為樹莓派寫個 lisp bootloader (用 c++)

descent

20160821

who is descent?

就是我

github 上的作品:

- simple_os
- simple_compiler
- simple_stdcpplib
- simple_scheme
- simple_gui

FAQ

事前準備 (1)

樹莓派2 開發版 (一定要2, 因為我沒1, 也沒有3)
uart to ttl 轉接線

安裝樹莓派 toolchain
https://github.com/raspberrypi/tools
git clone https://github.com/raspberrypi/tools.git
ls tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin
設定 PATH 指到樹莓派的 toolchain

ex:
export
PATH=/rpi2/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin:$
PATH

事前準備 (2)

bootloader source code:
https://github.com/descent/simple_scheme

編譯方法
git clone https://github.com/descent/simple_scheme.git
cd simple_scheme/
./ln.sh
make

產生 mymain.bin

事前準備 (3)

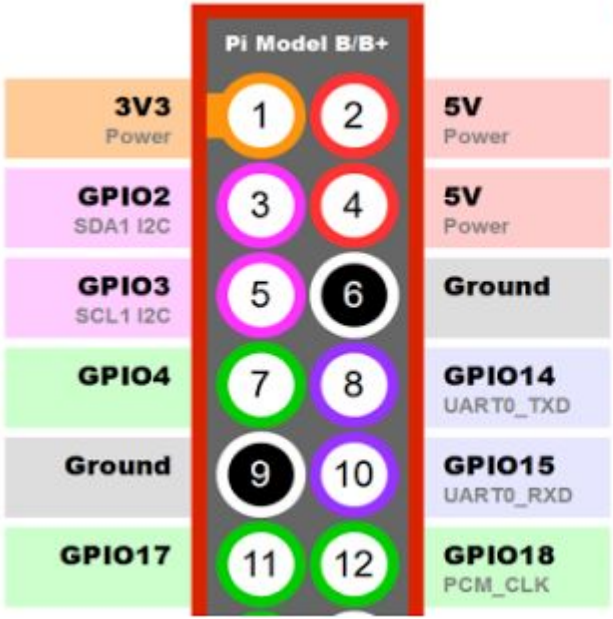
複製到 sd card

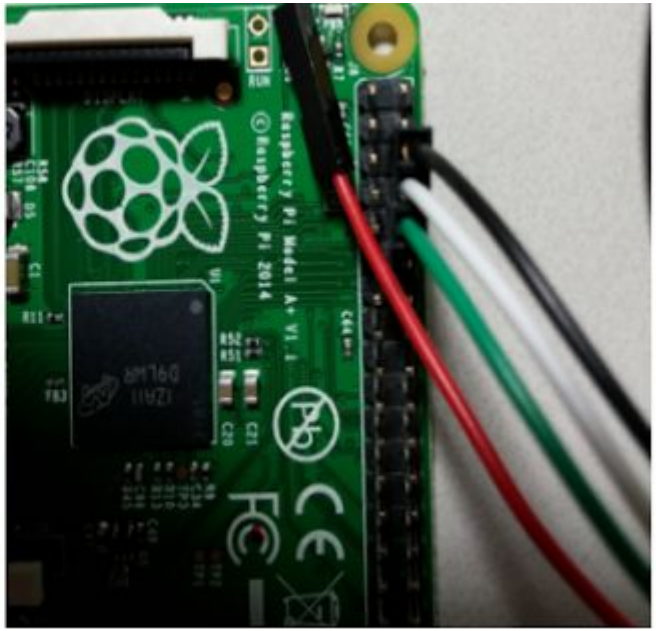
以 mymain.bin (編譯失敗的話, 可以下載這個事前編好的版本) 覆蓋 sd card 上的 kernel7.img

cp mymain.bin kernel7.img

事前準備 (4)

