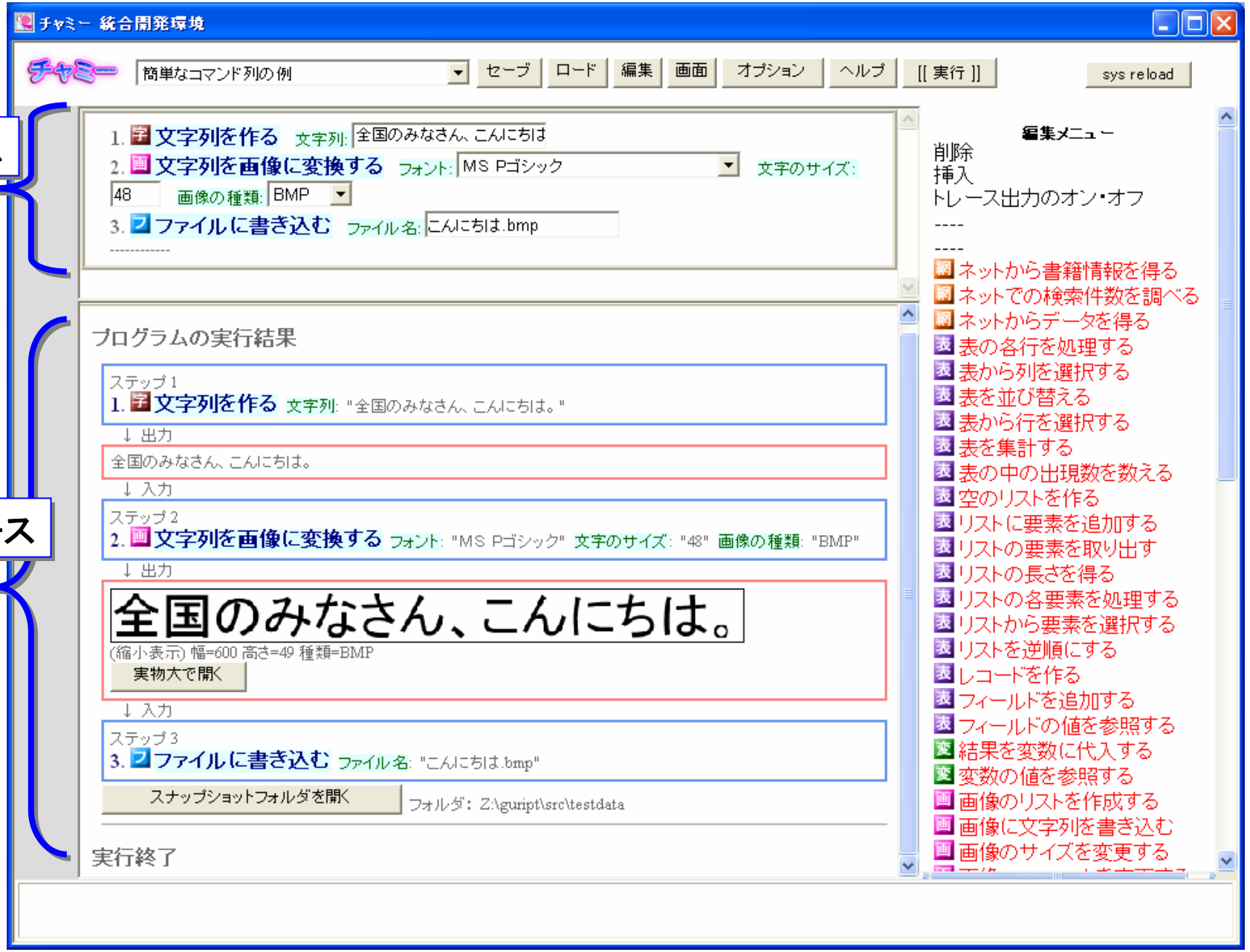
エンドユーザ向けのスクリプト言語

産業技術総合研究所
情報技術研究部門
一杉裕志、古川浩史

2005年3月10日

チャミーIDEのスクリーンショット



構造エディタによるプログラミング

- Unix のパイプのようにコマンドが順に実行される
- パラメタは GUI フォームで入力
- コマンドのネストも可能
  - 再帰的文法を持ったプログラミング言語

