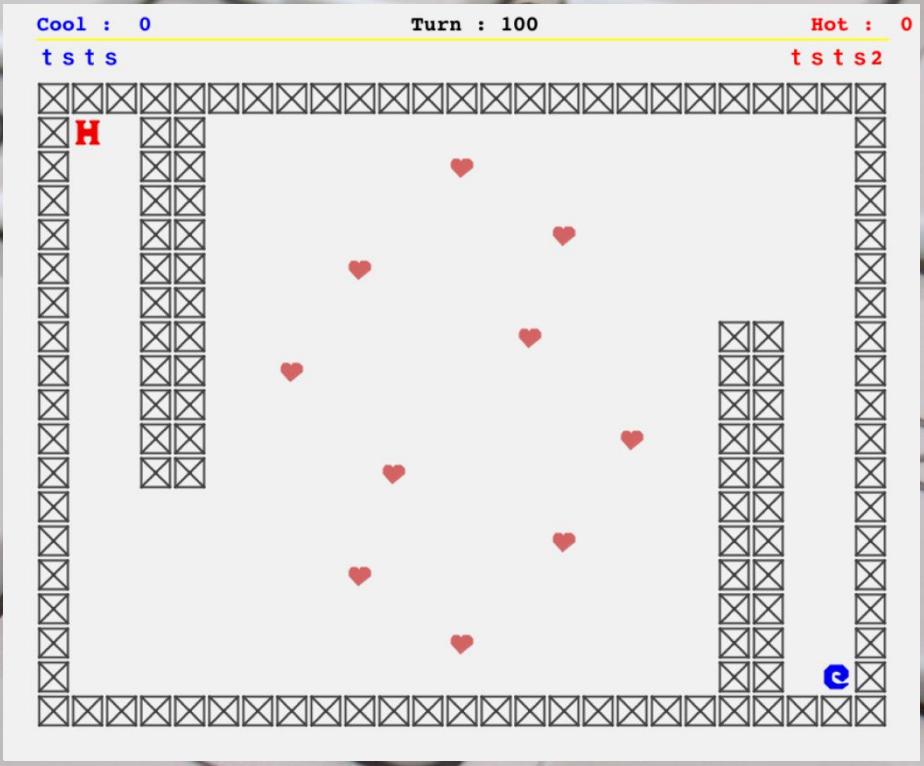
U-16 旭川プログラミングコンテスト



競技部門概要資料

旭川高専パソコン部著

目次

第 1 章

Linuxの使い方

—概要—

Linux は，windows などとは違い色々な個人が協力して趣味で作成した OS(オペレーティングシステムの略で，windows の様にパソコンを人間に取って使いやすくするためのソフトウェア) です．なので，見た目や機能などを自分に必要な状態にカスタマイズすることが出来るため，大学の研究室や企業などで広く用いられています．

今年の旭川プロコンは Linux を旭川プロコン専用 OS として改造したものを使っていきたいと思います．そのための使い方を勉強して行きましょう．

1.1 起動方法

(1) パソコンを起動する前に USB を差し込みます．

(2) BIOS(これは，windows などを呼び出すために使われるマザーボードに内蔵されているソフトウェアです) のブートマネージャを用いて変更します (**警告：これは間違えると PC が壊れる危険が有る行動なので，心配なら中学校の技術の先生や旭川プロコンで教えてくれる工業高校や高専の人たちに聞いてみて下さい**)．

(3) しばらく謎の黒い文字が出て来たあと，以下の様な画面が出てくると思います．

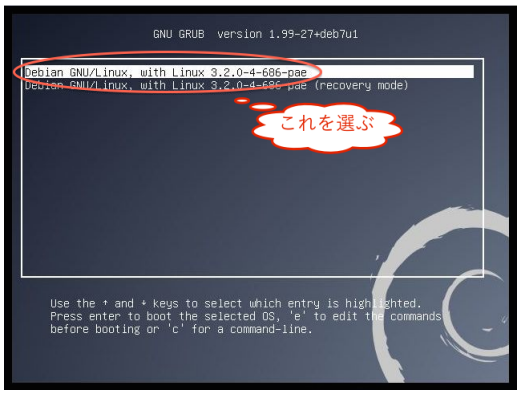


図 1.1　選択画面

これの上側を選んで下さい (下側は基本的に使いません)．

(4) 謎のハッカー的な画面の後，以下の画面になる筈です．これが，いわゆるデスクトップです．講習では，下の方にある 5 つのアイコンを主に使ってプログラムを書いていきます．

図 1.2　デスクトップ画面

また，デスクトップにおいてあるファイルは**いじらないでください！** なぜかと言うと，中には Windows のデータに関するモノもあるので，まちがって windows を壊したりする可能性があるからです．

起動方法は以上です．興味のある人は，自分で調べて他のソフトウェアなどの使い方を勉強してみるのも良いかもしれません．

1.2 終了方法

(1) 左上のアプリケーションの項目を押して，一番下にあるログアウトを押す．

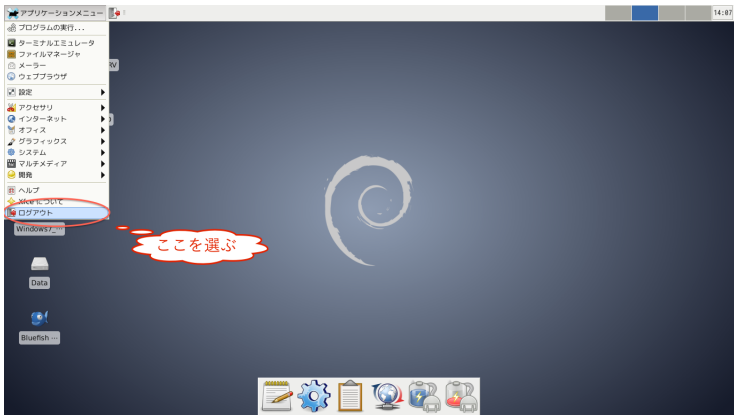


図 1.3　アプリケーション選択画面

(2) ログアウトを押すと出てくる，終了の一覧から，「電源を切る」を選択する (それ以外を選択すると，ユーザ名とパスワードの入力をしたりなど**めんどくさいことになる**).