uart 轉 ttl 線的接法





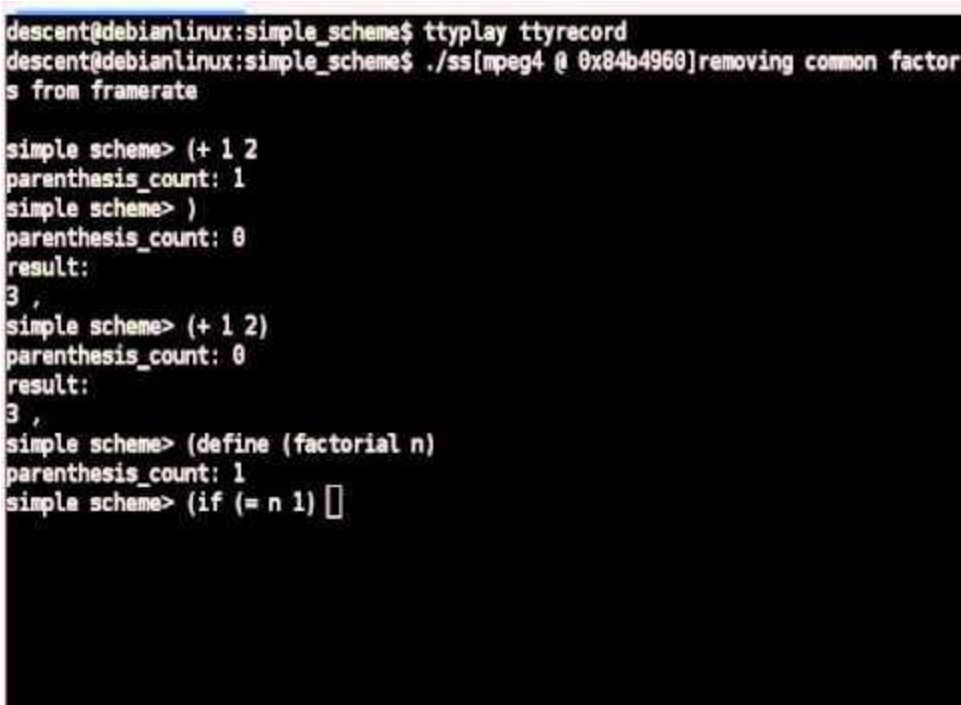
顯示畫面

minicom 設定 (需用 root 權限)

Serial port setup

```
A -    Serial Device      : /dev/ttyUSB0
B - Lockfile Location     : /var/lock
C -   Callin Program      :
D -  Callout Program      :
E -    Bps/Par/Bits       : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No
```

開啟樹梅派2後的畫面 (右側影片)



