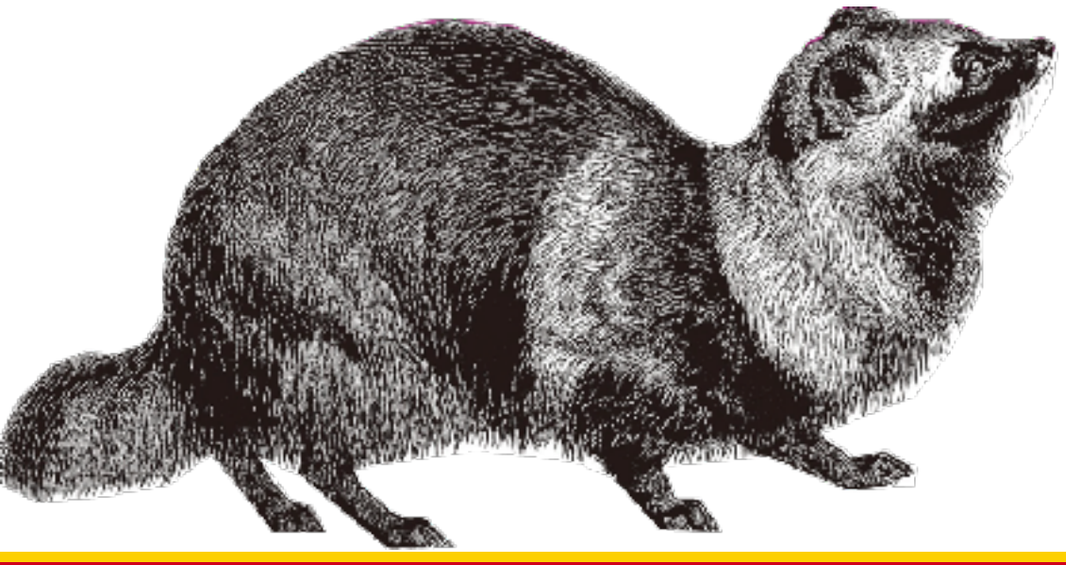
コンピュータシステムの理論と実装 11章

HAISYS B4 新人 massaman



JIN NAKAZAWA LABORATORY

0. アジェンダ

1. 導入
2. データ変換
3. コマンド変換
4. 仕様
5. コードレビュー

1. 導入

コンパイラは普通の存在になりつつある

レシピ

カレーの完成！

プログラムファイル

実行結果

青い矢印…
翻訳する役割 = コンパイラ

高級言語 ＝＞ 機械語への変換をしてくれる

これはすごいこと！！

まるで魔法のように！！！

JIN NAKAZAWA LABORATORY

1. 導入

本章では理解のためにコンパイラ解説をする (Jack を用いた開発)

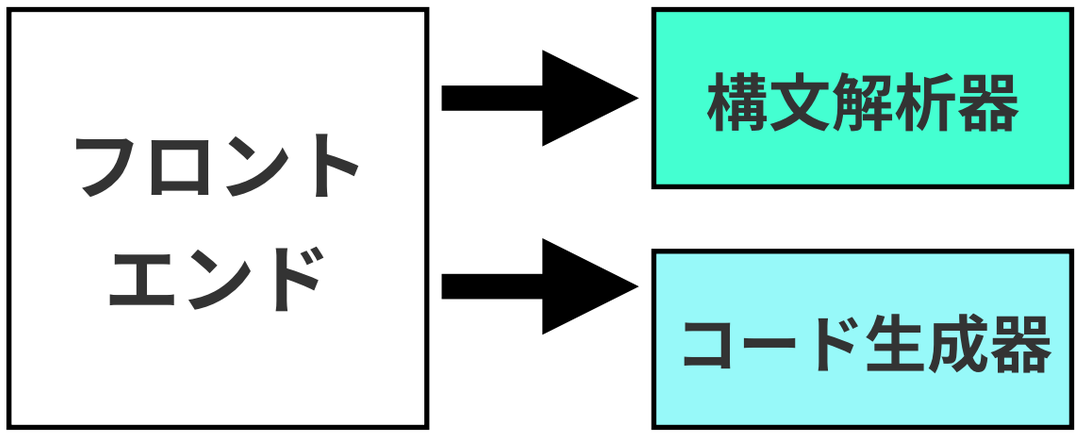
Jack コンパイラ

フロントエンド, バックエンドの2種類
本章で扱うのはフロント
バックは第7,8章で解説されたVMの話

覚えてない人は復讐習!!