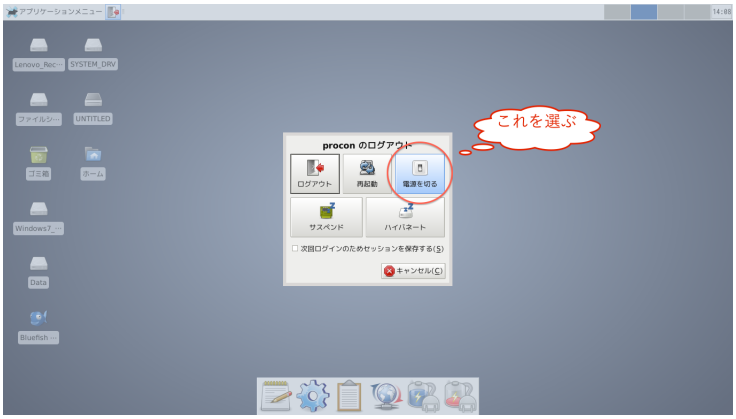


図 1.4　終了画面

第 2 章

プログラムの使用方法

では，Linux でのプログラムを書いていく方法を学んでいきましょう．

2.1 プログラム作成から実行の流れ

プログラミングは基本的に一連の流れによって，作成されます．

(1) プログラムソースを書く．
(2) コンパイルする．
(3) コンパイルが成功したら次へ行く．もし，失敗したら (1) へと戻ってやり直す．
(4) プログラムを実行する．今回の大会は以下の通りに実行する．
  i サーバプログラムを起動する．
  ii COOL(先攻側のプログラム) を起動する．
  iii HOT(後攻側のプログラム) を起動する．

この流れを普通は黒い画面に文字を打ち込んで行います．しかし，それは結構大変なので，今回の大会では**クリックするだけで実行出来る様にしました！** 各流れに対応する 6 個のアイコンを順番に押していくだけで実行できます．
その 6 個のアイコンは画面の下，以下の図の様な位置に存在します． このアイコンは，**左から順番に**押していきます．左から

図 2.1　Dock

順番に，押すとどのようになるのか，実際に図を交えながら説明したいと思います．

(1)  : このアイコンは，プログラムソースを作成するためのコマンドです．クリックすると次の画面が現れます． この画面に書かれているモノがプログラムソースと呼ばれるモノで，実際にこれを書き換えることでプログラムを作成していきます．
作成し終わったプログラムを保存するためには下に赤丸で示した場所を押します．

