前回に引き続きCとの連携の話

C言語の側をいろいろいじっていきます。

前回と同様に SSK-LampControl を VisualWorksWithJun の中に
SmalltalkSources の中には Smalltalk のソースがある

テスト駆動開発をやってみよう
CppUTest がテスト用のライブラリ

LampController ディレクトリは git のリポジトリになっている

```
LampController$ ls -la
total 24
drwxrwxrwx   8 fumiya  staff   272 12  5 18:46 .
drwxrwxrwx   6 fumiya  staff   204 12  5 18:47 ..
-rwxrwxrwx@  1 fumiya  staff  6148 12  4 22:35 .DS_Store
drwxrwxrwx  15 fumiya  staff   510 12  5 18:46 .git
-rwxrwxrwx   1 fumiya  staff  1631 11  4 22:03 Makefile
drwxrwxrwx   3 fumiya  staff   102 12  5 18:46 include
drwxrwxrwx   7 fumiya  staff   238 12  5 18:46 src
drwxrwxrwx   5 fumiya  staff   170 12  5 18:46 test
LampController$
```

master branch から始めていく。
version2 の方は最後までやったもの
(時間が足りなかったときはこっちを参照しながら確認を…)

```
LampController$ git branch
* master
  version2
LampController$
```

```
LampController$ git tag
1.0
2.0a1
LampController$
```

File Browser から SSK-LampControl.st を File in

Browser から SSK-LampControl, LampControlSimulator, Class, interface specs, WindowSpec2 を Open
前回の物と比べウィンドウが 2 つになっている

make することでライブラリが出来る

```
LampController$ ls
Makefile    include             src             test
LampController$ make
mkdir -p Release/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch i386 -c -o Release/src/Control.o src/
Control.c
mkdir -p Release/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch i386 -c -o Release/src/IO.o src/IO.c
mkdir -p Release/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch i386 -c -o Release/src/
LampController.o src/LampController.c
cc -dynamiclib -flat_namespace -undefined suppress -arch i386 Release/src/Control.o Release/src/
```

IO.o Release/src/LampController.o -o Release/libLampController.dylib
LampController$

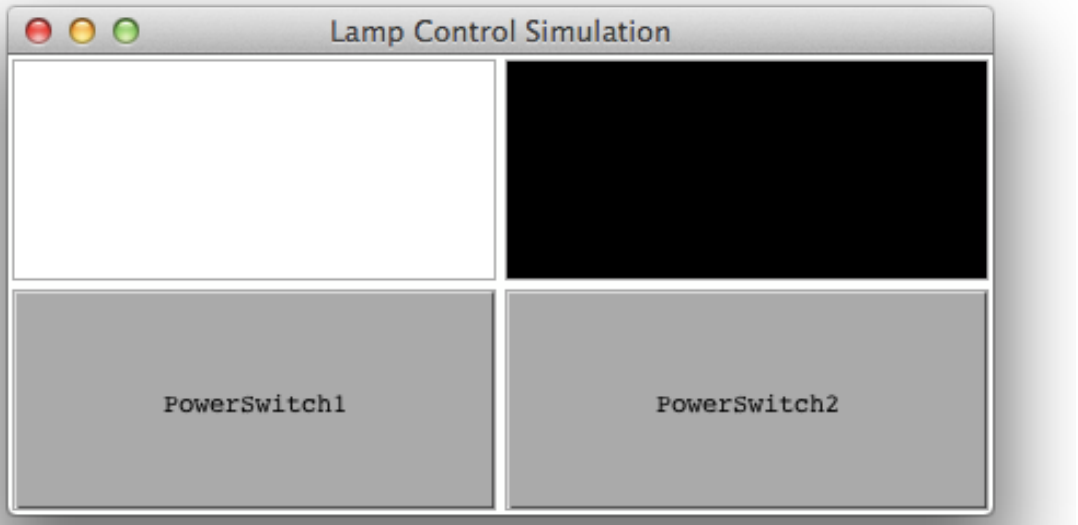
LampControllerByC_2.st と LampControllerCInterface_2.st を File in