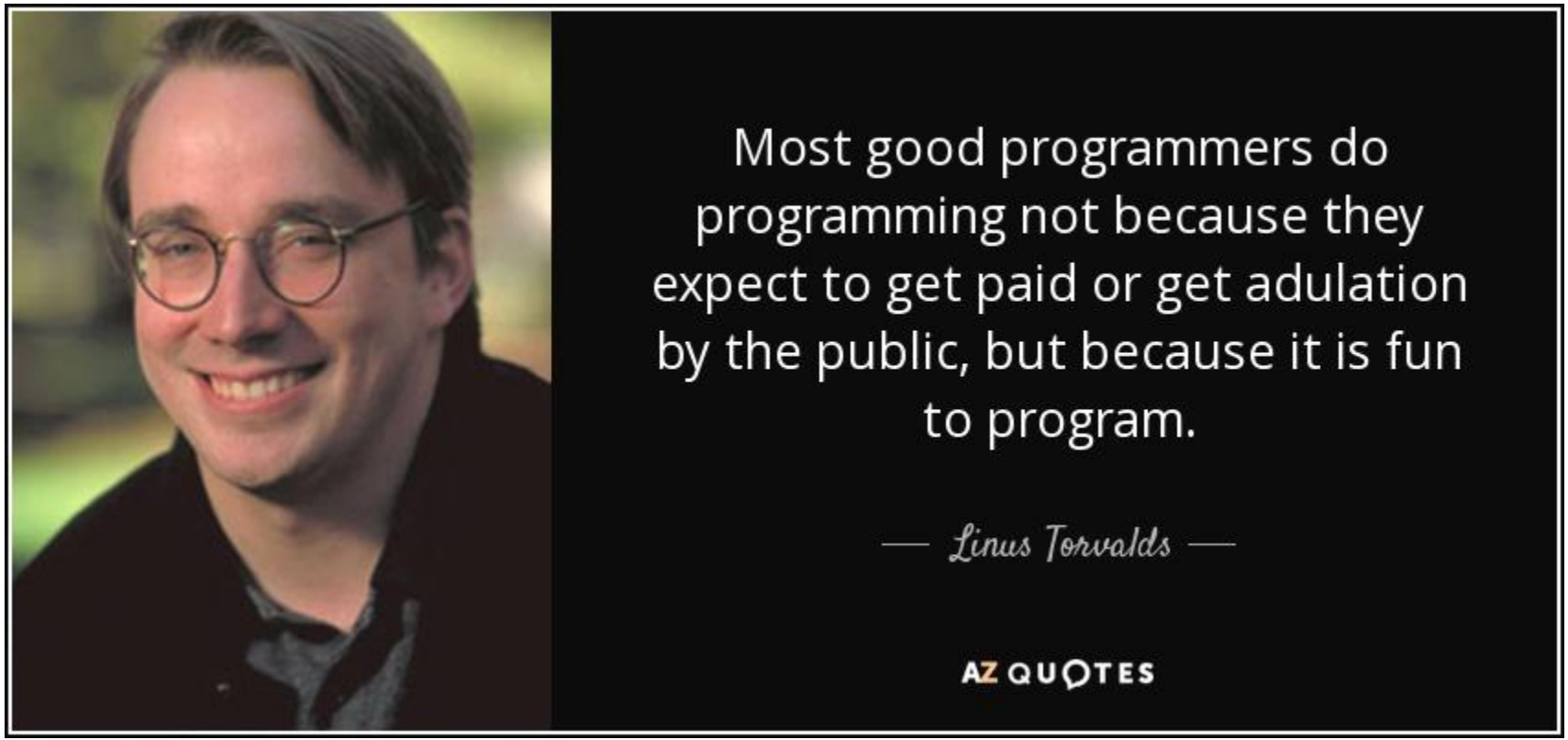
LISP X SCHEME vs FORTH

他們的關係有點類似 c, c++, Objective-C

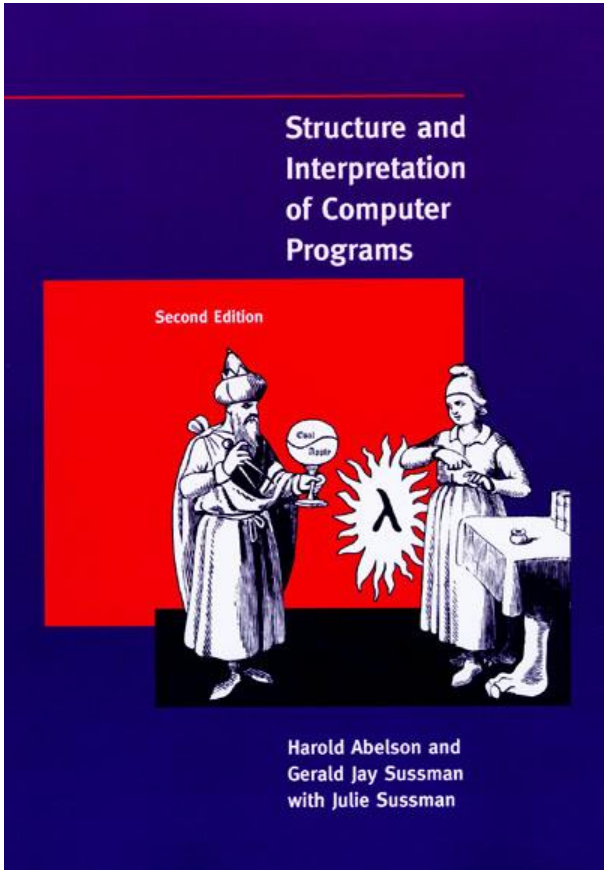
scheme 實作品:

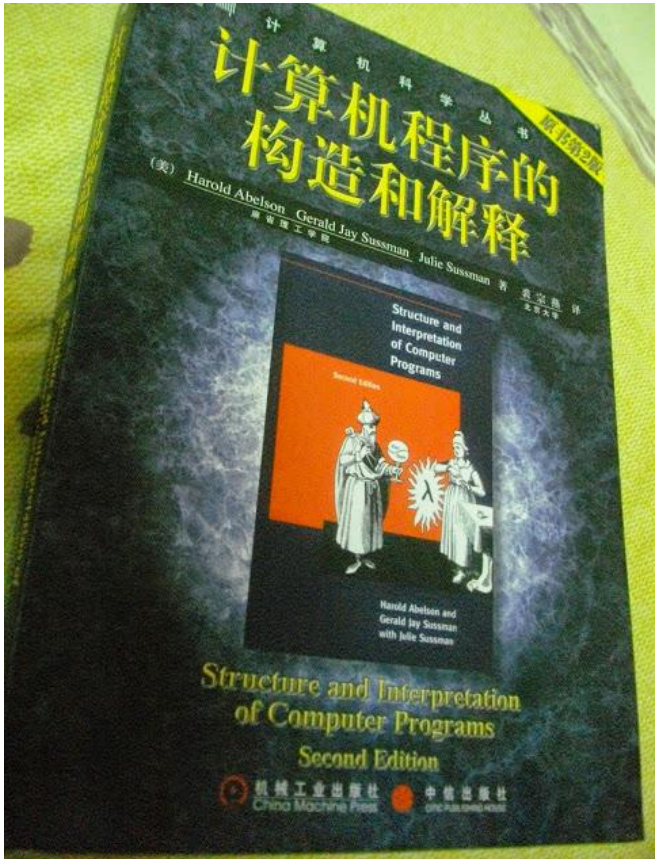
- racket
- mit-scheme
- guile

為什麼寫這個


Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.
— Linus Torvalds —
AZ QUOTES