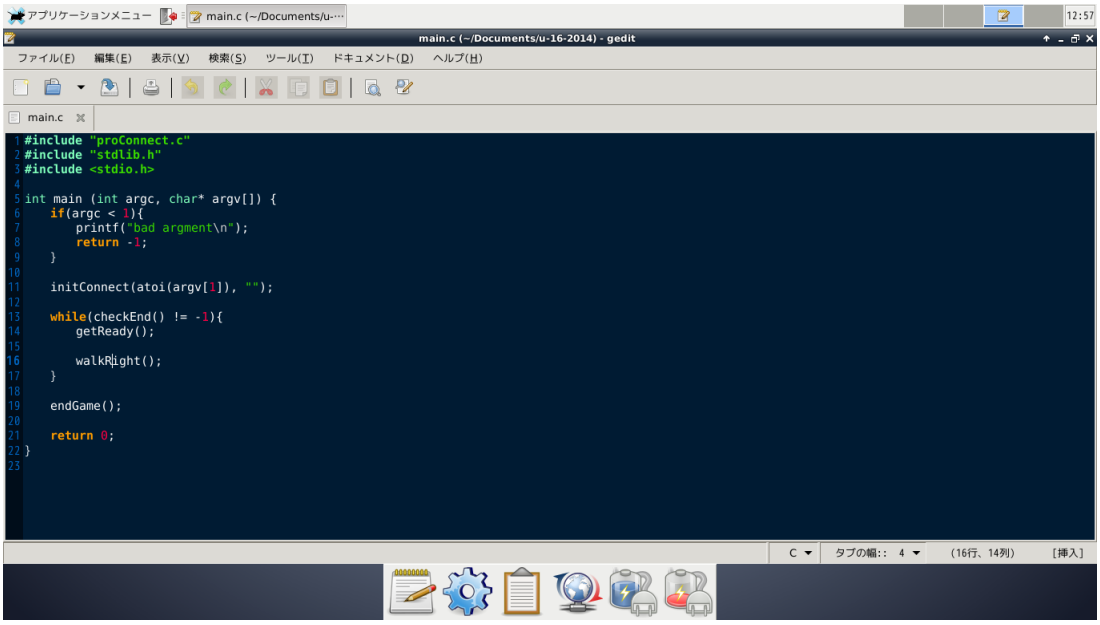


図 2.2　プログラムソースを書く画面

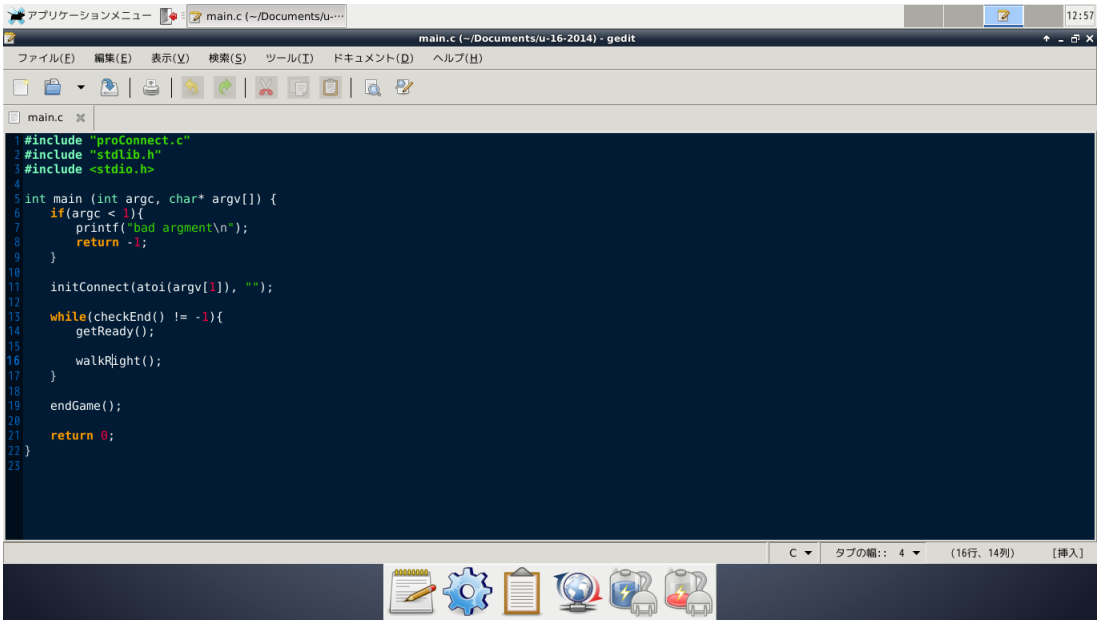


図 2.3　保存する場所

(2)  : コンパイルを行います．このボタンを押したら，一件何も変わりませんが，裏側でコンパイルが行われているので，本当に押したか分かりづらいので 2，3 回押すことをお勧めします．

