プログラミング言語 Ring

フリー・オープンソース

[公式サイト] http://ring-lang.net/
[日本語版 (非公式)] http://ring-lang-081.osdn.jp/

目次 :

[1] 動機
[2] 特徴
[3] なぜ Ring なのか？
[4] 実用例

動機

- 次世代版 Programming Without Coding Technology (PWCT) ソフトウェアの開発。
- 現行版の PWCT 1.9 は Windows 用に開発 (250,000行のコード、17,000,000 回以上のダウンロードと 200,000 人以上の愛用者)。
- 開発環境の作成、コンポーネント用スクリプトの記述、およびアプリケーション作成用言語を一本に絞ることを決めました。

動機

- C, C++, Java, C#, Lua, PHP, Python と Ruby などのプログラミング言語を調査しました。

- C, C++ 言語は目的とは違います。初心者や専門家向けの使いやすいビジュアル・プログラミング環境、さらに C, C++ 言語よりも高水準の生産性が欲しいのです。

動機

- Java と C# (Mono) は目的とは違い過ぎます！　静的型付け、マルチプラットフォームですが使いたいのは動的言語であり、膨大なクラスとオブジェクト指向を強要する冗長なプログラミング言語は不適切です。動的言語を使いたいのです。大規模アプリケーションの開発用に設計してあり、ガベージコレクター (GC) 制御の良くできている高速で生産性の高い小規模な動的プログラミング言語です。

動機

- Lua は小規模で高速ですが機能不足です。大規模アプリケーション開発用にもっと強力な言語が必要です。
- PHP はシンプルではなく一般的ではないので違います。シンタックスは C と非常に似ていますが、複雑怪奇なウェブ開発用言語です。
- Python と Ruby は目的に近いのですが、規模、動作速度、生産性に不満があります。

動機

- Python と Ruby は英数大小文字を区別します。リストのインデックス計数は0から始まり、関数は呼び出し前に定義する必要があります。 Ruby はオブジェクト指向とメッセージパッシングを多用するため、必然的に実行性能は低下します。 Python のシンタックス (インデント、self の使用、pass & _) は目標としているものではありません。

動機

- 紹介したプログラミング言語は、全て成功を収めたものであり、対象となる問題解決領域では非常に優れたものですが、求めているものは、新発想の異次元プログラミング言語です (画期的、即戦力、単純明快、柔軟かつ高速。さらに知性的な実装)。

特徴

- 自由なオープンソース (MIT ライセンス)
- ANSI C で記述。
- 単純明快、小規模、柔軟かつ高速。
- 移植性 (Windows, Linux と macOS, Android など)
- デスクトップ、ウェブ、モバイルとゲームを開発可能。
- マルチパラダイム (命令型、手続き型、オブジェクト指向、関数型、メタプログラミング、宣言型と自然言語)
- プログラミング言語 Supernova の後継言語
- C,C++, Java, C#, Lisp, Smalltalk, Lua, PHP, Python, Ruby, Harbour, Visual FoxPro, QML と REBOL などの言語から様々な教訓を得ています。

なぜ Ring なのか？

- プログラミング言語 Ring には簡明、違和感の排除、組織化の奨励、および透過性とビジュアル実装があります。簡潔なシンタックス、そして自然なインタフェースの作成を可能にする機能群、プログラマが短時間で作成・構築できる宣言型ドメイン特化言語機能を標準装備しています。非常に小規模、高速なスマートガベージコレクターにより、プログラマはメモリを制御下に置くことができます。また、多種多様なプログラミングパラダイムに対応しており、便利で実用的なライブラリが付属しています。Ring は生産性と拡張性に優れた高品質な解決方法の開発のために設計しました。

明確な設計目標

- アプリケーション開発用のプログラミング言語です。
- 生産性と拡張性に優れた高品質な解決方法の開発。
- 小規模・高速な言語で C/C++ プロジェクトへ組み込めます。
- 学習と入門 (文教用途、およびコンパイラ・仮想計算機の概念) に使用できる単純明快な言語です。
- ドメイン特化ライブラリ、フレームワーク、およびツールを作成できる汎用プログラミング言語です。
- ビジュアル・プログラミング言語 Programming Without Coding Technology (PWCT) ソフトウェアの次世代版の開発用に設計した実用プログラミング言語です。

