

第 1 章

ガイダンス・標準入出力を用いた自動テスト

2017-09-27 Wed

概要

情報数理学演習 II では、アルゴリズムとデータ構造に関連した演習を行う。概念の理解だけでなく、正しく動作するプログラムを書ける能力を養うことを重視する。ただし、プログラムを書く能力の演習はプログラミング演習 II で行っていることを前提に進める点に注意。

一学期間で以下のトピックを扱う: 基本データ構造の活用 (stack, queue, string, priority queue, set, map), 動的計画法 (線形, 二次元, 木), グラフ (最小全域木, 最短路, 探索), 計算幾何, 構文解析. ゆとりがあれば発展的な話題 (たとえば Segment tree, FFT) も交える。

1.1 オンラインジャッジで学ぶプログラミング

夏学期のプログラミング演習 II では、テストケースを自分で作成してプログラムの正しさを検証することを学んだ。冬学期は、インターネット上で稼働するオンラインジャッジを併用して、可能な範囲でテストの部分を自動化する。オンラインジャッジは、そこに提出されたプログラムを、予め作成されたテストケースを用いて判定 (ジャッジ) するサーバである。

1.1.1 先学期との違い

- 先学期は、関数の引数と計算結果の対応を直接テストした。

例: `sum(2, 3) == 5`

今学期は、様々なテストデータを試せるように、テストのためのデータを標準入力から読み込み、計算結果を出力するプログラムを作成する。

```
C++
1  int a, b;
2  cin >> a >> b; // たとえば 2 と 3 を読み込む
3  cout << sum(a, b) << endl; // 出力の正しさを入力作成者が確認
```

変数 `a, b` は標準入力から読む。簡単には、キーボードの入力が標準入力に相当し、`2 3` と打ち込めば

`sum(2, 3)` をテストすることに相当する。なお、後述するリダイレクションという方法により、ファイルの内容を標準入力に与えることができる。自動テストにはこの方法が用いられる。

- 問題概要とともに、サンプル入出力 (少量のテストケース) が与えられる。
- ジャッジの入出力 (網羅的なテストケース) は、別にあり、通常は非公開で量も多い。
- プログラミング言語は、C++ を標準とする。C++ を思い出すこと。本日は C++ で行うこと。来週以降は特に断った場合を除いて、既に C++ に習熟している人は、Java や Ruby, Python などを使っても良い (eecs で動作しかつオンラインジャッジが受け付けるものであれば、何を使っても良い)。ただし、一部の問題は、実行速度の問題で C++ でしか、時間やメモリ制限内で処理が終わらない可能性がありうる。

1.1.2 アカウント作成と使い方

主に以下のオンラインジャッジを用いる。メインは AOJ である。

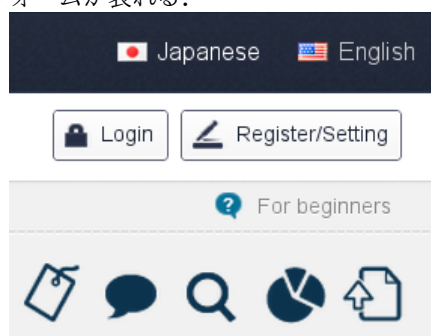
- Aizu Online Judge (AOJ) <http://judge.u-aizu.ac.jp>
- Peking University Judge Online for ACM/ICPC (POJ) <http://poj.org>
- Codeforces <http://www.codeforces.com/>
- MAIN.edu.pl <http://main.edu.pl/en>

初めの場合まずは AOJ のアカウントを作成する。各システムとも無料で使うことができる。なお、各オンラインジャッジは、運営者の好意で公開されているものであるから、迷惑をかけないように 使うこと。特に パスワードを忘れない こと。

■AOJ のアカウント作成 (初回のみ) ページ右上の Register/Setting からアカウントを作成する。User ID と Password を覚えておくこと (ブラウザに覚えさせる、もしくは暗号化ファイルにメモする)。この通信は https でないので、注意。Affiliation は the University of Tokyo 等とする。E-mail や URL は記入不要。