目的

- Unix のパイプのような処理を GUI ベースのOSでやりたい
- PC 上での日常作業の自動化
  例:
  - 古いファイルを削除したい
  - メールの文面を大量に自動生成したい
  - ファイルの名前を連番に付け替えたい

従来の方法

- アプリケーションのマクロ機能を使う
  - × アプリケーション間で連携する作業が難しい
  - × マクロで自動化しにくい作業もある

    例：古いファイルの削除
- スクリプトを書く
  - × 習得が必ずしも容易ではない

    AppleScript, VisualBasic

解決策

- エンドユーザ（プログラマではない普通のPCユーザ）でも使えるスクリプト言語

- エンドユーザは、プログラマが書いたプログラムをコピーして使うと想定
  - プログラムを読んで理解し、必要なところだけ修正

方針

- 設計目標:
  - 習得が容易
  - 間違いを起こしにくい
  - マニュアルがなくてもプログラムを理解できる
  - 実用的

チャミーの特徴的な言語機能

- IDE と相性のよい型システム
- 仮想フォルダ機構
- 実行時エラーリカバリ
- 表処理専用の制御構造
- 可視化されたトレース画面
- 表示オプション

- IDE と相性のよい型システム
- 仮想フォルダ機構
- 実行時エラーリカバリ
- 表処理専用の制御構造
- 可視化されたトレース画面
- 表示オプション

IDEと静的型

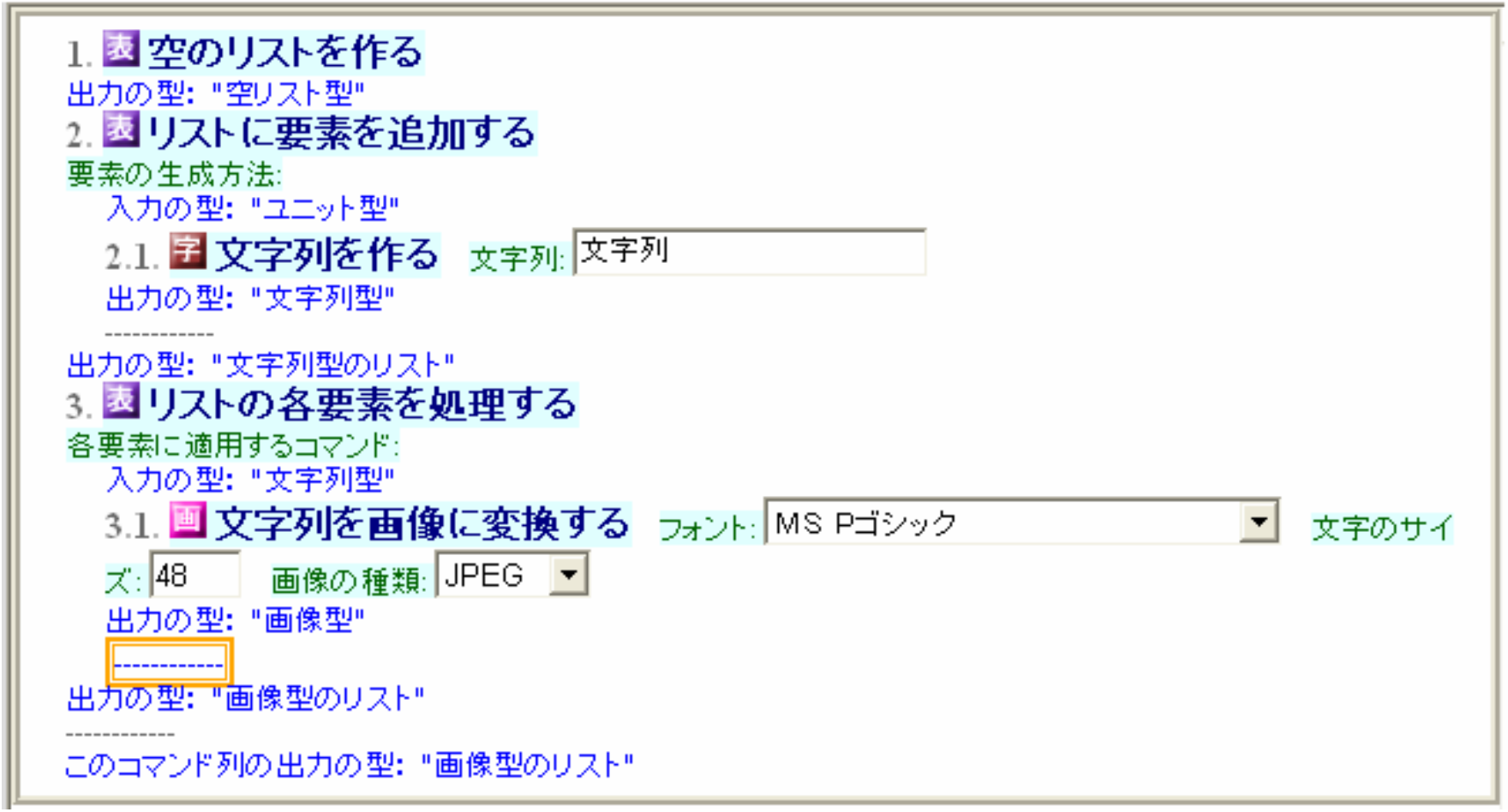
- IDE（統合開発環境）は生産性を数倍高める
  - コード補完
- コード補完が効果的に働くためには、静的型情報で候補をしぼる必要がある
- しかし、スクリプト言語では型宣言を書くのはめんどう
- 型推論はIDEのインクリメンタルな型チェックに向かない（？）