簡明

- Ring は非常に簡明な言語であり、非常に単純明快なシンタックスで構成しています。プログラマには、ボイラープレートコードのないプログラムの記述を奨励しています。

```
See "Hello, World!"
```

簡明

- Main 関数はオプション扱いであり、ステートメントの後に実行するため、ローカルスコープで便利です。

```
Func Main
        See "Hello, World!"
```

簡明

- 動的型付け、およびレキシカルスコープを使用しています。変数名の先頭に $ は不要です！
  文字列の連結は ‘+’演算子です。弱い型付け言語であり、文字列はコンテキストに基づいて数値と文字列との間で自動的に変換します。

```
nCount = 10     # Global variable
Func Main
        nID = 1 # Local variable
        See "Count = " + nCount + nl + " ID = " + nID
```

違和感の排除

- Ring は英数大小文字を区別しません。

```
See "Enter your name ? "
Give name
See "Hello " + Name      # Name is the same as name
```

- リストのインデックス (添字番号) は 1 から開始します。

```
aList = ["one","two","three"]
See aList[1]     # print one
```

違和感の排除

- 定義前の関数呼び出し:

```
one()
two()
three()
Func one
        See "One" + nl
Func two
        See "two" + nl
Func three
        See "three" + nl
```

違和感の排除

- 代入演算子は深いコピーを使用します (この操作は参照ではありません)。

```
aList = ["one","two","three"]
aList2 = aList
aList[1] = 1
see alist[1]     # print 1
see aList2[1]    # print one
```

違和感の排除

- 数値と文字列は値渡しですが、リストとオブジェクトは参照渡しです。For in ループ でリストの項目 (アイテム、要素とも言います) を更新できます。

```
Func Main
        aList = [1,2,3]
        update(aList)
        see aList          # print one two three

Func update aList
        for x in aList
                switch x
                on 1 x = "one"
                on 2 x = "two"
                on 3 x = "three"
                off
        next
```

違和感の排除

- 定義時のリスト使用:

```
aList = [ [1,2,3,4,5] , aList[1] , aList[1] ]
see aList        # print 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

- 一階層以上のループから脱出

```
for x = 1 to 10
        for y = 1 to 10
                see "x=" + x + " y=" + y + nl
                if x = 3 and y = 5
                        exit 2      # exit from 2 loops
                ok
        next
next
```

組織化の奨励

- Ring ではプログラムの組織化を奨励しています。まずは関数、次にクラス、そして、関数とヘンテコなモノと組み合わせるプログラミング言語を使用していた悪夢の日々を忘却の彼方へ追いやります！　ソースファイルの構造は下記の通りです:
- ファイルの読み込み
- ステートメントとグローバル変数
- 関数
- パッケージとクラス
- これにより、構成要素 (コンポーネント) で end キーワードを記述しなくてもパッケージ、クラスと関数を使えます。

コメント

- 一行コメント、または複数行コメントを使えます。
- 一行コメントは # または // で始まります。
- 複数行コメントは /* ～ */ の間に記述します。

```
/*
        Program Name : My first program using Ring
        Date         : 2015.05.08
*/

See "What is your name? "        # print message on screen
give cName                       # get input from the user
see "Hello " + cName     # say hello!

// See "Bye!"
```

透過型実装

- Ring は透過型実装です。コンパイラの処理段階、および仮想計算機による実行中の処理内容を把握できます。
- 例えば : ring helloworld.ring -tokens -rules -ic

