Scheme untuk Edgy

Aziz Amerul Faozi

February 28, 2020

Contents

1 Pengantar

LISP adalah bahasa pemrograman yang baik untuk edgy selain secara dialektika ini sangat mantap dan susah, karena lebih cenderung mirip sebagai bahasa computer daripada sebagai bahasa manusia

2 Installasi

2.1 Scheme

Petama kita mencoba untuk menggunakan scheme dalam part 1 kali ini untuk memulai scheme kita bisa menggunakan mode interpreter terlebih dahulu. Seperi contoh berikut ini :

```
scheme
```

maka akan muncul code seperti berikut

```
1]=> (+ 1 1)
```

code diatas merupakan contoh untuk ekseskusi interpreter dalam mode scheme. Hasilnya jelas muncul angka 2. Sama seperti halnya dengan python dimana selain memiliki interpreter scheme juga bisa dibuild dengan menggunakan editor, salah satu editor favorit saya adalah emacs. Tapi kali ini saya tidak akan fokuskan untuk penggunaan emacs, walau nanti dalam buku ini juga mungkin akan ada screenshot dari emacs. Berikut contoh code yang disimpan dalam bentuk file berekstensi scm.

```
(+ 1 1)
```

Simpan file tersebut dalam file ujischeme.scm

3 Aritmatika

Pada bagian ini saya mencoba untuk menggunakan buku yang dibuat oleh kurikum MIT untuk memberikan tutorial saja, karena contohnya saya takut akan bisa membuat saya terjerat pasal plagiasi maka yang saya buat akan sedikit berbeda dari yang para dosen MIT buat.

3.1 Aritmatika Standar

Pertama kita akan mencoba melakukan penjumlahan, pengurangan, perkalian dengan menggunakan scheme.

```
(+ 1 1); ini adalah contoh code untuk penjumlahan
```

bagian setelah tanda semicolon ";" merupakan komentar maka code tersebut hanya sebagai dokumentasi dari bahasa pemrograman sajah. Tidak akan dieksekusi oleh komputer. Untuk membuat anda lebih familiar dengan bahasa pemrograman scheme, anda bisa menebak hasil keluaran tersebut dengan menggunakan bahasa scheme

```
(/ 6 2) ; contoh dengan pembagian
(* 3 8); contoh dengan perkalian
(- 4 1); Contoh dengan pengurangan
```

Sebagai informasi untuk anda bahasa Scheme itu mirip dengan bahasa lisp maka dari itu penulisannya juga sama untuk kasus penulisan dengan negatif misalkan anda tidak bisa langsung bisa menggunakan perintah berikut

```
(-1); contoh salah mengesign angka negatif
(- 1); contoh benar
```

hal ini dikarenakan pada variable pertama pada tanda kurung digunakan sebagai operand dari program. Berikut contoh salah ketika kita menjumlahkan angka empat dengan min 1.

```
(+ 4 -1) ; ini contoh salah
(+ 4 (- 1)); ini contoh benar
(+ 4 (-1)); ini contoh benar
```

dengan code tersebut anda bisa menilai sendiri lah yah, bagaimana cara memasukkan perintah aritmatika kedalam dialektika emacs. Oke gengs biar tambah edgy mari kita coba kerjakan soal MIT berikut,

3.1.1 Latihan 1

1. Tentukan dari keluaran code berikut.

```
11
(+ 6 4 5)
(- 9 1)
(/ 6 2)
(+ (* 2 4) (- 6 7))
```

3.2 Constanta

Lalu bagaimana yah kalau mau assign constanta di scheme, jadi gampang nih gengs, semisal kita punya fungsi $E = mc^2$, lalu nilai $m = 2$ nilai $c = 3 * 10^8$ maka kamu bisa menulis seperti ini gengs.

```
(define m 2) ; memasukkan nilai m
(define c (* 3 (expt 10 8))); memasukkan nilai c
(* m (expt c 2)); mengjadikan e = mc quadrat
```

3.3 Fungsi

Seperti halnya dengan bahasa pemrograman lain bahasa pemrograman Scheme juga memiliki fungsi deklarasinya mirip seperti deklarasi konstanta tapi dia meiliki variable masukkan.

```
(define (pangkat2 x) (expt x 2)) ; deklarasi fungsi
```

dan eksekusinya tinggal panggil ajah fungsinya sebagai berikut

```
(pangkat2 9) ; pemanggilan fungsi
```

3.4 Aritmatika Anak Teknik

Oke gengs ternyata Scheme bisa support dengan komputasi anak teknik, dia ternyata mampu untuk memanipulasi varible complex. Berikut contohnya

```
(* 1+7i 1-7i) ; maka hasilnya pasti 50
```

ini berarti memang schemers bisa terbantu sekali dengan menggunakan bahasa ini, apalagi jika schemers adalah seorang anak teknik.