チャミーの型システム

- 特徴
  - 多相コマンドあり
  - 型宣言不要（制限された型推論）
  - IDEによるインクリメンタルな型チェック
  - （単純な言語仕様が前提）
- IDEはプログラムの先頭から順に型を確定
  - ○入力中であっても入力位置より手前は型が確定
    → 型情報を使った入力支援が可能
  - ○型エラーメッセージが分かりやすい

型チェックの例

- 入力位置より上の情報だけで型が決まる



オプションで型情報を表示させたところ

インクリメンタルな型チェック

- すべての組み込み多相コマンドは、先頭から型を確定できる形をしている
- IDEは先頭から順に型を確定
  例：map: 'a list -> ('a -> 'b) -> 'b list
  1 2 3 4
- チャミーで許されないコマンド：
  'a -> ('b -> 'a) -> 'a
  - 'b の型宣言が必要になってしまう

限界1：ファイルから読み込む場合

- ファイルタイプを指定しないと静的型が決まらない
  - 文字列、画像、 CSV のいずれかをプログラマが指定

限界2: 変数と subtyping

- チャミーの変数の型は初期値の型で決まる。
  ○ 型宣言不要
- しかし、subtype の値で初期化したいこともある。

```
a = nil;          // a:  emptyList
while (...){
  // 実は a: int list のつもりだった
  a = cons(123, a);
}
```

限界2: 変数と subtyping（つづき）

- 現在のチャミーでは以下のようにして対処。
- 変数に super type の値が代入されると、変数の型が super type に変わる。例:

```
a = nil;          // a:  emptyList
a = cons(123, a);    // a:  int list
```

  - 現在のチャミーでは emptyList が xxx list に変わるケースしかない。この場合 super type に変わっても適用可能な操作は減らないので安全。
  - （一般には super type に型を変えるのは危険。）

限界3:関数の引数

- 無名関数やユーザ定義関数の引数は、現在の方法では型を決められない
  - 将来の課題
  - 単体テストの入力を使って型を決定？
  - ○ test first を強制できて一石二鳥

例: testConcat(){
    assert "a".concat("b") == "ab";
}

- IDE と相性のよい型システム
- 仮想フォルダ機構
- 実行時エラーリカバリ
- 表処理専用の制御構造
- 可視化されたトレース画面
- 表示オプション

仮想フォルダ機構

- 目的:ファイル処理プログラムの安全な開発を支援
  - 間違って大事なファイルを壊さないように
- IDEによる開発中は、カレントフォルダを作業領域にコピーし(仮想化)、それに対してファイル処理
- 仮想化はプログラムからは完全に透過
  - 開発が完了したら仮想化をオフにして実行

- IDE と相性のよい型システム
- 仮想フォルダ機構
- 実行時エラーリカバリ
- 表処理専用の制御構造
- 可視化されたトレース画面
- 表示オプション

実行時エラーリカバリ(現在実装中)