```
See "Hello, World!"
```

透過型実装

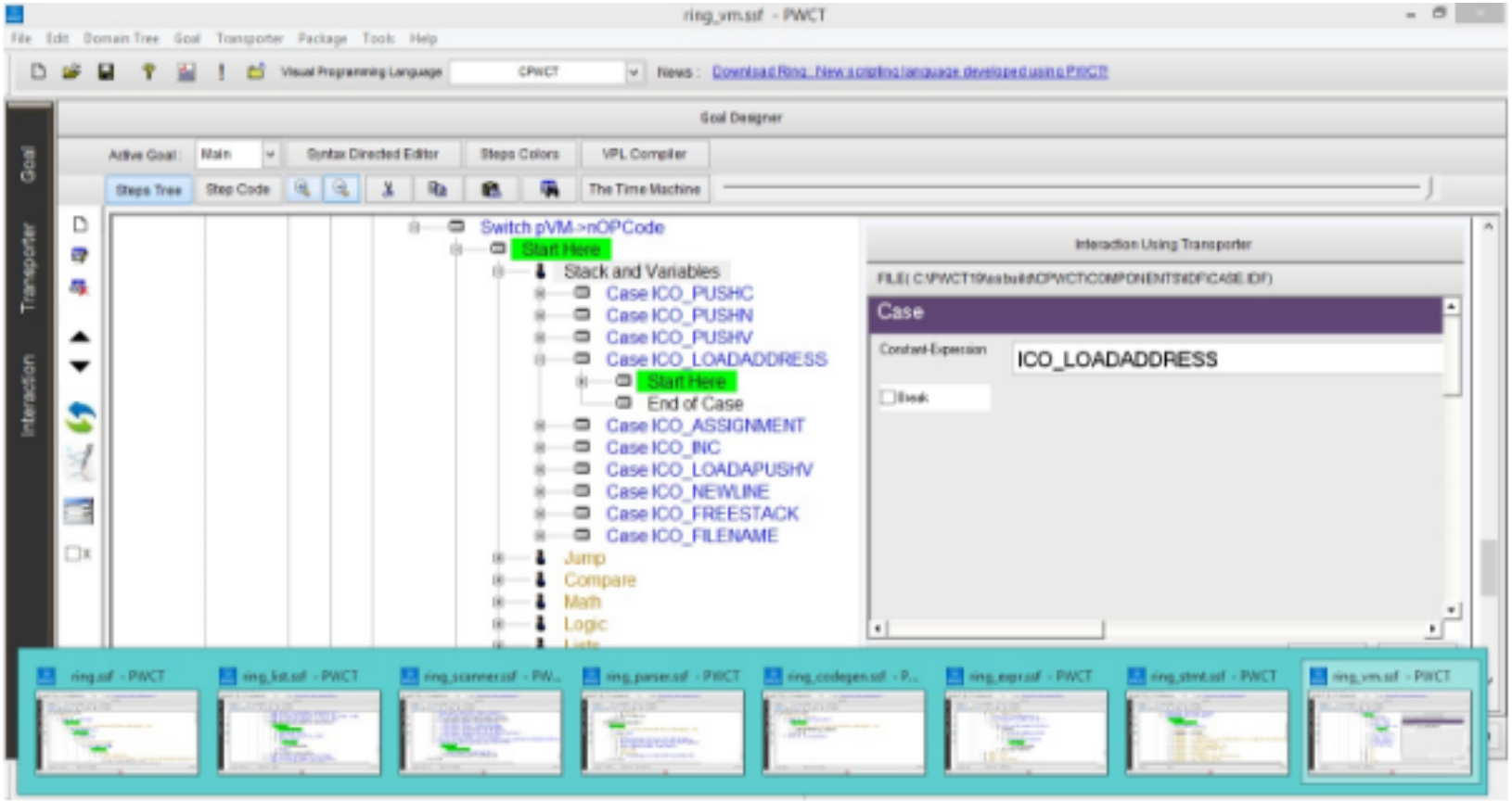
```
==========================================================================
Tokens - Generated by the Scanner
==========================================================================

    Keyword : SEE
    Literal : Hello, World!
    EndLine

==========================================================================

==========================================================================
Grammar Rules Used by The Parser
==========================================================================

Rule : Program --> {Statement}

Line 1
Rule : Factor --> Literal
Rule : Range --> Factor
Rule : Term --> Range
Rule : Arithmetic --> Term
Rule : BitShift --> Arithmetic
Rule : BitAnd --> BitShift
Rule : BitOrXOR -->  BitAnd
Rule : Compare --> BitOrXOR
Rule : EqualOrNot --> Compare
Rule : LogicNot -> EqualOrNot
Rule : Expr --> LogicNot
Rule : Statement  --> 'See' Expr

==========================================================================
```

透過型実装

```
==========================================================================
Byte Code - Before Execution by the VM
==========================================================================