3.5 String

Untuk bisa lucu-lucuan harus ada string di bahasa pemrograman gengs, memang sih schemers akan cukup ribet untuk memanipulasi ini di scheme. Karena memang ini dibuat anak teknik untuk ngitung yang aneh-aneh ketimbang buat curhat makanya schemers harus mengurangi curhatnya dengan banyak berhitung. Tapi bukan berarti schemers tidak bisa menggunakan string loh gengs

```
(string "mantap")
```

untuk memanggilnya dalam bentuk variable kamu bisa menggunakan konstanta atau fungsi

```
(define teks1 (string "mantap"))
(define teks2 (string "tidak mantap"))
```

3.6 Conditional Statement

Jika maka pasti merupakan suatu hal yang wajib untuk para pemrogram (AKA programmer) pastinya sih kalau scheme udah support untuk conditional Statement yah, berikut adalah contoh penggunaan conditional statement dengan mengguanakan scheme

```
(define x 10)
(display x)
```

4 Guile

4.1 Pengantar

4.2 Instalasi

Oke gengs kali ini gue akan kombinasiin nih antara Scheme dan C. Guile sebuah taktik legendaris untuk menggabungkan Scheme-MIT dan C, biar makin edgy.

```
wget ftp://ftp.gnu.org/gnu/guile/guile-3.0.0.tar.gz
tar -zxvf guile-3.0.0.tar.gz
cd guile-3.0.0
./configure
make
sudo make install
```

4.2.1 Scheme with Guile

Menggunakan Guile, untuk menggunakan guile just type command guile after installation succed.

```
guile
scheme@(guile-user)> (+ 1 2 3); just try to execute some command
$1 = 6
scheme@(guile-user)>
```

Sekarang coba untuk melakukan eksekusi fungsi misalnya menggunakna factorial. Berikut untuk kodenya dengan

```
(define (factorial n) ; define a function
(if (zero? n) 1(* n (factorial (- n 1))))
)

(display (factorial 20))
```

simpan file tersebut dengan nama mantap.csh lalu compile dengan kode berikut

```
guile mantap.sch
```

Untuk mencoba menambahkan code anda bisa mencobanya dengan code berikut

```
(newline)
(display (getpwnam "root"))
(newline)
```

kemudian file mantap.scm kemudian file tersebut menjadi seperti berikut

```
(newline)
(define (factorial n) ; define a function
(if (zero? n) 1(* n (factorial (- n 1))))
)
(newline)
(display (getpwnam "root")) ; untuk menampilkan bash dokumen
(newline)
```

Code diatas adalah code lengkap dari kode untuk menghitung factorial dan menampilkan direktrori root.

4.2.2 Menggunakan C untuk Guile

Pertama untuk buat dengan C

```
/*
filename : simple_guile.c
*/
#include<stdlib.h>
#include<libguile.h>

static SCM my_hostname(void){
  char *s = getenv("HOSTNAME");
  if(s==NULL)
    return SCM_BOOL_F;
  else
    return scm_from_local_string(s);
}

static void inner_main(void *data, int argc, char **argv){
  scm_c_define_gsubr("my-hostname", 0, 0, 0, my_hostname);
  scm_shell(argc, argv);
}

int main(int argc, char **argv){
  scm_boot_guile(argc, argv, inner_main, 0);
  return 0; /* never reached */
}
```

set path

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

Kemudian kompile kode berikut dengan methode shared builing.

```
gcc -o simple-guile simple-guile.c\
 -I/usr/local/include/guile/3.0\
 -L/usr/local/lib -lguile-3.0
```

Kemudian set library path anda agar programnya bisa dieksekusi.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

Jika kamu ingin menjalankan program anda yang telah anda build dengan cara shared build anda bisa menggunakan perintah berikut

```
./simple-guile
```

Untuk kopmilasi dengan menggunakan pkg-config anda bisa menggunakan perintah ini, (ini sudah di test dengan arch linux)

```
gcc -o simple-guile3 simple-guile.c\
$(pkg-config --cflags --libs guile-3.0)
```

