言語処理系分科会
第 3 回 - 変数，型，構文

semiexp

今回の内容

- 型の多様性
- 変数
- 構文

型の多様性

- まだ，1 つの型 (実数) しか扱えていない
- 整数，実数，文字列くらいは扱えるとうれしそう
- とりあえずは動的型付け言語の話をします

- もし，型の種類がもっと増えて構造体なんかが出てきたりするとさらに面倒になる
- 動的型付けの言語では，原理的に「式自体の型」が存在し得ないので，AST は型というものを完全に無視していてよい
  - 静的型付けになると，型が分からないとまともにコンパイルもできない

実装

- 構文解析は，特に何も変える必要がない
- 式 expr の評価のところを変える
- 変えるといっても，戻り値の型を double 固定ではなく，独自に用意した Value 型に変えるだけ
- Value 型は，Expr などと同様に，union を使うか，基本 Value クラスの継承を使うことにより実現できる
- ただし，+ など実際に計算するのが少し面倒になる
  - 整数 + 整数，整数 + 実数，実数 + 実数，整数 + 文字列，…
    - 実は，型なし言語では整数 / 整数を整数で計算するべきではないという説も…

変数

- Expr などを評価するときに，環境 environment を渡す
- 「環境」に，変数を定義したり呼び出したりする機能を任せる
- まだ関数は出てこないが，関数が出てくるとローカル変数環境が必要になったりする
  - crowbar の実装では，グローバル変数はインタプリタ自身に，ローカル変数はローカル環境に保存している
- とりあえず動かすだけなら，変数の参照は「変数名 -> 変数の中身」への map で十分そう
- C みたいに，スコープが複雑（関数内で新しいスコープが生えたりする）だと面倒
  - とりあえず，そういう難しいことは考えない

メモリ管理

- 今まで「メモリ管理」は大分ごまかしてきた
  - 構文木のノードはいちいち new で生成していた
  - いつ release するんだろう…
- データ型が数値だけのうちは，データ型についてのメモリ管理はいい加減でも困らなかった
  - Value が定数サイズで表現しきれる
  - スタック渡しで十分
- そろそろまじめにメモリを管理したほうがよくなってくる
  - メモリ管理モジュールも必要？

メモリ管理

- 構文木ノードについて
  - 構文木を動的に作ったり消したりということは eval しない限りない
  - 構文木ノードを取ってくるための領域を interpreter あたりにプールしておく
  - ノードが欲しくなったら，new する代わりにプールから placement new する
  - 足りなくなったら，プールを拡張する
    - プールの構成要素たちを vector に放り込むか，連結リストとして管理する
  - interpreter が要らなくなったら，プールもまとめて破棄する

garbage collector

- 文字列が出てくると大変
  - 文字列の長さは不定 (無限に長くなりうる)
- 不要になった文字列は release しないとメモリリークする
- メモリリーク回避のため，garbage collection が必要

- 文字列だけだったら，参照カウント方式で十分

参照カウンタ

- オブジェクトが何箇所から参照されているかを覚えておく
- もはや参照されなくなったらオブジェクトを破棄

- オブジェクト同士が参照を始めると，循環参照が起きてメモリリークし放題になるという欠点がある
  - Mark-sweep garbage collector (今回は説明しません)

制御構文 Stmt

- If, For, While など
- Expr と異なり，普通値を返さない (もちろん返してもよいです)
- Clang では，Expr は Stmt の一種 (Stmt を継承) という扱いになっている
  - crowbar でも，同じような扱いになっている
  - Expr の実行と Stmt の実行を同じ関数にすると，値の取り扱いが少し面倒？
  - 値付き実行，値なし実行の 2 つの関数を用意するのがよいかもしれない

Stmt の実装 (1)

- if, else など
  - else の中身も IfStmt に持たせたほうがよい
- Expr とだいたい同じ
- まず基本 Stmt クラスを実装し，if などはそれを継承する
- それぞれの特殊 Stmt クラスは，仮想関数の形で実行関数を持つ

- if だったら，単に
  - 条件節を評価して，
  - 真だったら if の中身を，偽だったら else の中身 (あれば) を実行するだけ

Stmt の実装 (2)

- for, while など
- 繰り返し文も，「それだけだったら」if とかとほとんど変わらない
  - if と同様に適切にシミュレートするだけ
- break が出てくると厄介

Stmt の実装 (3)

- break の対処法
  1. break が出てきたら，「break 中」という情報を Stmt 評価の戻り値に渡して，ループが捉えるまで戻る
     - Stmt の戻り値に状態を覚えさせるだけで済む
  2. setjmp(), longjmp() を使う
     - 関数の壁を超えた goto みたいなもので，かなりの荒業
     - Ruby の処理系では使いまくっているらしい
  3. 処理系で用いるスタックを自前で作り，そのスタック上でループの位置まで戻る
     - スタックオーバーフローの心配もなくなる
     - かなり面倒

特殊な Stmt

- 関数や if 節の中身などで，Stmt のリストというのがほしくなることがある
- C/C++ だと，{ } で囲むことで自由に「Stmt のリスト」みたいなものが作れる
  - Clang だと CompoundStmt というものがある
- これがあると，if 節の中身などでいちいちリストを作る必要がなくなる
- この「Stmt のリスト」を実行するのは，本当にリストの中身を順番に実行するだけでよい

練習問題

1. yacc の出力を C++ にしてみよう (C で書く人はこの問題は無視してください)
2. 前回の「電卓」の文法構造に対する AST を設計しよう
   - できれば，プログラミング言語の AST に拡張しやすいように
3. 「電卓」のパーサを，計算しきった結果ではなく AST を返すようにしよう
4. その返された AST を受け取って，計算結果を返す関数を書こう
5. (参考) この AST に対して，pretty printer を作成しよう
   - AST に対する pretty printer は，AST を受け取って，その AST の構文的意味を読みやすい形にして（たとえば，ソースコード状にする）出力するもの
   - これがあると，AST の内容を確認できるようになってデバッグがしやすくなる