SICP - The Metacircular Evaluator

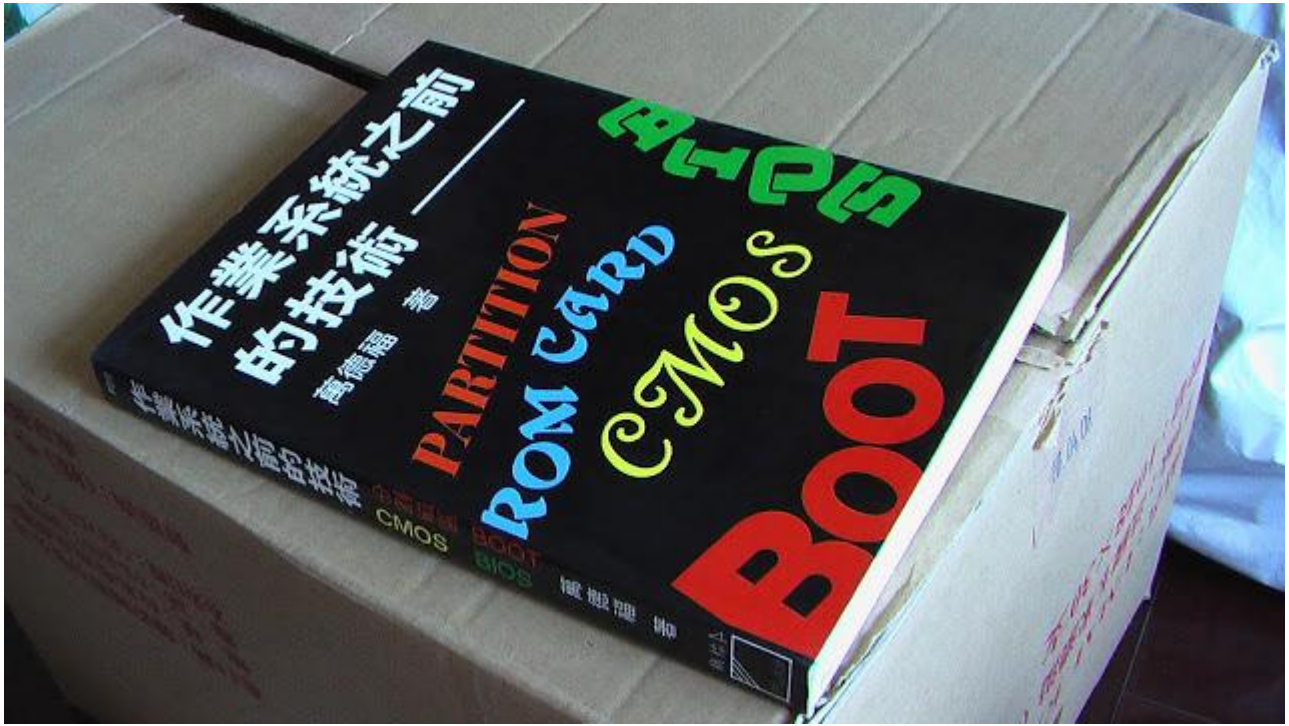




bare-metal 程式

沒有 OS 的環境

沒有標準程式庫可以用



為什麼用 c++ 寫 bare-metal 程式

因為其他的語言都不會

還用了 c++11 的特性 (auto)

用 c++ 可以寫 bare-metal 程式嗎?

可以的, c 做得到, c++ 當然也做得到。那在 bare-metal 環境上, c++ 的特性哪些無法使用呢?

很多 …

- global object
- static object
- exception handle
- rtti

bare-metal 環境上可用的 c++ 特性

- function overloaded
- template
- 繼承
- 預設函式參數
- lambda
- raw string

raspberry pi 2 Model B

rpi2 開機流程:

和一般的 arm 開發板從 arm core boot 並從 address 0 讀入第一個 arm 指令不同, rpi2 是從 gpu 開機, 也就是說某個地方放著執行 gpu 的程式碼, 它會去找 sd card 上的 bootcode.bin 然後載入並執行 bootcode.bin, bootcode.bin 再去找 sd card 上的 start.elf, 這兩個檔案都是 gpu 執行檔, 不是 arm machine code, 再來終於到 kernel.img, kernel7.img, 那個 7 就是給 rpi2 (arm cortex-A7) 用的, kernel.img 則是原來 rpi (ARM1176JZF) 用的, start.elf 會自動去判斷載入正確的 kernel*.img。

```
descent@NB-debian:boot$ file start.elf
start.elf: ELF 32-bit LSB executable, Broadcom VideoCore III, version 1 (SYSV), statically linked, stripped
```

scheme 語法

(+ 1 2 3)

(if (> 1 0) -1 1)

定義一個匿名函式

(lambda (x) (+ x 4))

執行這匿名函式

((lambda (x) (+ x 4)) 3)

定義變數

(define x 1)

(define plus4 (lambda (x) (+ x 4)))

(plus4 2)