4.2.3 Build to shared library

Guile sebenarnya memiliki shared link library. Tapi kali ini gue akan mecoba utnuk membuat ekstensi dengan memnggunakan guile. Sebenarnya itu object library biasa.

```
/*
  Author : Aziz Faozi
  Description : this code will create extension in shared library.
  filename : bessel.c
*/

#include<math.h>
#include<libguile.h>

SCM j0_wrapper(SCM x){
  return scm_from_double(j0 (scm_to_double (x)));
}

SCM init_bessel(){
  scm_c_define_gsubr("j0", 1, 0, 0, j0_wrapper);
}
```

untuk membuat ekstensi dari kode tersbut kamu bisa menggunakan perintah berikut

```
gcc $(pkg-config --cflags guile-3.0) -shared -o libguile-bessel.so -fPIC bessel.c
```

untuk mengetes atau melihat dependency nya kamu bisa melihat dengan perintah ldd

```
ldd ./libguile-bessel.so
```

Maka akan nampak tampilan seperti berikut

```
ldd ./libguile-bessel.so
linux-vdso.so.1 (0x00007fff105d8000)
libpthread.so.0 => /usr/lib/libpthread.so.0 (0x00007efc9cf3d000)
libc.so.6 => /usr/lib/libc.so.6 (0x00007efc9cd75000)
/usr/lib64/ld-linux-x86-64.so.2 (0x00007efc9cf95000)
```

setelah itu kita coba load extensionnya dengan menggunakan guile

```
?????  test git:(master) ????? guile
guile
GNU Guile 3.0.0
Copyright (C) 1995-2020 Free Software Foundation, Inc.

Guile comes with ABSOLUTELY NO WARRANTY; for details type ‘,show w’.
This program is free software, and you are welcome to redistribute it
under certain conditions; type ‘,show c’ for details.
```

```
Enter ‘,help’ for help.
scheme@(guile-user)> (load-extension "./libguile-bessel" "init_bessel")
(load-extension "./libguile-bessel" "init_bessel")
scheme@(guile-user)> (j0 2)
(j0 2)
$1 = 0.22389077914123567
scheme@(guile-user)>
```

4.2.4 Menggunakan Module

Untuk melihat module di contentn ya kalian bisa melihat direktory di

```
/usr/local/share/guile/3.0
```

Disana kita bisa melihat daftar module yang bisa kita pakai dengan guile, seperti contoh ice-9 kalian akan mendapatkannya dalam list daftar berikut.

```
~ mantap => ls -la /usr/local/share/guile/3.0
total 456
drwxr-xr-x 13 root root   4096 Feb 24 22:43 .
drwxr-xr-x  3 root root   4096 Feb 24 22:43 ..
-rw-r--r--  1 root root 307025 Feb 24 22:43 guile-procedures.txt
drwxr-xr-x  3 root root   4096 Feb 24 22:43 ice-9
drwxr-xr-x 10 root root   4096 Feb 24 22:43 language
drwxr-xr-x  3 root root   4096 Feb 24 22:43 oop
drwxr-xr-x  5 root root   4096 Feb 24 22:43 rnrs
-rw-r--r--  1 root root  12708 Feb 24 22:43 rnrs.scm
drwxr-xr-x  2 root root   4096 Feb 24 22:43 scheme
drwxr-xr-x  2 root root   4096 Feb 24 22:43 scripts
drwxr-xr-x  7 root root   4096 Feb 24 22:43 srfi
-rw-r--r--  1 root root  36375 Feb 24 22:43 statprof.scm
drwxr-xr-x  4 root root   4096 Feb 24 22:43 sxml
drwxr-xr-x  5 root root   4096 Feb 24 22:43 system
drwxr-xr-x  2 root root   4096 Feb 24 22:43 texinfo
-rw-r--r--  1 root root  52321 Feb 24 22:43 texinfo.scm
drwxr-xr-x  3 root root   4096 Feb 24 22:43 web
```

Sekarang kita akan mencoba untuk memanggil model ice-9.

```
scheme@(guile-user)> (use-modules (ice-9 popen))
(use-modules (ice-9 popen))
scheme@(guile-user)> (use-modules (ice-9 rdelim))
(use-modules (ice-9 rdelim))
scheme@(guile-user)> (define p (open-input-pipe "ls -l"))
(define p (open-input-pipe "ls -l"))
scheme@(guile-user)> (read-line p)
```

```
(read-line p)
$1 = "total 500"
scheme@(guile-user)> (read-line p)
(read-line p)
$2 = "-rw-r--r-- 1 faoziaziz faoziaziz    180 Feb 25 14:50 #mantap.scm#"
scheme@(guile-user)>
```