     PC       OPCode         Data

      1      FuncExE
      2        PushC    Hello, World!
      3        Print
      4   ReturnNull

==========================================================================

Hello, World!
```

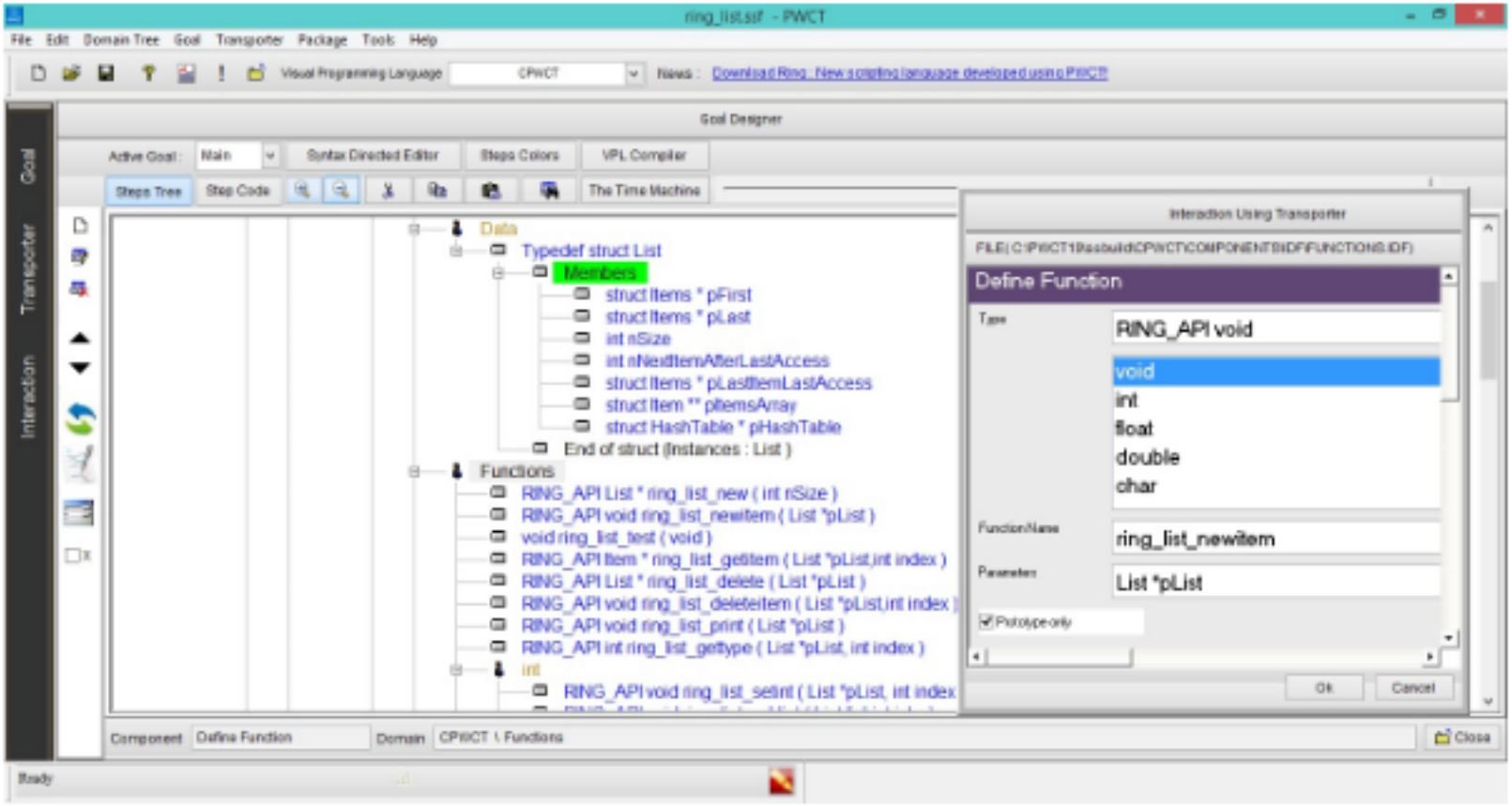
ビジュアル実装

- Ring は、ビジュアル・プログラミングツール
  Programming Without Coding Technology (PWCT) で設計しました。



ビジュアル実装

- Ring は、ビジュアル・プログラミングツール
  Programming Without Coding Technology (PWCT) で設計しました。



簡潔なシンタックス

- **行の区別はしません**ので、ステートメントの後に ; は不要です。また、ENTER や TAB の入力は不要であるため、このようなコードも記述できます。