フロントエンドは2つに分類される※

※モジュールとしては一括り



高級言語とVMの**橋渡し**をする！！

2. データ変換

プログラムを書く際には型がある

sample.c

```c
char str; //文字列型
int x; //整数型
float y; //浮動小数点型, doubleよりもバイトが少ない

#include <stdio.h>

int main(void) {
    return 0;
}
//関数宣言の例
```

c言語: 型宣言をする

sample.js

```javascript
var x = 0;//jsに文字列, 整数などの種類別
var arr = new Array();//配列宣言
//上記2つはグローバル変数
function myfunc(x,arr){
  var y = 1;//ローカル変数
  return 0
}
 //関数宣言例
```

JavaScript: var で宣言

sample.py

```python
def myfunc():
    x = 0;
    str = "massaman"
    list = []
    return 0

if __name__ == '__main__':
    myfunc()
```

python: 宣言必要なし

データ型の例:

単純型	整数型 (2, 100 など)	浮動小数点型 (3.14159265359, 1.0e-6など)	真偽型 (true, falseなど)	文字列型 (massaman など)
複雑型	配列型 ([1,2,3,4] な	オブジェクト型 (subject.name など)		

他にも…

- ライフサイクル
- スコープ

などの話が出てくる

このデータたちをコンパイラがどのように扱うのかを次章以降で説明

2.1 シンボルテーブル

シンボルテーブル = 識別子をまとめた場所のこと

✕

int x

bool flag

お前あっち！
あんたこっち！

毎回交通整理するのは大変…

class user()

math. floor()

○

jin 先生だから
これ！

massamanは…
いいやwww

しんくんだから
これ！

shusukeくん！
覚えないと！

全てのことを覚えてくれている

JIN NAKAZAWA LABORATORY

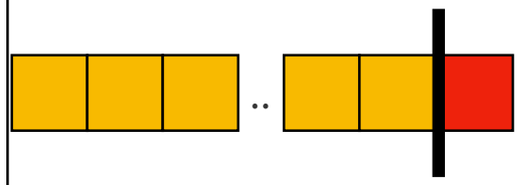
2.2 変数操作

こいつは何なんだ…?
- static で宣言されている定数？
- 関数の中で処理が終わるローカル変数？
- それともグローバルでずっと持っておく？
- 引数を受け取る用？
- インスタンスごとの生成？

などの悩みが発生する

コンパイラ

メモリをきちんと解放しないと…



オーバーフロー！！！！

X

この辺りは裏(VM)側が処理してくれる！！

（（　　））

高級言語と低級言語をVMコマンドを使って変換

2.3 配列操作

javaのサンプルコード

```
void foo (int k){
  int x, y;
  int [] bar; //配列宣言

  //配列生成
  bar = new int[10];

  bar[k] = 19;
}

Main.foo(2); //fooメソッドのコール

/*
この関数の処理であれば，10要素をもつ配列の宣言をした後に，
k番目に19を入れる処理を行う

Q: Mainで読んだメソッドでは配列に値を入れている
どうすれば19にアクセスできるだろう？
*/
```

vmコンパイルイメージ

```
push bar
push k
add
ここで,bar[k] が生成される

pop addr
push 19
pop *addr
ここでbar[k] = 19 とする宣言


push local 2
push argument 0
add

上で宣言しているkに2を入れる

pop pointer 1
push constant 19
pop that 0

bar[2] にアクセスしてそこに19をいれる
```

local変数で3種類を呼んでいる.