Membuat modulmu sendiri. Pertama kita bisa membuatnya dalam direktrory module untuk itu kita bisa membuat direktory tersebut dengan perintah

```
sudo mkdir /usr/local/share/guile/3.0/foo
```

And make text file name bar.scm on this directory, which the content is

```
; This code example to create module name bar on foo directory
; filename : bar.scm
(define-module (foo bar)
  #:export (frob))
(define (frob x)(* 2 x))
```

Kamu bisa membuatnya dengan perintah berikut

```
C-x C-f
/sudo::/usr/local/share/guile/3.0/foo/bar.scm
```

Anda bisa menggunakan perintah di guile seperti berikut

```
scheme@(guile-user) [1]> (use-modules (foo bar))
(use-modules (foo bar))
;;; note: auto-compilation is enabled, set GUILE_AUTO_COMPILE=0
;;;       or pass the --no-auto-compile argument to disable.
;;; compiling /usr/local/share/guile/3.0/foo/bar.scm
;;;
scheme@(guile-user) [1]> (frob 12)
(frob 12)
$1 = 24
scheme@(guile-user) [1]>
```

kita coba masukkan code bessel tadi (yang pernah kita buat kedalam directory modulnya)

```
sudo mkdir /usr/local/share/guile/3.0/math
#+END+SRC

Filnya bernama bessel.scm dengan content seperti berikut
#+BEGIN_SRC Scheme
(define-module (math bessel)
#:export (j0))

(load-extension "libguile-bessel" "init_bessel")
```

kemudian test dengan code berikut

```
scheme@(guile-user)> (use-modules (math bessel))
(use-modules (math bessel))
;;; note: auto-compilation is enabled, set GUILE_AUTO_COMPILE=0
;;;       or pass the --no-auto-compile argument to disable.
;;; compiling /usr/local/share/guile/3.0/math/bessel.scm
;;;
scheme@(guile-user)> (j0 2)
(j0 2)
$1 = 0.22389077914123567
scheme@(guile-user)>
```

4.2.5 Menggunakan Makefile

Dengan mengunakan simple-guile.c kita bisa menggunakan Makefile untuk membuild programnya berikut makefile untuk mmebuat progammnya.

```
# Menggunakan GCC, jika kamu sudah menginstallnya
CC=gcc
# Memberi tahu c compiler untuk mencari <libguile.h>
CFLAGS=‘pkg-config --cflags guile-3.0‘

# Memberitahu kepada linker library apa yang harus dicari
LIBS=‘pkg-config --libs guile-3.0‘

simple-guile: simple-guile.o
${CC} simple-guile.o ${LIBS} -o simple-guile4
simple-guile.o: simple-guile.c
${CC} -c ${CFLAGS}  simple-guile.c
```

sayangnya harus setting linker path dulu lagi untuk bisa menjalankan programmnya.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

lalu jalankan programnya

```
./simple-guile4
```

Jika elu pengin menggunakan Autoconf untuk membangun aplikasi dengan menggunakan C/Guile anda bisa menggunakan code berikut.

```
# filename configure.ac
AC_INIT(simple-guile.c)

# find a C compiler
AC_PROG_CC
```

```
#Check for guile
PKG_CHECK_MODULES([GUILE], [guile-3.0])

# Generate a Makefile, based on the result
AC_OUTPUT(Makefile)
```

lalu buat file dengan nama Makefile.in

```
# The configure script fills in these values.
CC=@CC@
CFLAGS=@GUILE_CFLAGS@
LIBS=@GUILE_LIBS@
simple-guile: simple-guile.o
${CC} simple-guile.o ${LIBS} -o simple-guile
simple-guile.o: simple-guile.c
${CC} -c ${CFLAGS} simple-guile.c
```

kemudian jalankan

```
autoreconf -vif
```

untuk membuat file configure, lalu jalankan file configurenya dengan perintah berikut

```
./confugure
```

eksekusi tersebut akan menghasilkan file make. lalu gunakan perintah make untuk mengeksekusi perintah tersebut. Dan jalankan program.

```
make
./simple-guile
```