```
See "The First Message" See " Another message in the same line! " + nl
See "Enter your name?" Give Name See "Hello " + Name
```

簡潔なシンタックス

- このコードは三種類の属性 X, Y および Z を有する Point クラスを作成します。**クラス、パッケージ、関数の定義を終了するために end キーワードは使用していません。**また、**クラス名の直下に属性名を書くことができます。**

```
Class Point X Y Z
```

簡潔なシンタックス

- **定義前にクラスと関数を使えます。**この用例では、オブジェクトの新規作成と属性の設定、および値を表示します。

```
o1 = New point  o1.x=10    o1.y=20   o1.z=30    See O1  Class Point X Y Z
```

簡潔なシンタックス

- ドット演算子 '.'でオブジェクトの属性とメソッドへアクセスするのではなく、**括弧 { } でオブジェクトへアクセスできます。その後にオブジェクトの属性とメソッドを使えます。**

```
o1 = New point { x=10 y=20 z=30 } See O1  Class Point X Y Z
```

簡潔なシンタックス

- メソッドの呼出し後に { } でオブジェクトへアクセスします。

```
oPerson = new Person
{
        Name = "Somebody"
        Address = "Somewhere"
        Phone = "0000000"
        Print()                 # here we call the Print() method
}
Class Person Name Address Phone
        Func Print
                See "Name :" + name + nl +
                    "Address :" + Address + nl +
                    "Phone : " + phone + nl
```

簡潔なシンタックス

- { } で、オブジェクトへアクセスしてからオブジェクト名を記述するとき、**自動的に呼び出される全ての setter/getter メソッドに対してクラスを検査します。**

```
New Number {
                See one          # Execute GetOne()
                See two          # Execute GetTwo()
                See three        # Execute GetThree()
}
Class Number one two three
        Func GetOne
                See "Number : One" + nl
                return 1
        Func GetTwo
                See "Number : Two" + nl
                return 2
        Func GetThree
                See "Number : Three" + nl
                return 3
```

オブジェクト指向をベースとした自然言語ステートメントの定義

- { } でオブジェクトへアクセス後に、クラスに BraceEnd() メソッドがあれば BraceEnd() メソッドを実行します！

```
TimeForFun = new journey
# The first surprise!
TimeForFun {
        Hello it is me          # What a beatiful programming world!
}
# Our Class
Class journey
        hello=0 it=0 is=0 me=0
        func GetHello
                See "Hello" + nl
        func braceEnd
                See "Goodbye!" + nl
```

オブジェクト指向をベースとした自然言語ステートメントの定義

- Read(ファイル名) 関数は、テキストファイルを読み取ります。また Write(ファイル名,文字列) 関数はファイルへ書き込みます。

```
See "Enter File Name:" Give cFileName See Read(cFileName) # Print the file content
```

- この用例は、二つの命令を定義するクラスの作成方法です。
  最初の命令は : I want window
  次の命令は : Window title = <式>
  the キーワードなどは無視します。

オブジェクト指向をベースとした自然言語ステートメントの定義

```
New App
{
        I want window
        The window title = "hello world"
}

Class App

        # Attributes for the instruction I want window
                        i want window
                        nIwantwindow = 0
        # Attributes for the instruction Window title
        # Here we don't define the window attribute again
                        title
                        nWindowTitle = 0
        # Keywords to ignore, just give them any value
                        the=0
```

オブジェクト指向をベースとした自然言語ステートメントの定義

- **Eval() 関数は、文字列に記述されたコードを実行します。**

```
cCode = "See 'Code that will be executed later!' "
Eval(cCode)     # execute the code to print the message
```

- リストの作成後に、実行用のコードをリストから生成できます。

```
aWords = ["hello","it","is","me"]
for word in aWords cCode=word+"=0" eval(cCode) next
```

オブジェクト指向をベースとした自然言語ステートメントの定義