實作 scheme 語言特性

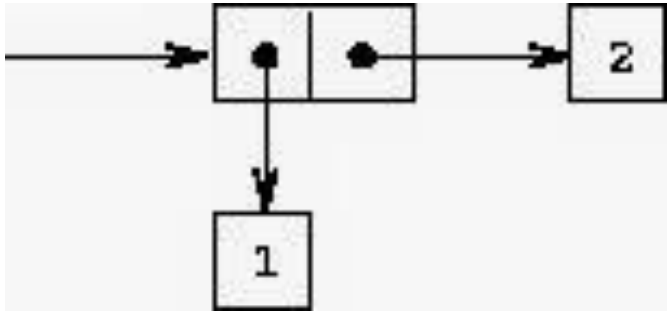
- eval expression
- if/else branch
- funcation call (lambda)
- while ??
- 定義變數

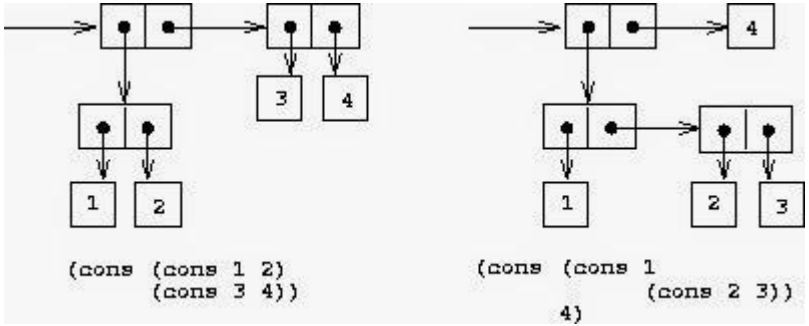
建立 AST 與符號表??

pair 資料結構

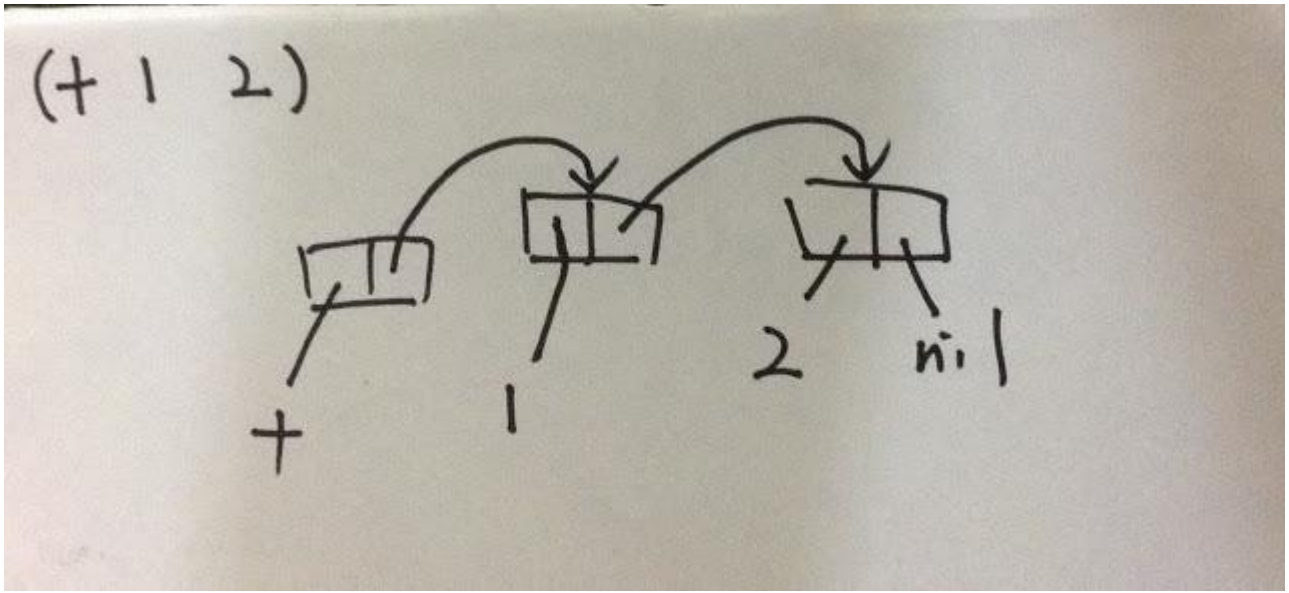
- cons
- car
- cdr

```
struct Cell
{
        CellType type_;
        PairAttr pair_attr_;
        char val_[MAX_SIZE];
        Cell *first_;
        Cell *second_;
        ProcType proc_;
        bool lambda_;
        Environment *env_;
};
```