LampControllerCInterface の中身を見ると、 C の方にいろいろ追加されていることが分かる

ここで Workspace.st を Do it する

```
| anApplication |
anApplication := SSK.LampControlSimulator new.
anApplication lampController: SSK.LampControllerByC new.
anApplication openInterface: #windowSpec2.
```

こんな感じのウィンドウが開く。
左の方は操作できるが、右の方は操作ができない。



次から C ライブラリを書き換えていく。
がライブラリを再ロードすることが難しいので、 Visual Works を終了させておくことにする。
ただし、イメージは保存しておくこと。

Control.c を見ると、現時点では左側(1のスイッチ)しか処理されていないことが分かる

```
#include "IO.h"
#include "Control.h"

static int previousPowerSwitchState = 0;

void Control_initialize(void)
{
    previousPowerSwitchState = IO_isPowerSwitch1On();
}

void Control_control(void)
{
    int presentPowerSwitchState = IO_isPowerSwitch1On();
```

```
    if (!previousPowerSwitchState && presentPowerSwitchState) {
        if (IO_isLamp1On()) IO_lamp1Off();
        else IO_lamp1On();
    }

    previousPowerSwitchState = presentPowerSwitchState;
}

/***** End of File *****/
```

そのまま、まるっとコピーして 2 を作っても良いが、面白くないのでオブジェクト指向にリファクタリングする

まずはスイッチとランプを分離する

テスト駆動でやっていくので、最初はテストコードから書くことにする

test/LampTest1s.cpp を作成
IOTests.cpp の中身をコピーしてペーストしておく

```
#include <CppUTest/TestHarness.h>

extern "C" {
#include "IO.h"
};

TEST_GROUP(Lamp1Test)
{
    void setup()
    {
        IO_initialize();
    }

    void teardown()
    {

    }
};

TEST(Lamp1Test, firstTest)
{
    FAIL("firstTest");
}

/***** End of File *****/
```

次に make test と入力して、とりあえず、テストが正しく動いているか確認する
(このテストは必ず失敗するようにした)

```
LampController$ make test
mkdir -p Debug/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/src/Control.o src/Control.c
mkdir -p Debug/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/src/IO.o src/IO.c
mkdir -p Debug/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/src/LampController.o src/LampController.c
mkdir -p Debug/test/
c++  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/test/AllTests.o test/AllTests.cpp
```

```
mkdir -p Debug/test/
c++  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/test/
ControlTests.o test/ControlTests.cpp
mkdir -p Debug/test/
c++  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/test/IOTests.o
test/IOTests.cpp
mkdir -p Debug/test/
c++  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/test/
LampTests1.o test/LampTests1.cpp
cc -L ../CppUTest-v3.3/lib -arch x86_64 Debug/src/Control.o Debug/src/IO.o Debug/src/
LampController.o Debug/test/AllTests.o Debug/test/ControlTests.o Debug/test/IOTests.o Debug/test/
LampTests1.o  -lCppUTest -lstdc++ -o Debug/LampControllerTest
Debug/LampControllerTest

test/LampTests1.cpp:22: error: Failure in TEST(Lamp1Test, firstTest)
        firstTest

.............
Errors (1 failures, 13 tests, 13 ran, 15 checks, 0 ignored, 0 filtered out, 1 ms)

make: *** [test] Error 1
LampController$
```