(3)  : これは，コンパイル結果を調べるコマンドです．以

下の画像の様に何も書かれていなければ**成功**です．

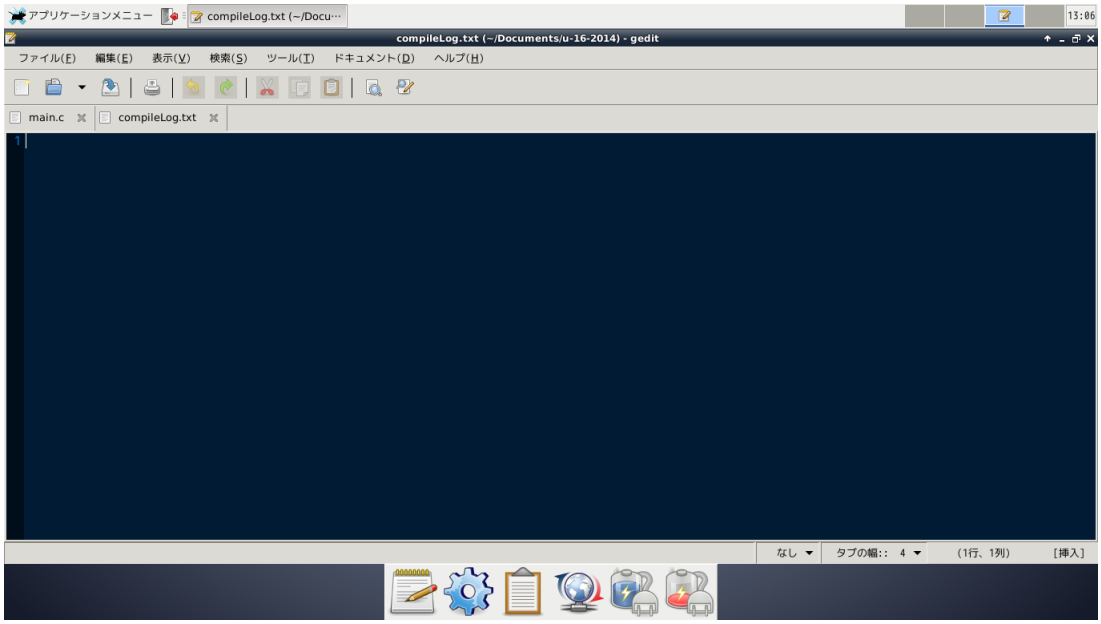


図 2.4　コンパイル成功時の画面

逆に，次の様に書かれていれば**失敗**です．最初に戻ってプログラムを書き直すことになります．

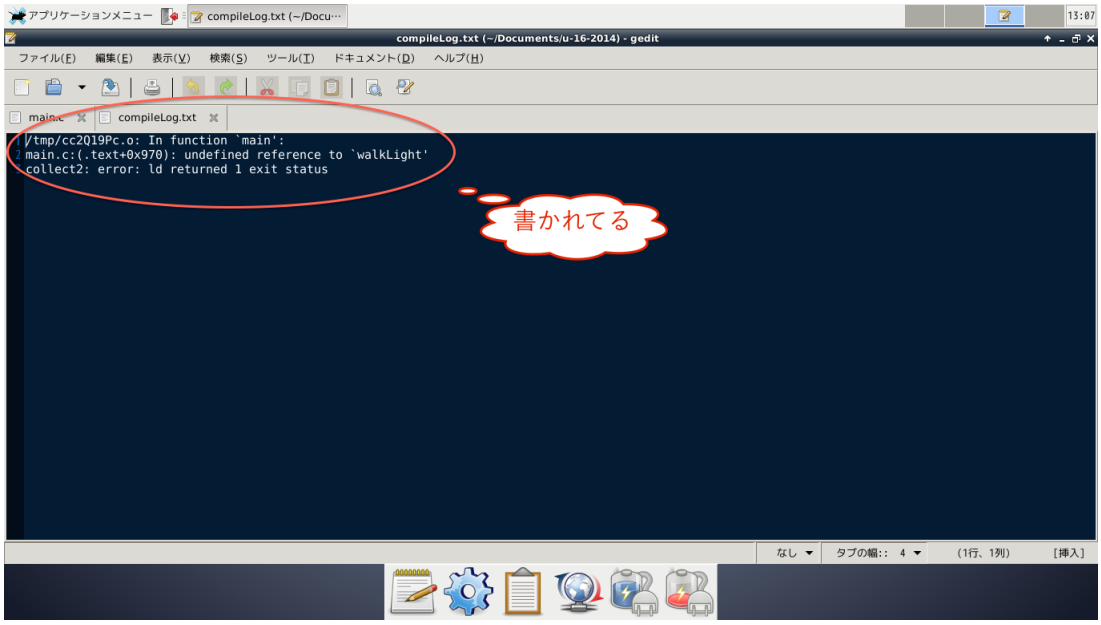


図 2.5　コンパイル失敗時の画面

また，何度かコンパイルし直すと，以下の様な画面になりますが，このときは上にある [再読み込み (R)] を選択して下さい．

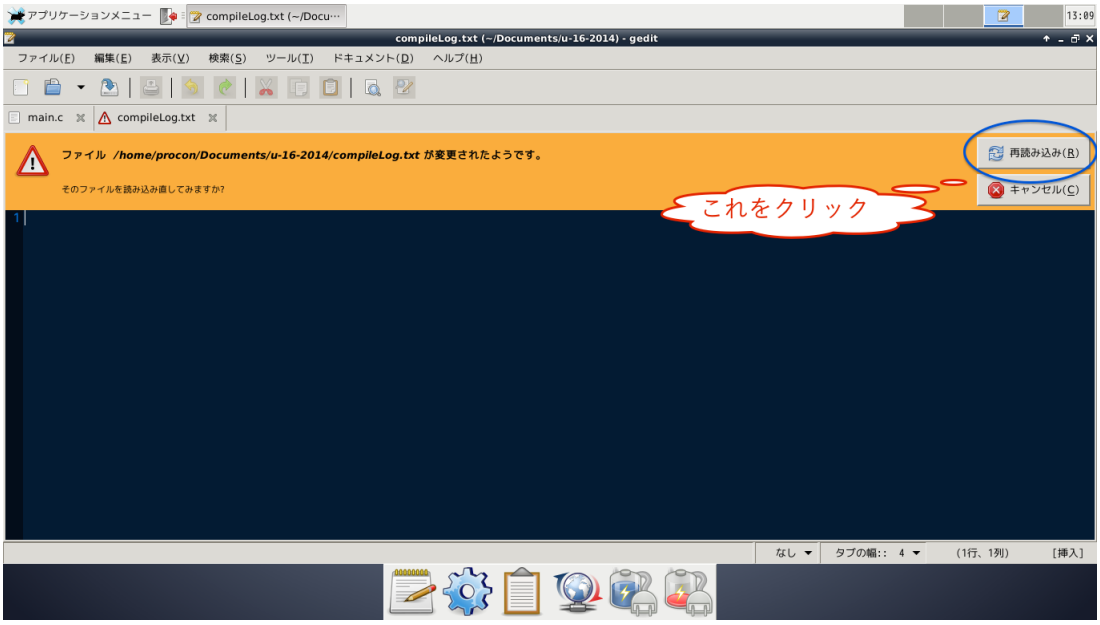


図 2.6　再読み込み

(4)  : サーバプログラムを起動します．起動すると，次の様な黒い画面が出てくると思います．

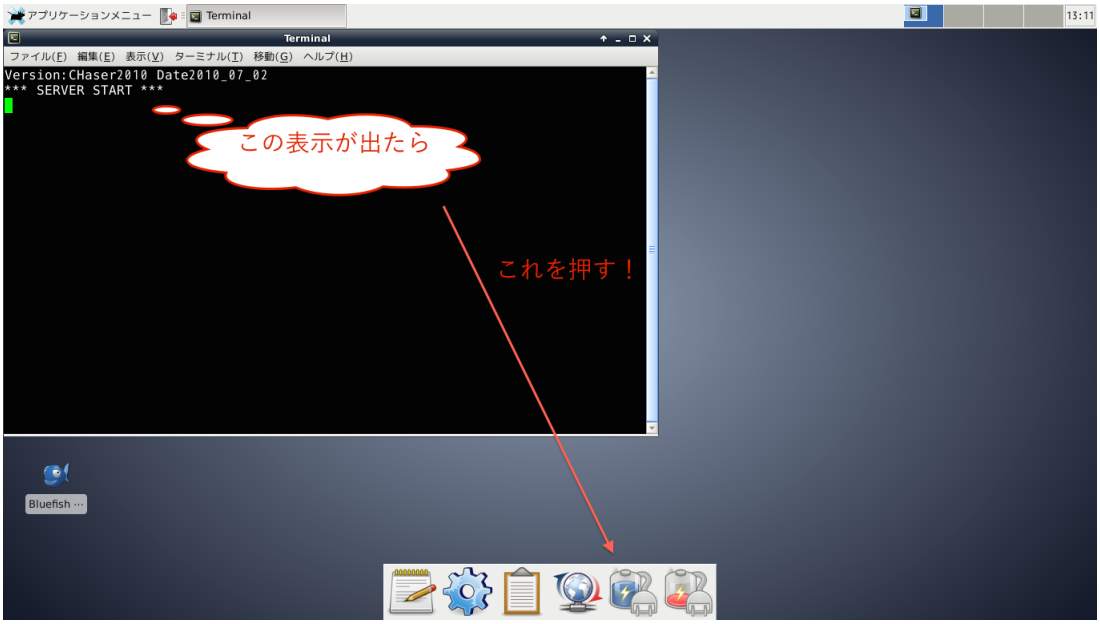


図 2.7　サーバプログラム起動時の画面

この画面に「*** SERVER START ***」と表示されていれば完了です．次の項目に進みます．

(5)  :COOL(先攻側) で接続します．押したら，先ほどの黒い画面に「Ｃｏｏｌが【接続時の名前】で接続しました.」と表示されたら成功です．表示されてなかった場合，実はコンパイルが失敗していたなどの可能性があります．