ここで、自分が提出したプログラムを公開するかどうかを選ぶことができる。公開して (“public” を選択) いれば、プログラムの誤りを誰かから助けてもらう際に都合が良いかもしれない。一方、「他者のコード片の動作を試してみる」というようなことを行う場合は、著作権上の問題が発生しうるので、非公開の方が良いだろう (“private” を選択)。

■AOJ への提出 (毎回) ログイン後に問題文を表示した状態で、長方形に上向き矢印のアイコンを押すと、フォームが表示される。



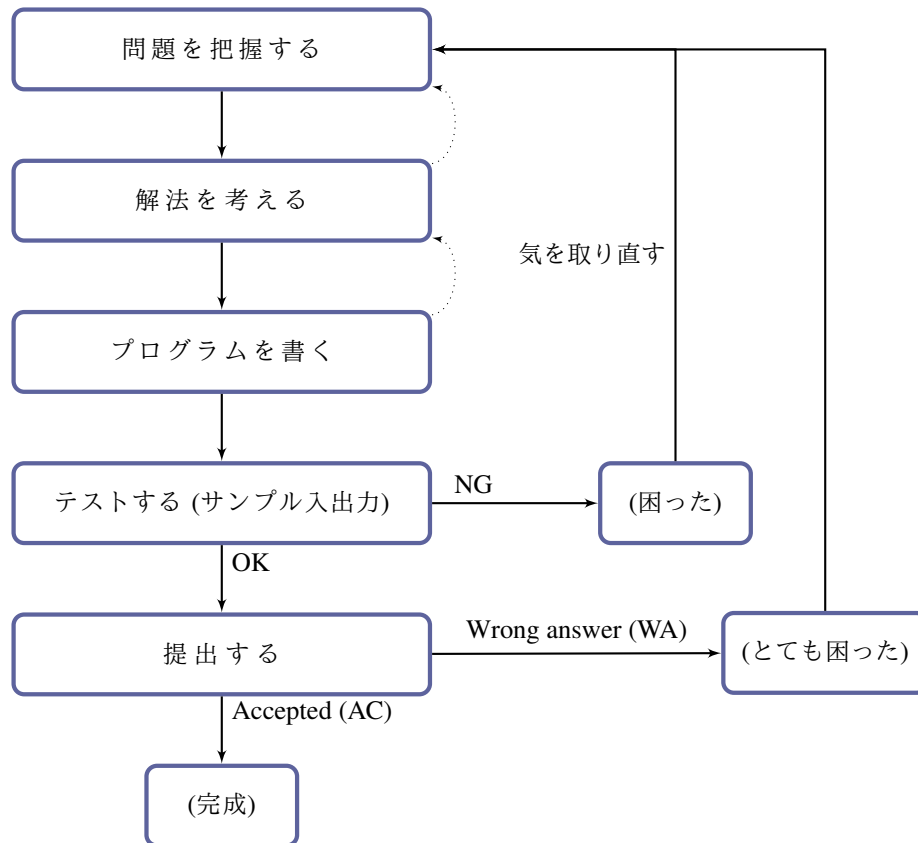
自分の提出に対応する行 (“Author” を見よ) の “Status が “Accepted” なら正答。

■正答でなかった時 様々な原因がありうるので、まずジャッジの応答がどれにあてはまるか、システムの使い方を誤解していないかなどを説明を読んで確認する。Submission notes (http://judge.u-aizu.ac.jp/onlinejudge/submission_note.jsp), Judge's replies (http://judge.u-aizu.ac.jp/onlinejudge/status_note.jsp), チュートリアル (http://judge.u-aizu.ac.jp/onlinejudge/AOJ_tutorial.pdf) などの資料がある。経験が少ない段階では、15分以上悩まないことをお勧めする。手掛かりなく悩んで時間を過ごすことは苦痛であるばかりでなく、初期の段階では学習効果もあまりないので、指導者や先輩、友達に頼る、あるいは一旦保留して他の問題に取り組んで経験を積む方が良いだろう。相談する場合は、「こう動くはずなのに(根拠はこう)、実際にはこう動く」と問題を具体化して言葉にしてゆくと解決が早い。なお、悩んで意味がある時間は、熟達に応じて2時間、2日間等伸びるだろう。