- 実世界のデータには異常値が含まれている
- 従来のスクリプト言語:
  - 自動型変換によってエラーを回避
    - × 意図しない変換が起きてしまう
    - × 潜在的バグが見つかりにくい
- 実行時エラーが起きても、その値に明らかに依存しない実行は継続
  (make -k のようなもの)
  - ○ 実世界データのロバストな処理
  - ○ デバッグ効率の向上

実行時エラーリカバリの方針

- コマンド実行中にエラーが起きたら、エラーオブジェクトを出力する。
- エラーオブジェクトに依存しないコマンドが現れるまで、コマンドの実行をスキップする。
  - ただし、代入文がスキップされた場合は、その変数にエラーオブジェクトを入れておく。
- 実行可能なコマンドがなくなったら停止。

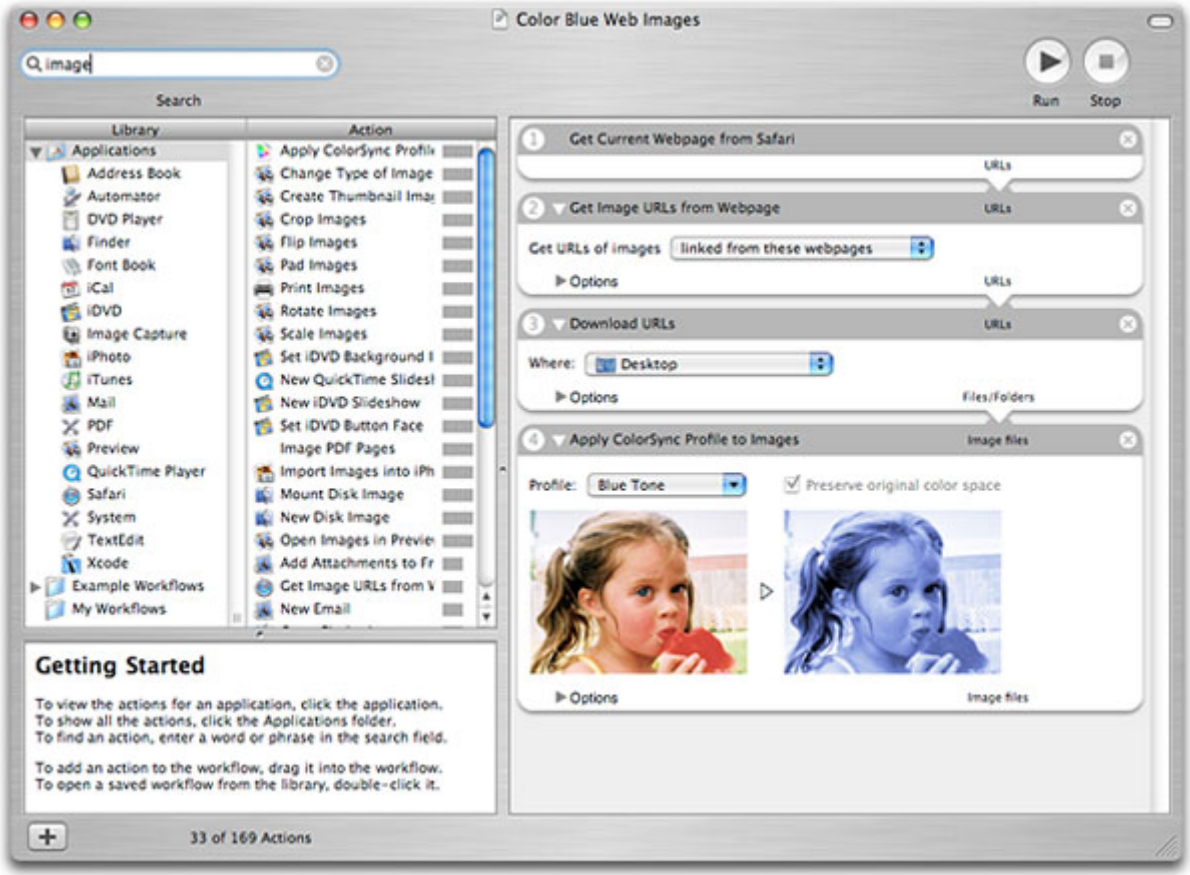
他の特徴

- IDE と相性のよい型システム
- 仮想フォルダ機構
- 実行時エラーリカバリ
- 表処理専用の制御構造
- 可視化されたトレース画面
- 表示オプション

（詳しくはデモで。）

関連研究1: Mac OS automator

- 繰り返し作業の GUI による自動化
- Unix のパイプの素直な GUI 化？



http://www.apple.com/macosx/tiger/automator.html より引用

関連研究2:SqueakToys (eToys)