(6)  :HOT(後攻側) で接続します．成功したらさきほどと同様に「Ｈｏｔが【接続時の名前】で接続しました.」と表示されるとともに，以下の様にマップなども表示されます．

図 2.8　ゲーム開始前の画面

このとき，上にある [Terminal] と書かれたバーを選択して，エンターキーを押すとゲームが始まります．

こうすると，ゲームが開始します．上の方に「Cool win!!」または「Hot win!!」と表示されたら決着がついたと思って大丈夫です．

そして，ゲームを終了させるときは，**ゲームが終了したのを確認してから**フィールドをマウスで一回クリックして，[Alt + F4] を押して下さい．そうすると終了します．また，ゲーム終了してからすぐサーバを起動すると，前の対戦結果が完全に終了していないため，勝負が始まらない可能性があります．なので，一息ついてから行動して下さい．

第 3 章

競技ルール

—概要—

この章では，プログラミングコンテスト競技部門のルールについて解説します．ゲームの基本となるため，ぜひ目を通しておいて下さい．また，中にはルールとして知ってるのと知らないのでは全然ゲームの難易度が違ってくるルールもあるので，隅から隅まで詳しく読んで少しでも優勝のために頑張って下さい．

3.1 説明に使う用語・記号について

ゲームに使う基本的な用語や記号と実際のゲームとの対応を説明します．

プレイヤ: ゲームに接続するプログラム (つまり，今回の大会で皆さんに書いてもらうプログラム) とそれを使って対戦する人です．

サーバ: 接続されて，対戦の状況などを管理するプログラムのことです．

COOL: ゲームの先攻側です．
サーバに先に接続したプログラムが先攻になります．

HOT: ゲームの後攻側です．
サーバに後から接続したプログラムが後攻になります．

ターン: 自分が 1 回行動する一連の動作 (getReady，周辺情報の取得，メソッド，周辺情報の取得) をターンと呼びます．

フィールド: ゲームを行う場所のことです．

周辺情報: 自分の居るマスを含めた周囲 9 マスの状態とゲームが終了しているか否かの情報のことです，

メソッド 自分のターン中に 1 つ行える動作のことです．

また，この章ではメソッドの説明など，言葉だけではイメージが掴みづらい部分が多いです．なので，これから使われる図と用語について対応表を書いておきます (そして，COOL を自分として書いていますが，HOT でも同様です)．

自分 (COOL) C

相手 (HOT) H

普通の床

ブロック

アイテム

3.2 画面のサンプル

具体的なサンプルをもとに実際にどのような配置になるかを見てみたいと思います．実際の競技画面は以下の様になります．

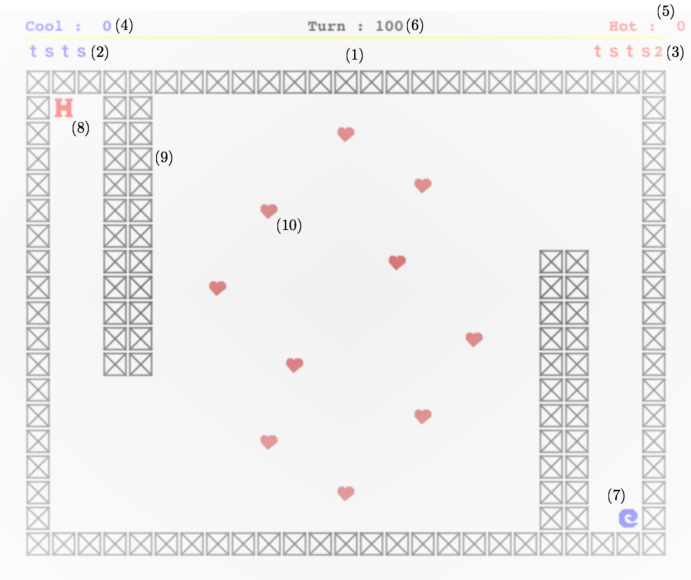


図 3.1　実際の画面

(1) フィールド
(2) COOL の名前
(3) HOT の名前
(4) COOL が獲得したアイテムの個数
(5) HOT が獲得したアイテムの個数
(6) 制限ターン数 (ゲーム開始からは終了までのターンのカウントダウンとなる)
(7) COOL のアイコン
(8) HOT のアイコン
(9) ブロック
(10) アイテム

3.3 ルール

3.3.1 基本的なルール

⋆ 競技は，サーバに接続された 2 人のプレイヤによる 1 対 1 での対戦になります．
⋆ 先に接続した側を先攻になり，後に接続した側が後攻になります．

- ⋆ 勝利条件 (3.3.2 節を参照) を先に満たしたプレイヤが勝利となります.
- ⋆ 競技に使用するフィールドは，15 × 17(縦 15 マス，横 17 マス) です.
- ⋆ アイテム・ブロックは，マップの中心から**点対称**になる様に配置されます.
- ⋆ アイテムを取得したとき，元居たマスは以下の様になります (下記の図を参照).