1.2 実践例

例題	X-Cubic	(AOJ)
1つの整数 x を読み、 x の3乗を出力するプログラムを作成せよ http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ITP1_1_B		

この問題を解いてみよう。



1.2.1 ファイルの作成

エディタ (Emacs, mi などお好みで) を起動し、以下のファイルを作り、xcubic.cc という名前で保存する。

```

C++
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int x;
5     cin >> x;
6     cout << 2*2*2 << endl; // bug
7 }
  
```

1.2.2 コンパイル

(以降\$記号は、ターミナルへのコマンド入力を示す)

C++ の場合:

```
$ g++ -Wall xcubic.cc
```

1

-Wall は警告を有効にするオプションで、何かメッセージが出た場合は解消することが望ましい。読み方が

分からないメッセージが出た場合は、誰かと相談する。

1.2.3 実行・テスト

実行例: 以下, 斜体はキーボードからの入力を示す。

\$./a.out	1
2	2
8	3

($2^3 = 8$, 成功)

\$./a.out	1
3	2
8	3

($3^3 = 27 \neq 8$, 失敗!)

1.2.4 ソースコード修正・再テスト

各自試すこと。

手元でうまく動くことを確認したら, AOJ に提出する。Accepted と判定されることを確認する。

1.3 標準入出力とリダイレクション

先の例題ではテストケースが数一つだったので, 目で見れば事足りた。しかし, ある程度の複雑なプログラムをテストするには数一つでは不十分で, テストケースは大きくなる場合がある。たとえば, この問題は最大 1 万個の数値列を扱う。また出力も大きくなりうる。そのような際には, 入力と, 出力の検証を自動化すると良い。

(以下の資料では, 意図的に誤りに誘導する部分があるが, 気づいた場合もしばらくは声に出さないこと)

問題	Min, Max and Sum	(AOJ)
n 個の整数 $a_i (i = 1, 2, \dots, n)$ が与えられるので, それらの最小値, 最大値, 合計値を表示せよ http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ITP1_4_D		

■入出力 以下のようなプログラムを作成する:

```
C++
1  /* minmaxsum.cc
2   */
3  #include <iostream>
```

```

4  using namespace std;
5  // 簡潔なスタイル
6  int N, A[10010]; // 少しだけ多めに取り
7  int min() {
8      return ...;
9  }
10 int max() {
11     return ...;
12 }
13 int sum() {
14     return ...;
15 }
16 int main() {
17     cin >> N;
18     for (int i=0; i<N; ++i)
19         cin >> A[i];
20     cout << min() << ' ' << max() << ' ' << sum() << endl;
21 }

```

変数 `N` や `A` は名前も含めて問題文に対応するものなるべく使用し、さらに、本質的な部分に集中するために `global` 変数とすることを勧める。配列 `A` の長さは入力により異なるが、最大値はたかだか 1 万なので、ケチケチせず最大値を確保する (気にする場合は `vector` を用いてもよいが、`new` や `delete` の使用は勧めない)。

なお、実用的なプログラムを書く際は、変数のスコープを必要な範囲に狭めることが望ましい。必要であれば、以下のような書き換えられることを前提に、本演習では `global` 変数を用いる書き方を勧めている。

C++

```

1  class MinMaxSumSolver { // 行儀の良いスタイル
2      int N;
3      vector<int> A;
4  public:
5      void setup();
6      int min() {
7          return ...;
8      }
9      int max() {
10         return ...;
11     }
12     int sum() {
13         return ...;
14     }
15     void solve();
16 };
17 int main() {
18     auto_ptr<MinMaxSumSolver> solver(new MinMaxSumSolver());
19     solver->solve();
20 }