配列の宣言はしているが中身はまた別の連続データとしてメモリを使用する.

javaのコードでそれぞれ一行で書かれているものが複数行になっている

VM命令に変換すると上記のようになる

2.4 オブジェクト操作

java のサンプルコード

```
class Complex{
  //プロパティー
  int re; //実数部分
  int im; //虚数部分

  //複素数オブジェクトのコンストラクタ
  public Complex(int aRe, int aIm) {
    re = aRe;
    im = aIm;
  }
}

public void bla(){
  Complex a,b,c;

  a = new Complex(5,17);
  b = new Complex(12,192);

  c = a;
}
```

先にローカルの入れ物 a,b,c を用意する.

a には Complexを処理した, 5 + 17i
bには同様に, 12 + 192i が入る.

c には参照値だけがコピーされるので,
c = 5 + 17i(処理過程は関係ない)

最初にメモリを割り当てるのではなくて,
実行時にメモリを割り当てる.

(上から順番に〜の処理じゃない)

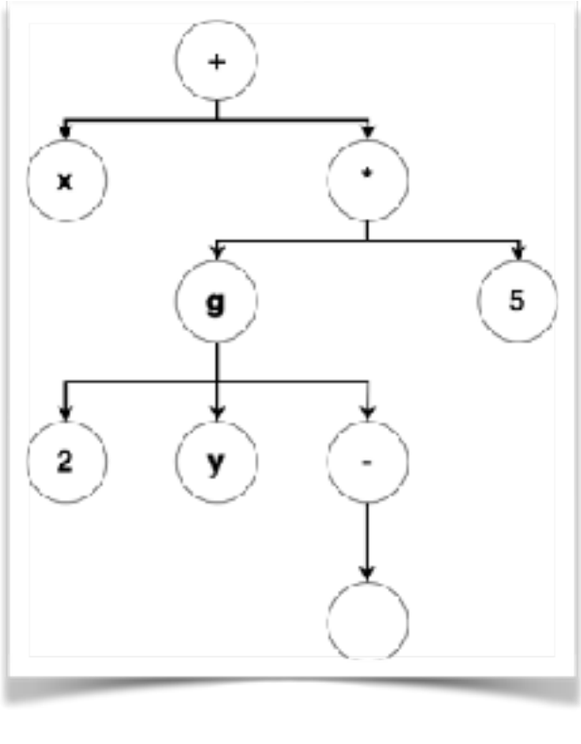
3. コマンド変換

式の評価:

ソースコード:

x + g(2,y,-z)*5

tree 構造
演算子 or 関数
でまず分類して
並列処理の変数を
考える



1.push x　6.call g
2.push 2　7.push 5
3.push y　8.call multiply
4.push z　9.add
5.neg

計算の順序に従って, 必要な順番で呼ばれる

フロー制御:

if や while などの分の処理　＝＞ 条件付き go to 文に変換される

実際にプログラムを書く際には, 条件文を書くのが複雑になりがち
しかし, VM命令に書き換える際は, 比較的単純に変換される

JIN NAKAZAWA LABORATORY

4. 仕様

jack の仕様について

- ファイル名, 関数名の定義の仕方
- メモリの使われ方
- OSの利用

コンパイラの実装へとつながっていく

5. コードレビュー

課題内容:
1. ユーザーの名前を入力させる
2. 任意の数,　数字をinputしてもらい, 配列格納する(やめるコマンドは自由, push "q" とかが一般的？)
3. inputしたデータ(名前, 配列)を表示する
4. 配列の中身をpopで取ってきて, 合計値を算出する
5. 合計値および, 名前, 数値の入っていた配列を表示する(配列は空になっている)
6. 処理にかかった時間を出力
output例:
input => massaman, 配列 = [1,2,3,4,5](配列はユーザー任意数)
output: name:massaman, sum = 15, 配列 = []; time: 100 ms

条件:
1. 複数の関数で処理をして, 引数をしようすること
ex: (pythonの例)
```
def massaman(){
  value1 = test1(x)
  test2(value) ....
}
if __name__ == '__main__':
  massaman()
```
2. str, floatなどを配列に入れると計算がおかしくなるのでエラー処理まで行う
3. デバッグの後等も全て残しておくこと
4. 必ずgithubにuploadすること(アカウントがない場合は作る)
5. 誰かにやってもらうのはダメ(難しくないため), 誰かに聞くのはOK(ただし新人同士はダメ)
github上にてコードレビューを行います(実行結果は手元でみせてください)

基本的なgithubの使い方はわかっておきましょう(RGでもやりました)
今回,できなかった人は**今日**できるようにしましょう.
逆に, **旧人になったときに使えないのはまずいです…**