FAIL は CppUTest が用意しているマクロ
赤色文字で示したところで失敗したとのこと

テストコードを書く方法だとか、 make の仕方が分かったところで時間が足りないので、 git であらかじめ準備している物を使って行くことにする

make clean しておく

```
LampController$ make clean
rm -rf Debug Release *~ src/*~ include/*~ test/*~
LampController$

LampController$ git log
commit a6c8921ee11f15a373ee7bc89c3d4b5bee3b6cfa
Author: Osamu Hamasaki <osamu.hamasaki@gmail.com>
Date:   Sun Dec 2 17:57:12 2012 +0900

    Change io name from 'powerSwitch' and 'lamp' to 'powerSwitch1' and 'lamp1'.
    Add io 'powerSwitch2' and 'lamp2'

commit 64dd0291d9162fdb57a73d99aafc32693fb0262d
Author: Osamu Hamasaki <osamu.hamasaki@gmail.com>
Date:   Sun Nov 25 12:53:58 2012 +0900

    Start new project.
LampController$
```

version2 の方に移動する

```
LampController$ git checkout version2
M       Makefile
M       include/LampController.h
M       src/Control.h
M       src/IO.c
M       src/IO.h
M       src/LampController.c
```

```
M	test/AllTests.cpp
M	test/IOTests.cpp
Switched to branch 'version2'
LampController$
```

こんな感じのエラーが出たらファイルが衝突しているのでとりあえず削除することに
(どうせ復活するので…)

```
LampController$ git checkout version2
error: Your local changes to the following files would be overwritten by checkout:
	src/Control.c
	test/ControlTests.cpp
Please, commit your changes or stash them before you can switch branches.
Aborting
LampController$
```

この様に * が version2 に移動していたらオッケー。

```
LampController$ git branch
  master
* version2
LampController$
```

git log で編集履歴を見ることが出来る

```
LampController$ git log
commit a92116f37018e21642fde9905c5f93a590e859d1
Author: Osamu Hamasaki <osamu.hamasaki@gmail.com>
…略
```

赤文字がリビジョン番号

リビジョン番号 (commit の右の番号を指定 (a92116f37018e21642fde9905c5f93a590e859d1)) すると、その状態のリビジョンへ移動する(ある程度の桁数を入れれば補完してくれる)

```
LampController$ git checkout a92116f37018e21642fde9905c5f93a590e859d1
M	Makefile
M	include/LampController.h
M	src/Control.h
M	src/IO.c
M	src/IO.h
M	src/LampController.c
M	test/AllTests.cpp
M	test/IOTests.cpp
Note: checking out 'a92116f37018e21642fde9905c5f93a590e859d1'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at a92116f... Add Lamp2, PowerSwitch2, then add Controller for Lamp2, PowerSwitch2.
LampController$
```

と、ここで、僕がチェックアウトするリビジョン番号を間違えていたので、再チェックアウト

LampController$ git checkout 434afc
M	Makefile
M	include/LampController.h
M	src/Control.h
M	src/IO.c
M	src/IO.h
M	src/LampController.c
M	test/AllTests.cpp
M	test/IOTests.cpp
Previous HEAD position was a92116f... Add Lamp2, PowerSwitch2, then add Controller for Lamp2, PowerSwitch2.
HEAD is now at 434afc8... Extract class Lamp1
LampController$

Lamp1Tests1.cpp にテストコードが増えている

```
#include <CppUTest/TestHarness.h>

extern "C" {
#include "IO.h"
#include "Lamp1.h"
};

TEST_GROUP(Lamp1Test)
{
    Lamp1 lamp1;
    Lamp1* fixture;

    void setup()
    {
        IO_initialize();
        fixture = &lamp1;
        Lamp1_initialize(fixture);
    }

    void teardown()
    {

    }
};

TEST(Lamp1Test, initialize_lamp1Off)
{
    CHECK_FALSE(IO_isLamp1On());
}

TEST(Lamp1Test, lamp1On)
{
    Lamp1_on(fixture);

    CHECK_TRUE(IO_isLamp1On());
}

TEST(Lamp1Test, lamp1Off)
{
    Lamp1_on(fixture);
    Lamp1_off(fixture);

    CHECK_FALSE(IO_isLamp1On());
}

/***** End of File *****/
```