```

■リダイレクションを用いたテスト まず `sample-input.txt` を、作成しておく。数字を 1 文字ずつ入力するのではなく、コピーペーストを用いること (打ち間違いを防ぐため)。本資料では作成できていることを示すために `cat` の実行例を載せることがある。

```
$ cat sample-input.txt 1
5 2
10 1 5 4 17 3
```

リダイレクションを使った実行 (キーボード入力の代わりにファイルから読み込む):

```
$ ./a.out < sample-input.txt 1
1 17 37 2
```

実行結果をファイルに保存 (画面に表示する代わりにファイルに書き込む):

```
$ ./a.out < sample-input.txt > my-output.txt 1
$ cat my-output.txt 2
1 17 37 3
```

2 つのファイル比較:


```
$ diff -u sample-output.txt my-output.txt 1
```

(差分がある場合のみ、出力がある)

資料:

- HWB 15 コマンド
<http://hwb.ecc.u-tokyo.ac.jp/current/information/cui/>
- HWB 14.4 コマンドを使ったファイル操作
<http://hwb.ecc.u-tokyo.ac.jp/current/information/filesystem/cui-fs/>

■ヒント 普通に組むと、Accepted ではなく Wrong Answer となる場合がある。

Run#	Author	Problem	Status	%	Lang	Time	Memory	Code	Submission
									Date
1515235	kaneko	ITP1_4_D: Min, Max and Sum	 Wrong Answer	18/20	C++	00:00 s	1200 KB	362 B	2015-09-16 10:50

中央付近の 18/20 という数字は、テストケースの 18 番目まで正答し、19 個目で失敗したという意味である。その部分がハイパーリンクになっているのでクリックすると、詳細が分かる。

Case #16:	✓ : Accepted	00.00 sec	1168 KB
Case #17:	✓ : Accepted	00.00 sec	1200 KB
Case #18:	✓ : Accepted	00.00 sec	1196 KB
Case #19:	✗ : Wrong Answer	00.00 sec	1200 KB

さらに Case #19: の行をクリックすると、実際のデータを見ることができる (問題による).

< prev | 19 / 20 | next > 04_maximum_02.in ✗ : Wrong Answer 00.00 sec 1200 KB

Judge Input #19 (in19.txt | 68926 B)

```
10000
430143 602887 783032 225925 905915 978433 239648 49
```

Judge Output #19 (out19.txt | 21 B)

```
28 999997 5019101515
```

このデータを手元で試してみよう。データは、コピーペーストするには大きすぎるので、まず in19.txt の部分をクリックしてダウンロードする。環境やブラウザによるが ITP1_4_D_in19.txt という名前でダウンロードフォルダに保存されたとすると、適宜、リダイレクションによって実行する。

```
$ ./a.out < ~/Downloads/ITP1_4_D_in19.txt 1
28 999997 724134219 2
```

また、今回はプログラムの出力と Judge Output との間に差があることは一目でわかるが、一般に 5019101515 のような桁数の数字を目で比較するのは困難であるから (1 文字異なっても気づかない)、同様にダウンロードして diff コマンドにより比較するのが良い。

(原因と対策の解説は、来週にも行う予定だが、白文字でこちらにも記しておく。コピーペーストなどで読めるはずだが、読む前に、十分に=15 分間は仮説を検討すること。却下した仮説も含めて文字で記しておく方が良い。)

1.4 本日の課題

- 問題を解いて、AOJ で Accepted を得たことを確認して、感想 (学んだことなど、特になければ合計所要時間と、どこにどの程度時間がかかったか) とともに、ITC-LMS に提出する。本日の到達目標は AOJ の使い方に一通り馴染むことであるので、発展課題は特に設けていない。

提出の注意:

- ソースコードにコメントを書き入れた テキストファイル を提出する (PDF, Microsoft Word, rtf などは避けること)。コメントの記述は、`#if 0` と `#endif` で囲むことが簡便である。

```
C++
1 // minmaxsum.cc
2 #if 0
3   所要時間は..であった...を復習した.
4   ...
5 #endif
6 int main() {
```