```
func geti
        if nIwantwindow = 0
                nIwantwindow++
        ok

func getwant
        if nIwantwindow = 1
                nIwantwindow++
        ok

func getwindow
        if nIwantwindow = 2
                nIwantwindow= 0
                see "Instruction : I want window" + nl
        ok
        if nWindowTitle = 0
                nWindowTitle++
        ok

func settitle cValue
        if nWindowTitle = 1
                nWindowTitle=0
                see "Instruction : Window Title = " + cValue + nl
        ok
```

オブジェクト指向をベースとした自然言語ステートメントの定義

- 別の用例

```
# Natural Code
new program {
        Accept 2 numbers then print the sum
}

# Natural Code Implementation
class program
        # Keywords
                Accept=0 numbers=0 then=0 print=0 the=0 sum=0

        # Execution
        func braceexpreval x
                value = x
        func getnumbers
                for x=1 to value
                        see "Enter Number ("+x+") :" give nNumber
                        aNumbers + nNumber
                next
        func getsum
                nSUm = 0
                for x in aNumbers nSum+= x next
                see "The Sum : " + nSum
        private
                value=0 aNumbers=[]
```

オブジェクト指向をベースとした自然言語ステートメントの定義

- 前述の用例を完了するには read() でファイルの内容を取得します。

```
I want window
The window title = "hello world"
```

- また、GUI ライブラリでウィンドウを作成するには GetWindow() と SetTitle() メソッドを更新します。

オブジェクト指向プログラミングをベースとした入れ子構造の宣言型言語の定義

- この用例は Web ライブラリからの引用です。Bootstrap ライブラリで HTML ドキュメントを生成します。この用例では、HTML コードを直接記述せずに、類似言語を作成しています (ただの用例です)。その後、宣言型言語を使用するために入れ子構造で HTML ドキュメントを生成しています。この用例での考えかたとして GetDiv() および GetH1() メソッドは { } でアクセスできるオブジェクトを返します。各オブジェクトへのアクセス後に BraceEnd() メソッドが実行されると、生成された HTML を BraceEnd() の出力表示がルートに到達するまで親オブジェクトへ送信します。

オブジェクト指向プログラミングをベースとした入れ子構造の宣言型言語の定義

```
Load "weblib.ring"
Import System.Web

Func Main

  BootStrapWebPage()
  {
        div
        {
          classname = :container
          div
          {
                classname = :jumbotron
                H1 {    text("Bootstrap Page")    }
          }
          div
          {
                classname = :row
                for x = 1 to 3
                  div
                  {
                        classname = "col-sm-4"
                        H3 { html("Welcome to the Ring programming language") }
                        P  { html("Using a scripting language is very fun!") }
                  }
                next
          }
        }
  }
```

オブジェクト指向プログラミングをベースとした入れ子構造の宣言型言語の定義

- このようなクラスで宣言型インタフェースを強化します。

```
Class Link from ObjsBase
        title  link
        Func braceend
                cOutput = nl+GetTabs() + "<a href='" +
                          Link + "'> "+ Title + " </a> " + nl

Class Div from ObjsBase
        Func braceend
                cOutput += nl+'<div'
                addattributes()
                AddStyle()
                getobjsdata()
                cOutput += nl+"</div>" + nl
                cOutput = TabMLString(cOutput)
```

スマートガベージコレクター

わずらわしいメモリ操作関連の問題からの解放:

- メモリへの不正アクセス
- メモリリーク
- 未初期化メモリへのアクセス
- ダングリングポインタ

スマートガベージコレクター

- グローバル変数は、代入ステートメントで削除するまでメモリに存在し続けます。
- 関数の処理終了後に、ローカル変数を削除します。
- プログラマは、代入ステートメントでメモリから変数を削除する時期を完全に制御できます。
- プログラマは callgc() 関数を呼び出すことで、ガベージコレクターを強制実行できます。
- 変数の参照時、参照カウントに基づいて変数を削除します。

シンタックスの柔軟性

- 言語のキーワードは変更可能
- 言語の演算子は変更可能
- 出入力での各種記法
- 制御構造での各種記法
  - BASIC 言語風の独自キーワードの使用
  - Pascal, Lua および Ruby 風の ‘end’ キーワードの使用
  - C,C++, Java および C# 言語風の括弧 { } の使用

実用例

- オンライン版 Ring の用法
- 取扱説明書
- ソースコード (GitHub)
- Ring ノートパッド
- ゲームとアプリケーション
- ウェブ開発のデモ