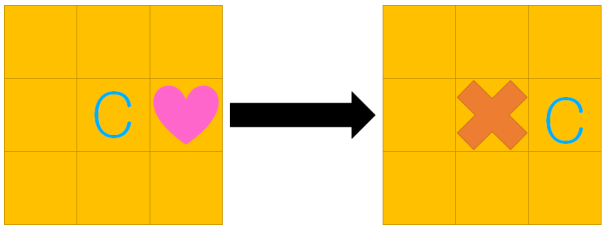


図 3.2　動作の例

3.3.2 勝利条件

- ⋆ 決められた，制限ターンになったときに，アイテムを相手よりも多く集めている.
- ⋆ ブロックを対戦相手の上に乗せる.
- ⋆ 対戦相手の上下左右をブロックで囲み，動けなくする.
- ⋆ 対戦相手がブロックの上に移動する.
- ⋆ 対戦相手が試合続行不可能になる.

この条件で勝負がつかなかった場合は引き分けとなります.

3.4 ゲームで行える行動

3.4.1 getReady

自分のターンがくるまで待機します．自分のターンが来たとき，サーバから与えられる周辺情報を取得します.
また，ターンの間で起動した場合，試合が続行出来ないのでターンの間は起動しない様にして下さい！

3.4.2 メソッド

メソッドは，“[動作][Up, Right, Left, Down のどれか]” という形で書かれています (例 :“walkRight”)．今回，載せる図などはすべて “Right” での動作です．ほかの “Up” などは適時読み替えて下さい．

walk

指定された上下左右のどれかの方向に移動します．メソッド実行後に取得される周辺情報は移動後の周辺情報です．

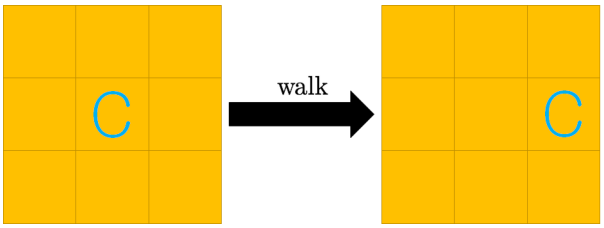


図 3.3　walk のイメージ (walkRight の場合)

look

指定された上下左右のどれかの方向に対して，正方形状に 9 マスの情報を取得します (下記の図を参照)．メソッド実行後に取得される周辺情報は look で取得した情報です．周辺情報割当ての順番は図に割り振られている番号と一緒です．

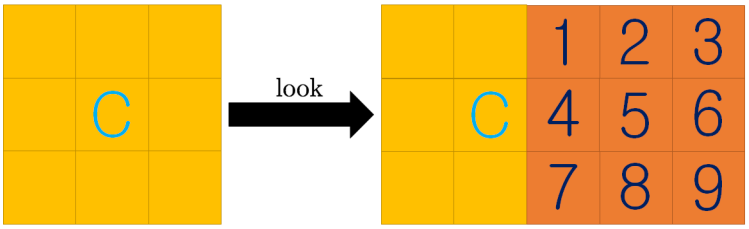


図 3.4　look のイメージ (lookRight の場合)

search

指定された上下左右のどれかの方向に対して，直線上に 9 マスの情報 (下記の図を参照) を取得します．メソッド実行後に取得される周辺情報は search で取得した情報です． 周辺情報割当ての順番は図に割り振られている番号と一緒です．

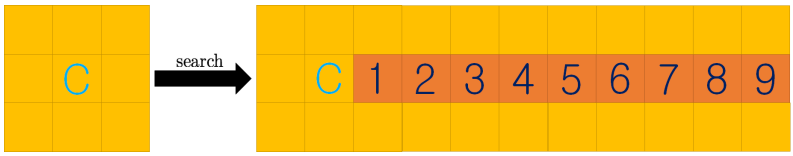


図 3.5　search のイメージ (searchRight の場合)

put

指定された上下左右のどれかの方向に対して，ブロックを配置します．メソッド実行後に取得される周辺情報は，ブロックをおいた後の周辺情報です．

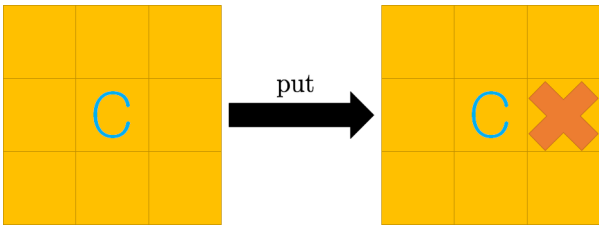


図 3.6　put のイメージ (putRight の場合)

3.5 周辺情報

サーバから送信される周辺情報については，具体的には以下の通りの様な仕様となっています．

(1) 10 桁の 0 から 3 までの数字です．
(2) 1 桁目には，ゲームの状態 (終了しているかどうか) を表しています．0 が終了状態で，1 が続行している状態です．

(3) 2 桁目から 10 桁目には対応する自分の周囲 9 マスの情報 (図 3.7 参照) が保存されています (5 番目が自分の現在居る位置です)．また，各桁に 0 から 3 までの数字が割り当てられて，以下に示す様な意味になります．