- 提出物は履修者全体に公開される。 この授業では、他者のソースコードを読んで勉強することを推奨するが、課題を解く上で参考にした場合は、何をどのように参加したかを明記すること。

1.A 指定したフォルダへのファイルの保存とコンパイル

冬学期もフォルダを活用して、ソースコードを整理することをすすめる。

各回では、サンプルや機能確認のために複数のコード片を扱う。そこで、混乱を避けるため、フォルダを作って、テーマごとに別の名前をつけてファイルに保存すると良い。そして、それぞれを動作可能に保つ。一方、お勧めしない方法は、一度作ったファイルを継ぎ足しながら、一つの巨大なファイルにすることである。そうしてしまうと、後から、2 種類のコードを実行して差を比べることが難しくなる。

「ターミナル」(MacOSX の場合) の動作例は以下の通り:

```
$ mkdir Math-IS-II 1
# (フォルダ作成, 最初の一回のみ) 2
$ mkdir Math-IS-II/week1 3
# (各週毎に行う) 4
$ cd Math-IS-II/week1 5
# (カレントディレクトリを変更. $HOME/Math-IS-II/week1/ 以下にソースコードを保存) 6
$ g++ -Wall sample1.cc 7
$ ./a.out 8
# (実行) 9
```

1.B バグとデバッグ

プログラムを書いて一発で思い通り動けば申し分ないが、そうでない場合も多いだろう。バグを埋めるのは一瞬だが、取り除くには 2 時間以上かかることもしばしばある。さらに C や C++ を用いる場合には、動作が保証されないコードをうっかり書いてしまった場合のエラーメッセージが不親切のため、原因追求に時間を要することもある。開発環境で利用可能な便利な道具に馴染んでおくと、原因追求の時間を減らせるかもしれない。特にプログラミングコンテストでは時間も計算機の利用も限られているので、チームで効率的な方法を見定めておくことが望ましい。

1.B.1 そもそもバグを入れない

逆説的だが、デバッグの時間を減らすためには、バグを入れないために時間をかけることが有効である。その一つは、良いとされているプログラミングスタイルを取り入れることである。様々な書籍があるので、自分にあったものを探すと良い。一部を紹介する。

- 変数を節約しない
悪い例: (点 (x_1, y_1) と (x_2, y_2) が直径の両端であるような円の面積を求めている)

```
C++
1 double area = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/2
2   * sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/2 * 3.1415;
```

改善の例:

```
C++
1 double radius = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/2;
2 double area = radius*radius*3.1415;
```

なぜ良いか:

- 人間に見やすい (x1 を x2 と間違えていないか確認する箇所が減る)
- タイプ/コピーペーストのミスがない
- 途中経過 (この場合は半径) を把握しやすい. (デバッガや printf で表示しやすい)

● 関数を使う

```
C++
1 double square(double x) { return x*x; }
2 double norm(double x1, double y1, double x2, double y2) {
3     return square(x2-x1)+ square(y2-y1);
4 }
5 double circle_area(double r) { return r*r*3.1415; }
```

● 定数に名前をつける

```
C++
1 const double pi = 3.1415;
2 const double pi = atan2(0.0,-1.0);
```

● 変数のスコープはなるべく短くする:

変数のスコープは短ければ短いほど良い. 関連して, 変数の再利用は避け, 一つの変数は一つの目的のみに使うことが良い.

```
C++
1 int i, j, k;
2 // この辺に j や k を使う関数があったとする
3 ...
4 int main() {
5     for (i=0; j<5; ++i) // ああっ!
6         cout << "Hello,_world" << endl;
7     ...
8 }
```

関数毎に必要な変数を宣言すると状況が大分改善する.

```
C++
1 int main() {
2     int i;
3     for (i=0; j<5; ++i) // コンパイルエラー
4         cout << "Hello,_world" << endl;
5 }
```

