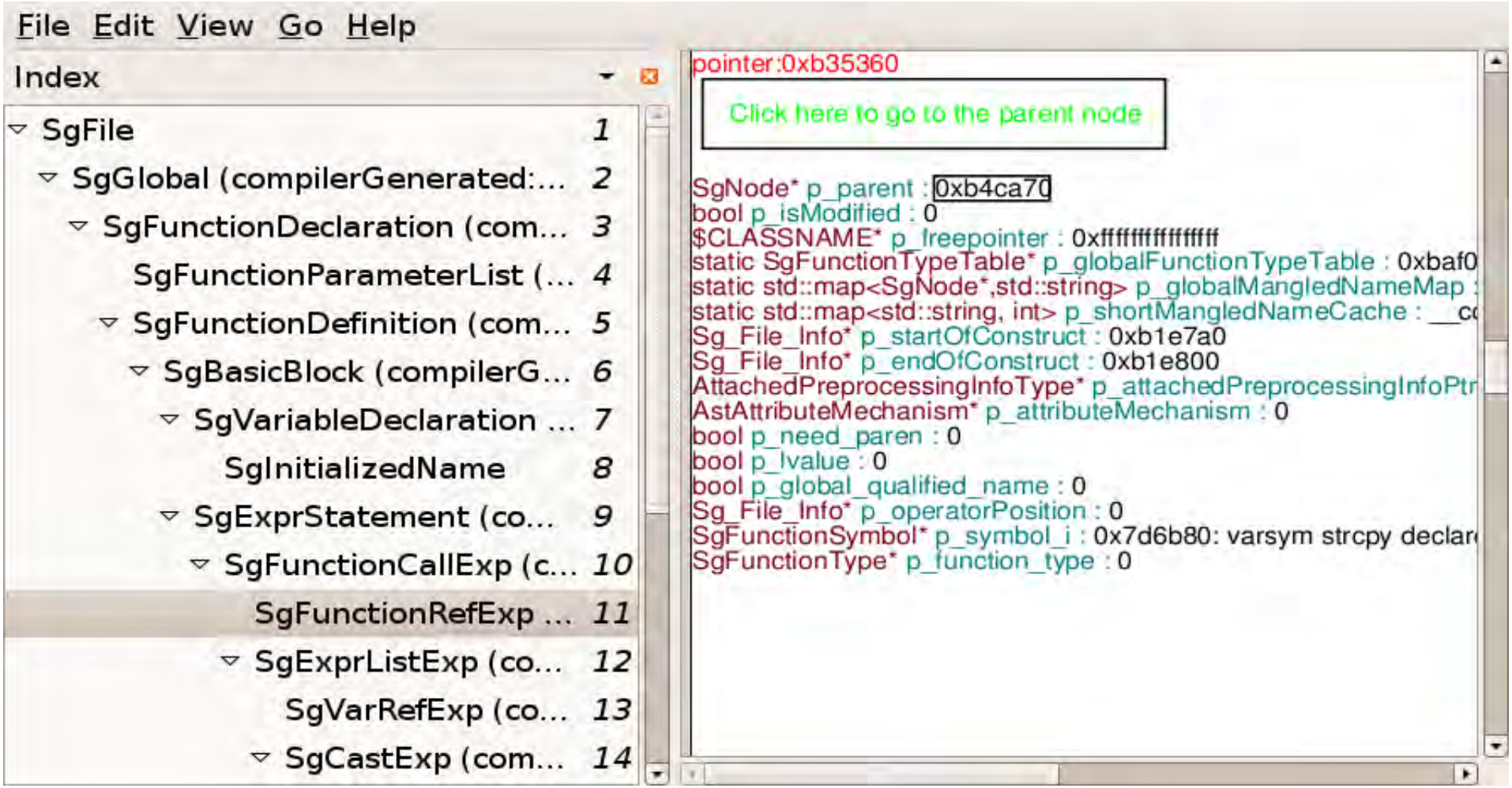
```
#include <string.h>
#include <stdlib.h>

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





ASTノードの属性値



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.

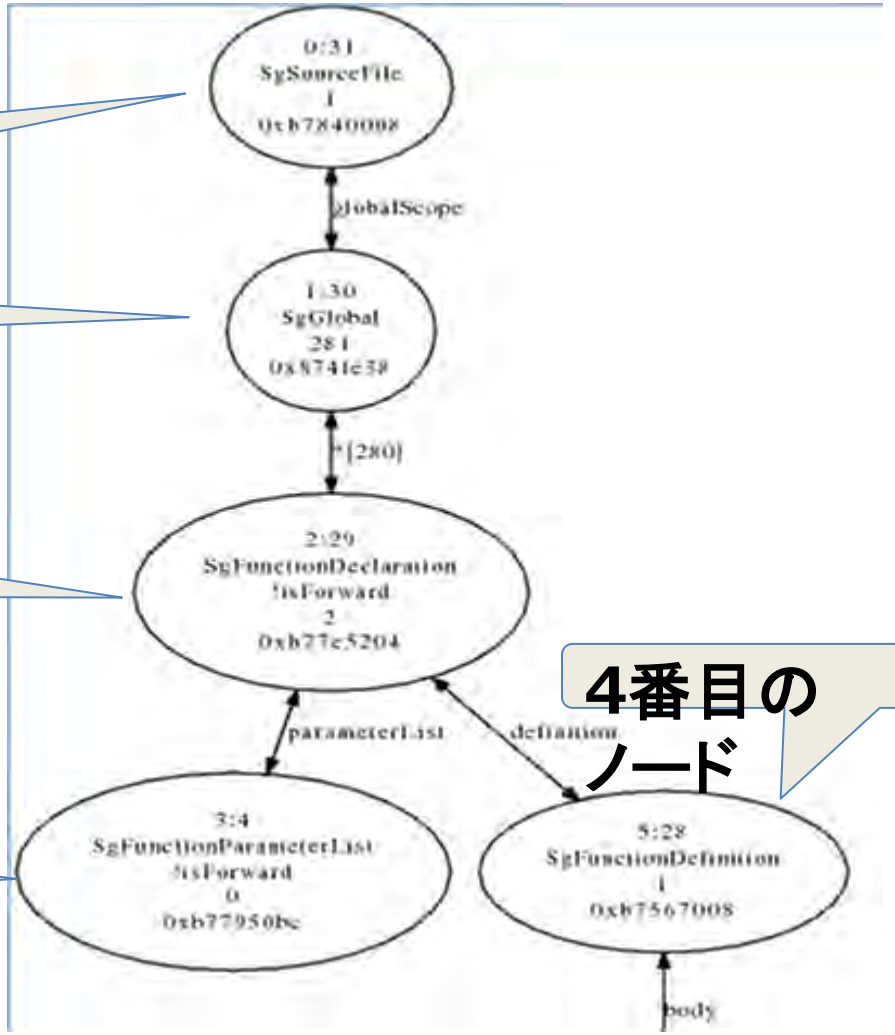
ROSEの Simple Traversal 機能を使っている

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

rosecheckers.C の main()関数

```
#include "rose.h"
int main(int argc, char* argv[]) {
  SgProject* project = frontend(argc,argv);
  ROSE_ASSERT( project );
  AstSimpleProcessing* visitorTraversal;
  ……
  visitorTraversal->traverseInputFiles(
                project, preorder);
  delete visitorTraversal;
  return 0;
}
```

ROSE がソースコードを解析して AST を生成

AST を辿り, 各ノードを調べる

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
4. 第二引数はポインタ型である

DONE

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列

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.