他にも色々編集されているので確認しておくこと

で、 make test すると

```
LampController$ make test
mkdir -p Debug/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/src/Control.o src/
Control.c
mkdir -p Debug/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/src/IO.o src/IO.c
mkdir -p Debug/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/src/
LampController.o src/LampController.c
mkdir -p Debug/src/
cc  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/src/lamp1.o src/
lamp1.c
mkdir -p Debug/test/
c++  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/test/AllTests.o
test/AllTests.cpp
mkdir -p Debug/test/
c++  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/test/
ControlTests.o test/ControlTests.cpp
mkdir -p Debug/test/
c++  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/test/IOTests.o
test/IOTests.cpp
mkdir -p Debug/test/
c++  -I src -I include -I ../CppUTest-v3.3/include -MMD -MP -arch x86_64 -c -o Debug/test/
Lamp1Tests.o test/Lamp1Tests.cpp
cc -L ../CppUTest-v3.3/lib -arch x86_64 Debug/src/Control.o Debug/src/IO.o Debug/src/
LampController.o Debug/src/lamp1.o Debug/test/AllTests.o Debug/test/ControlTests.o Debug/test/
IOTests.o Debug/test/Lamp1Tests.o  -lCppUTest -lstdc++ -o Debug/LampControllerTest
Debug/LampControllerTest
.............
OK (15 tests, 15 ran, 18 checks, 0 ignored, 0 filtered out, 1 ms)

LampController$
```

ちゃんと、テストが通りました。

次はこのリビジョン

```
LampController$ git checkout 10324e79dea935058c77d0f63d9eea8d43cb89cc
M	Makefile
M	include/LampController.h
M	src/Control.h
M	src/IO.c
M	src/IO.h
M	src/LampController.c
M	test/AllTests.cpp
M	test/IOTests.cpp
Note: checking out '10324e79dea935058c77d0f63d9eea8d43cb89cc'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 10324e7... Extract class 'PowerSwitch1'
LampController$
```

git log は下の方に古いリビジョンの物を出してくれる模様。
(これが原因で間違えたんだな…)

スイッチを押したときの挙動のテスト(網羅している)

PowerSwitch1Test.cpp

```
#include <CppUTest/TestHarness.h>

extern "C" {
#include "IO.h"
#include "PowerSwitch1.h"
};

TEST_GROUP(PowerSwitch1Test)
{
    PowerSwitch1 switch1;
    PowerSwitch1* fixture;

    void setup()
    {
        IO_initialize();
        fixture = &switch1;
        PowerSwitch1_initialize(fixture);
    }

    void teardown()
    {

    }
};

TEST(PowerSwitch1Test, initialize_switch1Off)
{
    CHECK_FALSE(PowerSwitch1_isOn(fixture));
}

TEST(PowerSwitch1Test, io_On_then_switch1On)
{
    IO_powerSwitch1On();

    CHECK_TRUE(PowerSwitch1_isOn(fixture));
}

TEST(PowerSwitch1Test, initialize_tick_NotChangedToOn)
{
    PowerSwitch1_tick(fixture);

    CHECK_FALSE(PowerSwitch1_hasChangedToOn(fixture));
}

TEST(PowerSwitch1Test, initialize_ioOn_tick_ChangedToOn)
{
    IO_powerSwitch1On();
    PowerSwitch1_tick(fixture);

    CHECK_TRUE(PowerSwitch1_hasChangedToOn(fixture));
}

TEST(PowerSwitch1Test, initialize_ioOn_tick_tick_NotChangedToOn)
{
    IO_powerSwitch1On();
    PowerSwitch1_tick(fixture);
    PowerSwitch1_tick(fixture);
```

```
    CHECK_FALSE(PowerSwitch1_hasChangedToOn(fixture));
}

TEST(PowerSwitch1Test, initialize_ioOn_tick_ioOff_tick_NotChangedToOn)
{
    IO_powerSwitch1On();
    PowerSwitch1_tick(fixture);
    IO_powerSwitch1Off();
    PowerSwitch1_tick(fixture);

    CHECK_FALSE(PowerSwitch1_hasChangedToOn(fixture));
}

TEST(PowerSwitch1Test, initialize_ioOn_tick_ioOff_tick_tick_NotChangedToOn)
{
    IO_powerSwitch1On();
    PowerSwitch1_tick(fixture);
    IO_powerSwitch1Off();
    PowerSwitch1_tick(fixture);
    PowerSwitch1_tick(fixture);

    CHECK_FALSE(PowerSwitch1_hasChangedToOn(fixture));
}

/***** End of File *****/
```