しかし、C++ や Java など最近の言語では、for 文の中でループに用いる変数を宣言できるので、こちらを推奨する。

```
C++
1 int main() {
2     for (int i=0; i<5; ++i)
3         cout << "Hello, world" << endl;
4 }
```

- ありがちな落とし穴をあらかじめ学び避ける

C 言語 FAQ http://www.kouno.jp/home/c_faq/ 特に 16 奇妙な問題

- コンパイラのメッセージを理解しておく:

– “if-parenth.cc:8:14: warning: suggest parentheses around assignment used as truth value”

```
C++
1 if (a = 1) return 1;
2 if (a =! 1) cout << "ok";
```

– “no return statement in function returning non-void [-Wreturn-type]”

```
C++
1 int add(int a, int b) {
2     a+b; // 正しくは return a+b;
3 }
```

1.B.2 それでも困ったことが起きたら

道具: assert

実行時のテストのために C, C++ では標準で assert マクロを利用可能である。assert 文は引数で与えられた条件式が、真であればなにもせず、偽の時にはエラーメッセージを表示して停止する機能を持つ。

■コード例 階乗の計算:

```
C++
1 int factorial(int n) {
2     if (n == 1)
3         return 1;
4     return n * factorial(n-1);
5 }
6 int main() {
7     cout << factorial(3) << endl; // 3*2*1 = 6 を出力
8     cout << factorial(-3) << endl; // 手が滑ってマイナスをいれてしまったら、止まらない
9 }
```

上記の関数は、引数 n が正の時のみ正しく動く。実行時に、引数 n が正であることを保証したい。そのためには、`cassert` ヘッダを include した上で、assert 文を加える。見て分かるように assert の括弧内に、保証したい内容を条件式で記述する。

```
C++
1 #include <cassert> // 追加
2 int factorial(int n) {
3     assert(n > 0); // 追加
```

```

4   if (n == 1)
5       return 1;
6   return n * factorial(n-1);
7 }

```

このようにして `factorial(-1)` 等呼び出すと、エラーを表示して止まる。

```

Assertion failed: (n > 0), function factorial, file factorial.cc, line 3. 1

```

このように、何らかの「前提」ののっとしてプログラムを書く場合は、そのことをソースコード中に「表明」しておくで見通しが良い。

`assert` は実行時のテストであるので、実行速度の低下を起こしうる。そのために、ソースコードを変更することなく `assert` を全て無効にする手法が容易されている。たとえば以下のように、`cassert` ヘッダを `include` する*前*に `NDEBUG` マクロを定義する

```

C++ 1 #ifndef NDEBUG
    2 # define NDEBUG
    3 #endif
    4 #include <cassert>

```

道具: `_GLIBCXX_DEBUG` (G++)

G++ の場合、`_GLIBCXX_DEBUG` を先頭で `define` しておくで、多少はミスを見つけてくれる。
(http://gcc.gnu.org/onlinedocs/libstdc++/manual/debug_mode_using.html#debug_mode_using.mode)

```

C++ 1 #define _GLIBCXX_DEBUG
    2 #include <vector>
    3 using namespace std;
    4 int main() {
    5     vector<int> a;
    6     a[0] = 3; // 長さ 0 の vector に代入する違反
    7 }

```

実行例: (単に `segmentation fault` するのではなく、`out-of-bounds` であることを教えてくれる)

```

/usr/include/c++/4.x/debug/vector:xxx:error: attempt to subscript container1
with out-of-bounds index 0, but container only holds 0 elements. 2

```

道具: `gdb`

以下のように手が滑って止まらない `for` 文を書いてしまったとする。

```

C++

```

```

1  int main() { // hello hello world と改行しながら繰り返すつもり
2      for (int i=0; i<10; ++i) {
3          for (int j=0; j<2; ++i)
4              cout << "hello_" << endl;
5          cout << "world" << endl;
6      }
7  }