図 3.7　周辺情報の割当図

表 3.1　周辺の状態割り当て表

割り当てられている数値	状態
0	なにもなし
1	相手プレイヤがいる
2	ブロックがある
3	アイテムがある

以下に，実際の状態と値の対応表を書きます．

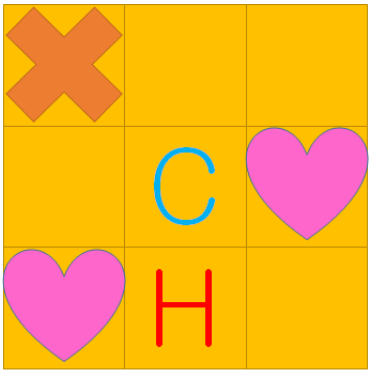


図 3.8　周辺情報の例

表 3.2　周辺情報の値

周辺情報の桁	値
1	1
2	2
3	0
4	0
5	0
6	0
7	3
8	3
9	1
10	0

3.6 具体的な通信の仕様

Warning!：**この章は，自分で新しくライブラリを作成する人向けの章です．それ以外の人は読まなくても問題有りません．**

別の言語などで，独自のライブラリを構築する場合，以下の仕様にのっとって作ればサーバとの通信が行えます．

(1) サーバとの接続を確立 (接続のポート番号は前述の値を参

照して下さい),

(2) サーバへチーム名を送信します (ここで，先頭 5 文字目以降は無視されます).

(3) サーバが送信する “@” を受け取ります.

(4) “gr\r\n” をサーバに送信します (これを，本テキストでは getReady と呼んでいます).

(5) サーバ側から送信される 10byte の文字列を受け取ります (これを，本テキストでは周辺情報と呼び，実装では value に格納しています). 例としては, “1000000000” となります.

(6) サーバにメソッドを表現する 2 文字とその末尾に “\r\n” を付加した文字列を送信します．表現の対応表は,

表 3.3　メソッドの表現対応表

	Up	Right	Left	Down
walk	“wu”	“wr”	“wl”	“wd”
look	“lu”	“lr”	“ll”	“ld”
search	“su”	“sr”	“sl”	“sd”
put	“pu”	“pr”	“pl”	“pd”

(7) (5) と同様に周辺情報を受け取ります.

(8) サーバに “#\r\n” を送信します.

(9) 以降，周辺情報の最初の文字が “0” になるまで，(4) からここまでを繰り返す.

以下に概要図を示すので，これを利用して作成に役立てて下さい.

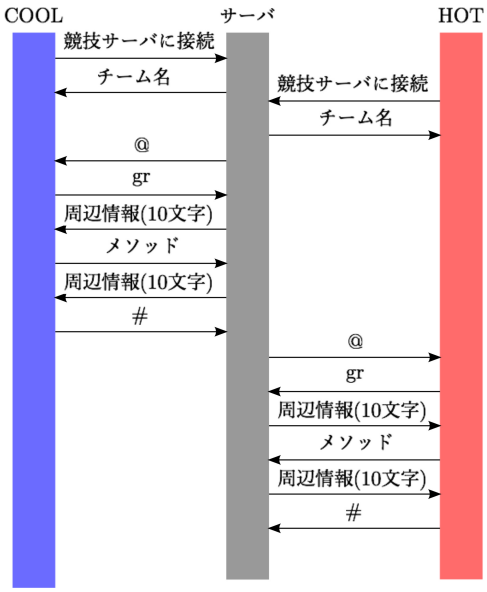


図 3.9　動作模式図

第 4 章

ライブラリの詳細

—概要—

今回のプログラミングコンテストで使用する ***proConnect.h*** について詳細な説明を加えます．基本的には必要最低限の機能しか搭載していないために，仕様の詳細といっても，今大会で，楽に勝てる助けとなるなツールとしての期待は出来ません．しかし，独自の言語・環境でライブラリを書き直したいしたいのなら，proConnect.h の詳細な仕様を知る事で大きな手助けとなれることと思いますので，ぜひそのような野心を持つ読者の皆さんには興味を持って読んでもらいたいと思います．　また，今回のライブラリ作成にあたって DLL などでの配布ではなく敢えてソースファイルとしての配布としました．これは，実際にライブラリを読んで具体的な接続方法・各関数の動きなどを理解してもらい実際に書きやすくするためです．なので，興味を持って頂けたのならぜひ一回は見てみて下さい．

4.1 構成要素

このヘッダファイルは，パラメータの最大値・設定番号を記録してある**定数**，サーバから受信された値を記録する**変数**，サーバへ動作を送信する**関数**の 3 つの要素で構成されています．基本的に名称は，ルールセクション (第 3 章参照) と同じです．なので，わからない単語が出て来た場合はルールセクションに戻って意味を再確認することをお勧めします．

定数，変数，関数については具体的な説明はプロコンの講習資料を参照して頂くとしてこの章ではざっくりと以下の意味で使い分けていることを先に明らかにしておきます．

定数: プロコンを通して (仕様変更が無い限り) 変化しない数値，または文字．

変数: ゲーム中に，サーバ側もしくはプレイヤ側からから変更される可能性のある値，または文字．

関数: ゲーム中に，複雑な動作をひとまとまりにしたもの．

4.2 定義一覧

まずは，ここの具体的な説明を避けて一覧とその概要を載せます．

表 4.1　proConnect.h に含まれる定義の一覧

種別	定義名	概説
定数	HOTPORT	HOT（後攻）の接続ポート番号
	COOLPORT	COOL（先攻）の接続ポート番号
	NAME_MAX	サーバに送信されるチーム名の最大文字数
	VALUE_SIZE	value の最大値
変数	value	サーバから送信される周辺情報を記録する
関数	initConnect	サーバへの接続処理を行う
	coolConnect	CPP+ として接続処理を行う
	hotConnect	HOT として接続処理を行う
	getReady	待機状態になる
	checkEnd	ゲームが終了しているかどうかの判定
	endTurn	ターンの終了処理を行う
	endGame	ゲームの終了処理を行う
	walkUp	上方向へ walk する
	walkLeft	左方向へ walk する
	walkRight	右方向へ walk する
	walkDown	下方向へ walk する
	lookUp	上方向へ look する
	lookLeft	左方向へ look する
	lookRight	右方向へ look する
	lookDown	下方向へ look する
	searchUp	上方向へ search する
	searchLeft	左方向へ search する
	searchRight	右方向へ search する
	searchDown	下方向へ search する
	putUp	上方向へ put する
	putLeft	左方向へ put する
	putRight	右方向へ put する
	putDown	下方向へ put する

4.3 定数・変数

4.3.1 COOLPORT

サーバへの接続の際に COOL(先攻側) として接続するためのポート番号を表す定数です[*1].