- Alan Kay
- 教育用言語
  - ゲームなどが簡単に作れる
  - 構造エディタ
  - 日本語化されている
- ループが書けないなど、制約が強い

関連研究3:
Intentional Programming

- **Charles Simonyi** (Microsoft Research)
  - "The WYSIWYG"
- representation と notation の分離
- 構造エディタ
- 問題領域に適した表示
- 公開されていないため、使い勝手は不明

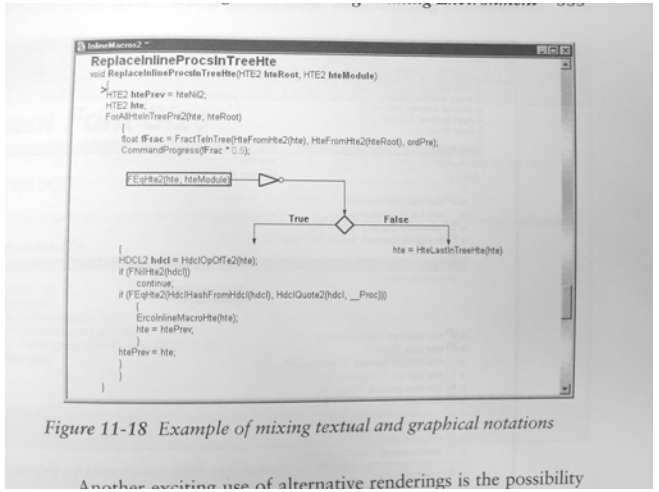


Figure 11-18 Example of mixing textual and graphical notations

Another exciting use of alternative renderings is the possibility

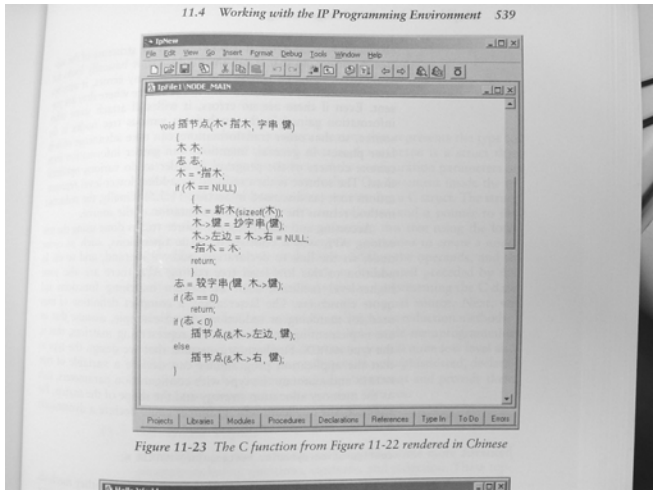
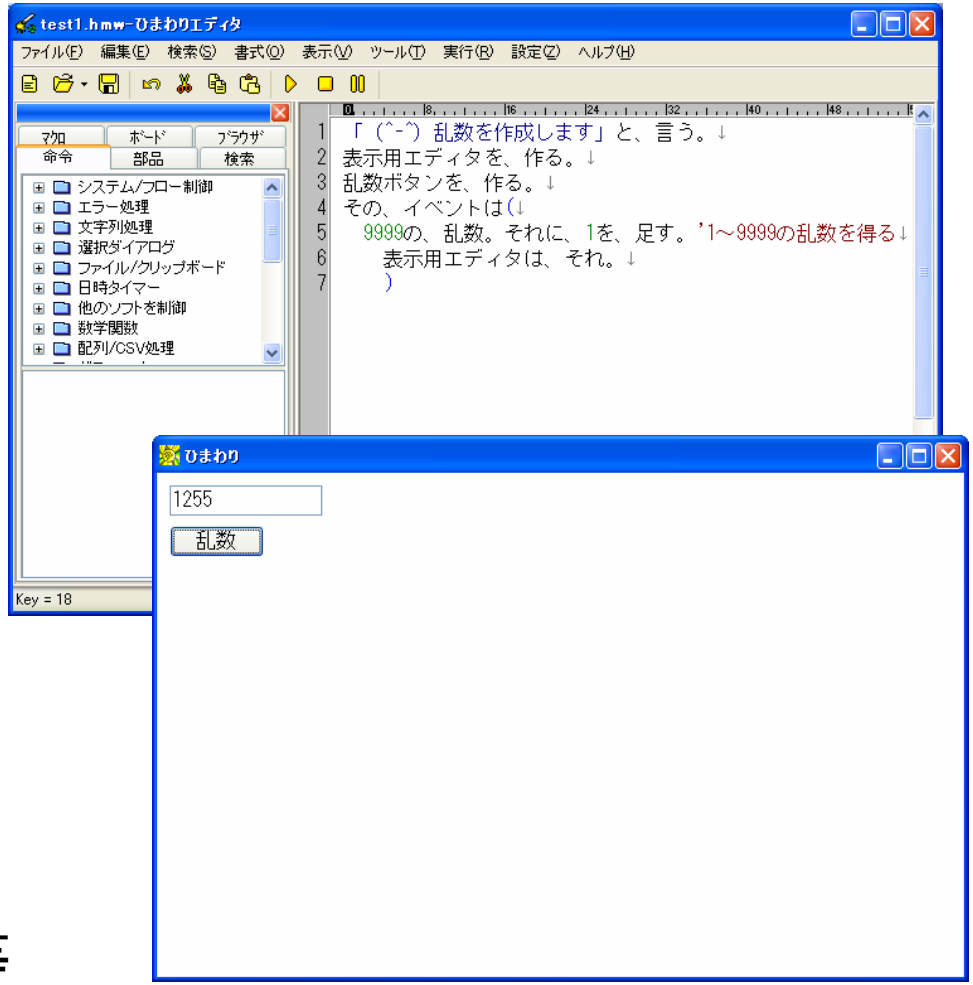