```

gdb を用いる準備として、コンパイルオプションに `-g` を加える。

```
$ g++ -g -Wall filename.cc
```

1

実行するには、gdb にデバッグ対象のプログラム名を与えて起動し、gdb 内部で `run` とタイプする

```

$ gdb ./a.out
(gdb) (gdb が起動する)
(gdb) run # (通常の実行)
(gdb) run < sample-input.txt # (リダイレクションを使う場合)
# ... (プログラムが実行する)...
# ... (Ctrl-C をタイプするか, segmentation fault などで停止する)
(gdb) bt
(gdb) up // 何回か up して main に戻る
(gdb) up
#12 0x080486ed in main () at for.cc:6
6          cout << "hello_" << endl;
(gdb) list
1      #include <iostream>
2      using namespace std;
3      int main() {
4          for (int i=0; i<10; ++i) {
5              for (int j=0; j<2; ++i)
6                  cout << "hello_" << endl;
7                  cout << "world" << endl;
8              }
9          }
(gdb) p i
$1 = 18047
(gdb) p j
$2 = 0

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

主なコマンド:

- 関数の呼び出し関係の表示 `bt`
- 変数の値を表示: `p` 変数名
- 一つ上 (呼び出し元) に移動: `u`

- ソースコードの表示: list
- ステップ実行: n, s
- 再度実行: c
- gdb の終了: q

ソースコードの特定の場所に来た時に中断したり、変数の値が書き換わったら中断するようなこともできる。詳しくはマニュアル参照。

道具: valgrind

```
C++
1  int main() {
2      int p; // 初期化忘れ
3      printf("%d\n", p);
4  }
```

gdb を用いる時と同様に `-g` オプションをつけてコンパイルする。

```
$ g++ -g -Wall filename.cc
```

1

実行時は、valgrind コマンドに実行プログラムを与える。

```
$ valgrind ./a.out
Conditional jump or move depends on uninitialised value(s)
...
```

1

2

3

現状の eccs の imac 端末では、valgrind が動作しないが、ssh サーバにログインすれば使用可能である。
<http://www.ecc.u-tokyo.ac.jp/system/outside.html#ssh>

1.B.3 標本採集: 不具合の原因を突き止めたら

バグの原因を特定したら、標本化しておくで将来のデバッグ時間を減らすための資産として活用できる。「動いたからラッキー」として先に進んでしまうと、何も残らない。本筋とは離れるが、問題の制約を見落としたり、文章の意味を誤解したために詰まったなどの状況でも、誤読のパターンも採集しておくに役に立つだろう。

配列の境界

```
C++
1      int array[3];
2      printf("%d", array[3]); // array[2] まで
```

初期化していない変数

```
C++
1      int array[3];
2      int main() {
3          int a;
```

```

4     printf("%d", array[a]); // a が [0, 2] でなければ不可解な挙動に
5     }

```

return のない関数

```

C++ 1  int add(int a, int b) {
    2     a+b; // 正しくは return a+b;
    3 }
    4  int main() {
    5     int a=1,b=2;
    6     int c=add(a,b); // c の値は不定
    7 }

```

stack 領域溢れ

```

C++ 1  int main() {
    2     int a[100000000]; // global 変数に移した方がよい
    3 }

```

不正なポインタ

```

C++ 1  int *p;
    2  *p = 1;
    3
    4  char a[100];
    5  double *b = &a[1];
    6  *b = 1.0;

```

文字列に必要な容量: 最後には終端記号'\0' が必要

```

C++ 1  char a[3]="abc"; // 正しくは a[4] = "abc" もしくは a[] = "abc"
    2  printf("%s\n", a); // a[3] のままだと大変なことに

```

// A[i] (i の範囲は $[0, N-1]$) を逆順に表示しようとして

```

C++ 1  for (unsigned int i=N-1; i>0; ++i)
    2     cout << A[i] << endl;

```

// 整数を 2 つ読みたい

```

C++ 1  int a, b;
    2  scanf("%d_%d", &a, &b);

```