4.3.2 HOTPORT

サーバへの接続の際に HOT(後攻側) として接続するためのポート番号を表す定数です[*2].

4.3.3 NAME_MAX

送信出来る最大文字数を表す定数です．ただし，ここで言う定数は**バイト数で換算したもの**です (つまり，日本語などの 2 バイト文字は 2 文字分使う).

4.3.4 VALUE_MAX

value(周辺情報を表す変数) の長さを表す定数です.

4.3.5 value

周辺情報を表す変数です．格納の順番などの詳細は，ルールセクションの項目を参照して下さい.

[*1] 現在は，2009.
[*2] 現在は，2010.

4.4 関数

4.4.1 initConnect

プロトタイプ宣言

```
void initConnect(const int port, const char* name);
```

説明

サーバとの接続を行う関数です．ゲームの開始時に接続を行います．引数の port は，接続に行うポート番号，name は接続時に表示される名前です．

接続に失敗した場合は，エラーを返してプログラムを終了します．接続に成功した場合は，“Connect success : [port 番号]” と表示します．

4.4.2 coolConnect

プロトタイプ宣言

```
void coolConnect(const char* name);
```

説明

COOL(先攻側) としてサーバとの接続を行う関数です．initConnect の port に COOLPORT を設定した場合と同じ動作をします．

4.4.3 hotConnect

プロトタイプ宣言

```
void hotConnect(const char* name);
```

説明

HOT(後攻側) としてサーバとの接続を行う関数です．initConnect の port に HOTPORT を設定した場合と同じ動作をします．

4.4.4 getReady

プロトタイプ宣言

void getReady(void);

説明

自分のターンまでの待機状態に入る関数です．実行すると，value が getReady 時の周辺情報に書き換えられます．動作に成功すると，“getReady：[周辺情報 (10 桁)]” という風に表示されます．

動作として，ターンの間で呼び出されると，無限ループに陥り，またサーバーとの通信が途切れるとプログラムを強制終了します．

4.4.5 checkEnd

プロトタイプ宣言

int checkEnd(void);

説明

ゲームが終了しているかどうかを判定する関数です．終了していれば-1 を返し，そうでなければ 1 を返します．

また，終了時には “checkEnd：End”，そうでなければ “checkEnd：not End” と表示します．

4.4.6 walkUp

プロトタイプ宣言

void walkUp(void);

説明

上側に walk します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“walkUp：[周辺情報 (10 桁)]” を表示します．

ゲームが終了した場合は動作は実行されません．

4.4.7 walkLeft

プロトタイプ宣言

void walkLeft(void);

説明

左側に walk します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“walkLeft：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.8 walkRight

プロトタイプ宣言

void walkRight(void);

説明

右側に walk します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“walkRight：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.9 walkDown

プロトタイプ宣言

void walkDown(void);

説明

下側に walk します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“walkDown：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.10 lookUp

プロトタイプ宣言

void lookUp(void);

説明

上側に look します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“lookUp：[周辺情報 (10 桁)]” を表示します．

ゲームが終了した場合は動作は実行されません．

4.4.11 lookLeft

プロトタイプ宣言

void lookLeft(void);

説明

左側に look します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“lookLeft：[周辺情報 (10 桁)]” を表示します．

ゲームが終了した場合は動作は実行されません．

4.4.12 lookRight

プロトタイプ宣言

void lookRight(void);

説明

右側に look します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“lookRight：[周辺情報 (10 桁)]” を表示します．

ゲームが終了した場合は動作は実行されません．

4.4.13 lookDown

プロトタイプ宣言

void lookDown(void);

説明

下側に look します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“lookDown：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.14 searchUp

プロトタイプ宣言

void searchUp(void);

説明

上側に search します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“searchUp：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.15 searchLeft

プロトタイプ宣言

void searchLeft(void);

説明

左側に search します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“searchLeft：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.16 searchRight

プロトタイプ宣言

void searchRight(void);

説明

右側に search します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“searchRight：[周辺情報 (10 桁)]” を表示します．

ゲームが終了した場合は動作は実行されません．

4.4.17 searchDown

プロトタイプ宣言

void searchDown(void);

説明

下側に search します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“searchDown：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.18 putUp

プロトタイプ宣言

void putUp(void);

説明

上側に put します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“putUp：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.19 putLeft

プロトタイプ宣言

void putLeft(void);

説明

左側に put します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“putLeft：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.20 putRight

プロトタイプ宣言

void putRight(void);

説明

右側に put します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“putRight：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.21 putDown

プロトタイプ宣言

void putDown(void);

説明

下側に put します．成功した場合はターンの終了処理を行った上で行動後の周辺情報を value に更新し，“putDown：[周辺情報 (10 桁)]” を表示します．
ゲームが終了した場合は動作は実行されません．

4.4.22 endTurn

プロトタイプ宣言

void endTurn (void);

説明

ターンの終了処理を行います，動作系の命令 (walk，look, search, put) 終了後にそれぞれの関数側で呼び出されるので，特に呼び出す必要は有りません．
また，この関数の起動時に周辺情報を更新します．

4.4.23 endGame

プロトタイプ宣言

int endGame (void);

説明

ゲームの終了時に呼び出すプログラムです．ゲームが終了後に呼び出して下さい．