この間にソースコードの説明があったのだが、とてもログを取るのが大変なので…

LampController$ make test
…略

.....
OK (22 tests, 22 ran, 25 checks, 0 ignored, 0 filtered out, 1 ms)

LampController$

だんだん、テストの項目が増えている

次はこれ
LampController$ git checkout b209b110

次はスーパークラスを作る
(機能的な追加はないのでテスト項目は増えない)
Lamp.c と Lamp.h

Lamp.c

```
#ifndef _LAMP_H_
#define _LAMP_H_

typedef struct _Lamp Lamp;

typedef struct _LampVFuncs LampVFuncs;
struct _LampVFuncs
{
    void (*ioOn)(Lamp*);
    void (*ioOff)(Lamp*);
};

struct _Lamp
{
```

```
    LampVFuncs* vfuncs;
};

extern void Lamp_on(Lamp* self);
extern void Lamp_off(Lamp* self);

#define Lamp_ioOn(self) \ // <-- 邪魔くさいのでマクロ化
    (*((Lamp*)(self))->vfuncs->ioOn)((Lamp*)(self))

#define Lamp_ioOff(self) \
    (*((Lamp*)(self))->vfuncs->ioOff)((Lamp*)(self))

#endif // _LAMP_H_


#include "IO.h"
#include "Lamp1.h"

static void Lamp1_ioOn(Lamp* super);
static void Lamp1_ioOff(Lamp* super);

// C99 でこの様に書けるようになった
static LampVFuncs vfuncs = { // 関数テーブル
    .ioOn = Lamp1_ioOn,
    .ioOff = Lamp1_ioOff
};

void Lamp1_initialize(Lamp1* self)
{
    self->_lamp.vfuncs = &vfuncs;
}

void Lamp1_ioOn(Lamp* super)
{
    IO_lamp1On();
}

void Lamp1_ioOff(Lamp* super)
{
    IO_lamp1Off();
}

// End of File
```

LampController$ git checkout bcb4b51d

```
#ifndef _SWITCH_H_
#define _SWITCH_H_

#include <stdbool.h>

typedef struct _Switch Switch;

typedef struct _SwitchVFuncs SwitchVFuncs;
struct _SwitchVFuncs
{
    bool (*isIoOn)(Switch*);
};

struct _Switch
{
    SwitchVFuncs* vfuncs;

    int previousState;
```

```
    bool hasChangedToOn;
};

extern void Switch_initialize(Switch* self);
extern bool Switch_isOn(Switch* self);
extern void Switch_tick(Switch* self);
extern bool Switch_hasChangedToOn(Switch* self);

#define Switch_isIoOn(self) \
    (*((Switch*)(self))->vfuncs->isIoOn)((Switch*)(self))

#endif // _SWITCH_H_
```

Control.c を見ると Lamp1 に強くヒモ付いている箇所がある

```
#include <stdbool.h>
#include "Control.h"
#include "Lamp1.h"
#include "Lamp.h"
#include "PowerSwitch1.h"
#include "Switch.h"

static Lamp1 lamp1;
static Lamp* pLamp1;

static PowerSwitch1 powerSwitch1;
static Switch* pPowerSwitch1;

static bool isLamp1On;

void Control_initialize(void)
{
    isLamp1On = false;

    Lamp1_initialize(&lamp1);
    pLamp1 = (Lamp*)&lamp1;

    PowerSwitch1_initialize(&powerSwitch1);
    pPowerSwitch1 = (Switch*)&powerSwitch1;
}

void Control_control(void)
{
    bool isChanged = false;

    Switch_tick(pPowerSwitch1);

    if (Switch_hasChangedToOn(pPowerSwitch1)) {
        isLamp1On = !isLamp1On;
        isChanged = true;
    }

    if (isChanged) {
        if (isLamp1On) {
            Lamp_on(pLamp1);
        }
        else
        {
            Lamp_off(pLamp1);
        }
    }
}

/***** End of File *****/
```