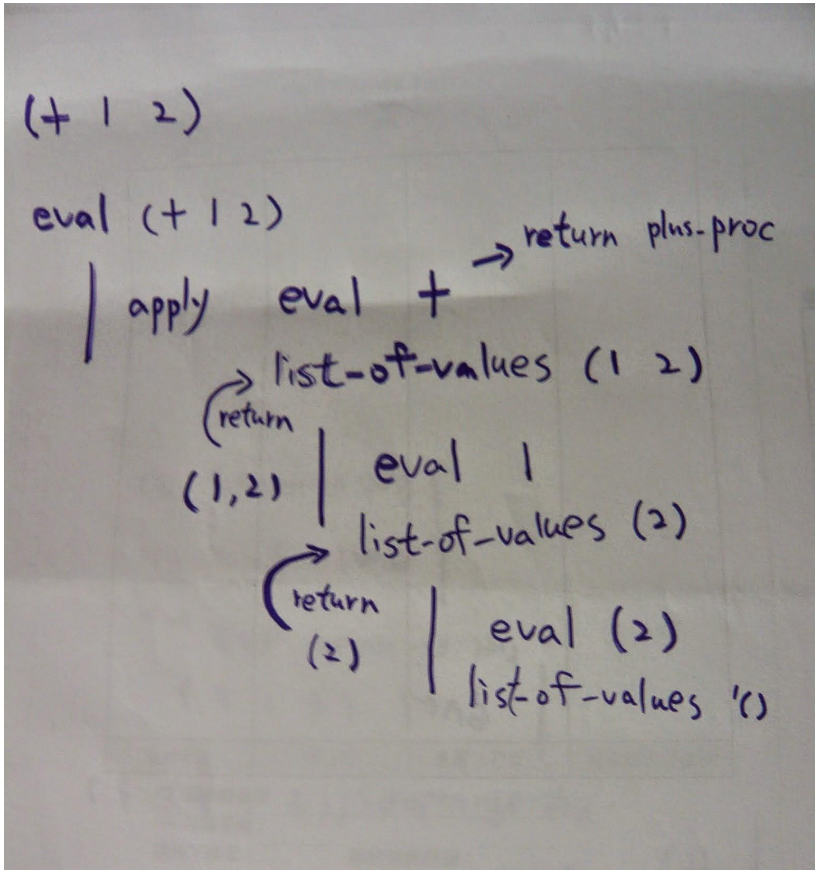


list 資料結構



eval expression

1. eval
2. list_of_values
3. apply



環境 (environment)

就是變數名稱和變數值的對應: (variable, value)

用 c++ 的 map 可表示為:

map<string, Cell*> env;

x=2;

(define x 2)

simple scheme 使用了多少記憶體

stm32f4 的版本

- stack: 44360
- bss: 122740
- 程式本身的檔案大小: 229
- 全部: 44360 + 122740 + 22948 = 190048 bytes

參考連結: 找出 stm32f4 simple scheme 記憶體使用量

寫了 c++ 標準程式庫

支援平台:

- p103 模擬器
- stm32f407
- rpi2
- x86 16 bit mode

支援的 class

- vector
- string
- map
- list

在樹莓派上的實作

- uart
- gpio (led)

```
simple scheme> (led_init)
simple scheme> (led 1)
simple scheme> (led 0)

(define (delay n) (if (= n 1) 1 (delay (- n 1)) ))
(define (led_test) (led_init) (led 1) (delay 20) (led 0) )
(led)
```

雜項

- bss 爆炸了
- stack 爆炸了
- 支援 backsapce
- line edit history
- timer