Figure 11-23 The C function from Figure 11-22 rendered in Chinese

Generative Programming,
Krysztof Czarnecki, Ulrich W. Eisenecker, Addison Wesley, p.535, p.539

関連研究4：日本語プログラミング

- ひまわり、TTSneo
  - フリーソフト
  - とっつきやすい
  - わかりやすいマニュアル
  - すぐれた開発環境
  - 実用的なライブラリ
- 他に、Mind 、言霊、旧AppleScript など
- 入力に多少難あり
  - 語順、送り仮名、句読点等に関する自明でない規則



ひまわりエディタと
プログラム実行例

実用化に向けた課題

- 実用的ライブラリの充実
  - Webサービス、表処理、XML処理、・・・
- ユーザインタフェースの向上（undo など）
- サードパーティによるコマンド作成の支援
- 複雑な式も快適に編集可能な構造エディタ
  - 四則演算など

まとめ

- エンドユーザ向けのスクリプト言語を設計・実装
  - エンドユーザに適したシンタックス
  - IDEと相性のよい静的型システム
  - 実行時エラーリカバリ、仮想フォルダ、可視化トレース
- いくつかのアイデアは、普通の言語にも応用可能

チャミーの「文法」

- プログラム　：＝　コマンド列
- コマンド列　：＝　コマンド＊
- コマンド　：＝　コマンド名 引数＊
- 引数　：＝　引数名：式
- 式　：＝　コマンド列 ｜ フォーム ｜ 計算式 ｜ 変数参照
- フォーム：＝ <セレクタ> ｜ <チェックボックス> ｜ <ファイル･フォルダ選択> ｜ <テキスト>
- 計算式　：＝　<JavaScriptの式>
- 変数参照　：＝　<セレクタ>

チャミーのシンタックスの利点

- コマンド列 ＝ オブジェクト指向的シンタックス

  (length (cdr (car list))) → list.car().cdr().length()

  ○ コントロールフローとデータフローの一致

  ○ ターゲットの型でコード補完候補をしぼれる

  ○ 構文のネストが深くならない

- ネストしたコマンド

  ＝ call by name

  ＝ ruby のブロック引数のようなもの

  ○ ruby のように抽象度の高い制御構造が提供可能

チャミーの名前の由来

- Chummy = なかよし
- 「**な**にも**か**も**よ**くつながる**し**すてむ」の略
- 計算機とユーザがなかよしに
- いろんなアプリケーションどうしがなかよしに