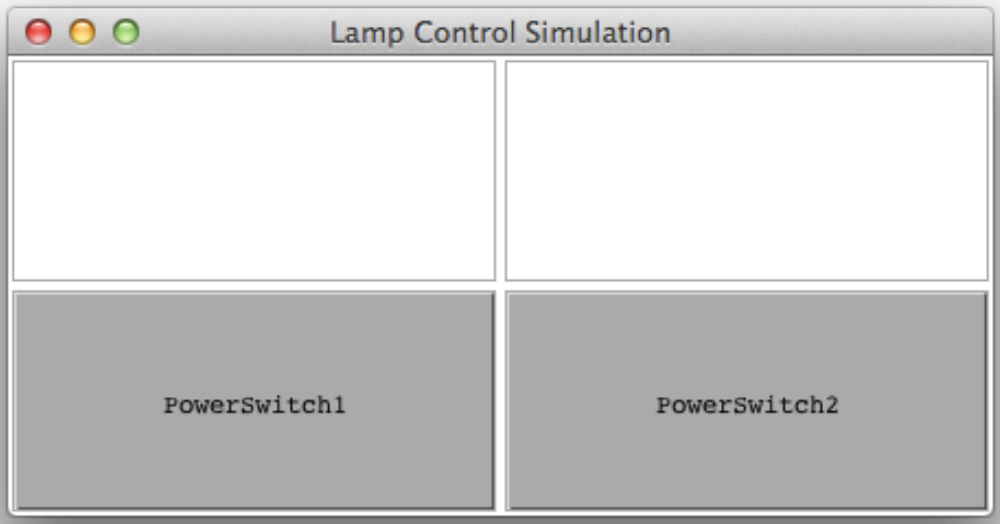
LampController$ git checkout 289b48c
親クラスに移動させた

Controller.h
```
#ifndef _CONTROLLER_H_
#define _CONTROLLER_H_

#include <stdbool.h>
#include "Switch.h"
#include "Lamp.h"

typedef struct _controller Controller;
struct _controller
{
    Lamp* lamp;
    Switch* powerSwitch;

    bool isLampOn;
};

void Controller_initialize(Controller* self, Switch* powerSwitch, Lamp* lamp);
void Controller_control(Controller* self);

#endif // _CONTROLLER_H_
```

Controller.c
```
#include "Switch.h"
#include "Lamp.h"
#include "Controller.h"

void Controller_initialize(Controller* self, Switch* powerSwitch, Lamp* lamp)
{
    self->isLampOn = false;
    self->powerSwitch = powerSwitch;
    self->lamp = lamp;
}

void Controller_control(Controller* self)
{
    bool isChanged = false;

    Switch_tick(self->powerSwitch);

    if (Switch_hasChangedToOn(self->powerSwitch)) {
        self->isLampOn = !self->isLampOn;
        isChanged = true;
    }

    if (isChanged) {
        if (self->isLampOn) {
            Lamp_on(self->lamp);
        }
        else
        {
            Lamp_off(self->lamp);
        }
    }
}

// End of File
```

ココまで来ると、 Smalltalk 側で両方ちゃんと動く様に修正できたので

改めて Workspace.st を実行すると



そんなわけで、 git の使い方及び、 C 言語でオブジェクト指向なプログラミングをする方法の紹介でした。
Makefile もかなり参考になるので、見ておくように。
上の方だけ修正すれば使い回せる