デバッギング – 個人的な体験と対策

和田 英一

幾何学と同様, プログラミングやデバッギングにも王道はないようだ. 最近は静かだが, Dijkstra 先生は, かつて「デバッグは虫のいることは証明するが, 虫がいないことは証明しない」といい, プログラムを書く段階から, 証明を入れていくことを勧めていた. これはいうのは簡単だが, 実行は困難である. 大体, 証明するためにはどういうプログラムを書こうとしているかがはっきりしていなければならない. それが出来るくらいならプログラムは書けたようなものである.

日本の計算機の歴史に登場するパラメトロン計算機 PC-1 の稼働開始は, 1958 年 3 月 26 日. PC-1 の総合テスト中, どうやら動いているらしいというので走らせたプログラム, neglect erase が私の作った, そして PC-1 で動いた, 最初のプログラムであった. イニシアルオーダーもアセンブリプログラムもなく, 2 進法で, 直接パンチしたのだが, 幸いなことに, 短かったので最初のプログラムは一度で動いた.

しかし, その後は多少大きいプログラムになると, やはり虫なしとはいかず, 「人は過つもの」としみじみ思う.

ここ 40 年弱の間にプログラミング, デバッギング環境は著しく変わった. 紙テープにプログラムをパンチし, 自分で計算機を操作した日々; カードにパンチし, センターに実行してもらった頃 (当時はプログラムを提出してから, 結果が戻ってくるまで, 長ければ数日かかった); ミニコンの時代; そして現在の端末の前, マルチウインドウを開き, オンラインマニュアルを眺めつつ仕事をする今日. 昔はしまったと思っても, 次のトライアルまで数日あるから, どうしても多様なデバッグ機構を始めから挿入し, 自衛する必要があった. いまでは自分の計算機で何回でもオンラインテストが出来るという環境になっている. しかし, 基本はまったく変わらないように思う.

40 年弱のプログラミング人生を振り返り, デバッグの体験とその対策につき, 所感を述べよう.

プログラムがうまく行かなくなる原因のほとんどは, 計算目的の取り違えとシステムの不慣れである. マニュアルや Info を良く読まずに取り掛かるのが自業自得の根源といえばそれまでだが, どこにも書いてない機能に悩まされることも少なくない.

虫とりで往生した Pascal での経験を話そう. これは ETH の Wirth さんのところから来た処理系だったと思うが，ある時, どうしてもわからない振舞をする. 実行中, 突然終了してしまうのだ. いろいろ苦労の末, やっと判明したのは, 私が `close` という名前の手続きを作り, それを呼んだのがいけなかったのである. この Pascal の処理系は, 入出力の制御用に `open/close` というシステムサブルーチンを持っており，ユーザーがそういう名前の関数 / 手続きを宣言していても，ユーザーが呼ぶと, システムサブルーチンを呼んでしまうのだ. これはマニュアル (Pascal Users' Report) にも書いてなく, 見つけるまで大変であった.

思えらく, Pascal のような構造のプログラム言語では, システムに `close` という手続き, 関数が仮にあったにせよ, ユーザーが `close` なるサブルーチンを宣言すれば, その方がローカルであり, まずそちらを探すのが筋であろう. そうならシステムがどんな名前を使っていても恐くないのだが, `close` に限り, そちらを優先するというのは Pascal の基本哲学に反する行為である. と, 一人でいきまいていても仕方ないか.

一旦正常に動作しているプログラムが, 環境の変化により突然おかしくなるというのも頭がいたい. このところ騒がれている 2000 年問題はその一種である.

unix 環境でプログラムする場合, 私がときどき振り回されるのは, `HOME` のような環境変数を使わず, ディレクトリ構造を直接プログラムに書いてしまうことだ. これはもちろん私が悪いのだが, 取り敢えず動くので安心していると, システム管理者がディレクトリ構造を変更したりすることで, プログラムが発狂する騒ぎが何回かあった. 似たようなのでは, 特定のメイルを

選択するのに, 電子メイルアドレスの綴りなどをキーにして string-search をかけていると, 綴りがいつの間にか変更になり, 機能を失うことになったりする.

対策としては, 慣れたプログラム言語, 慣れた操作システム以外には, 徒や疎かには手を出さないことである. 以前どこかに書いたことだが, その道の匠とは, 自分の使う道具にも通暁している人である. また必要以上に自己防衛的コーディングをすることだ.

最近, 私が使うプログラム言語は Lisp と Postscript である. C や Emacs Lisp は, プログラムを読むことはあるが, 自分からそれで書くことはほとんどない. プログラムに虫を入れない予防法の第一は, 上述のように熟知しているプログラム言語しか使わないことだからだ. また熟知している機能だけ使うようにする. 私の場合は Lisp (それも UtiLisp) でほとんど事足りるので, 99 パーセントは Lisp である. あとの 1 パーセントは図を書くため PostScript を使う. AWK も嫌いではないが手元にマニュアルが常にあるわけではないので, 冒険を避けて Lisp を使うことになる.

今年 (1996 年) のプログラミングシンポジウムで, 慶応の大駒さんは「日本の揺籃期のコンピュータのソフトウェア的復刻」の発表のとき, 1 年前のシンポジウム会場で, 私が PC-1 のソートのプログラムを書き, それに一つも誤りがなかったといわれ, 赤面したが, それには重要な秘密がある. PC-1 を作った高橋研では, ループやサブルーチンコールに標準のパターンがあったのである. それを呪文のように唱えながら書くと, つまらない失敗はしないことになっていた. つまり慣れたプログラムパターンも耐過失効果がある.

私は PC-1 の頃を除き, デバッガーを使わない. 当時は EDSAC のデバッガー (ポストモーテムなど) に感動し, PC-1 用のチェッキングルーチンを作り, 使っていた. その後使わないのは, しばらく使わなかったため, 「不慣れなシステムには手を出さぬポリシー」によるのだろうか. 手元にエディタがあるものだから, プリントルーチンの臨時挿入で済ませてしまう.

Lisp の利点は, プログラムの核心部分だけをまず書き, そこだけのテストができことである. テストデータも S 式だけだから用意するのが簡単である. Lisp の不具合は, プログラムがカッコだらけになり, プログラムの構造が判然としなくなることだ. そうなったら迷わず PrettyPrint する. (以前 MIT に滞在していたとき, MacLisp で PrettyPrint の仕方がわからず, 途方にくれていたら, 隣の端末から, 「grind したら?」と助け舟を出してくれたのは, bit9 月号で井田君が紹介している Bernie Greenberg であった. MacLisp では PrettyPrint のことを grind という.)

デバッグに手間取らないためには, プログラムをとにかく動くところまでもっていくことだ. 一旦動いたプログラムは目から鱗が落ちたように, 構造が見え始め, 仕事の能率が急上昇する. そのためにも核心から開始するのはいいことである.

この辺からは教科書にも載っていることだが, うまく動かないプログラムはもちろん, 正しいプログラムでも要チェックの場所にはプリントを入れておく. ときには最後の結果が偶然正しく, 完成かと思っても, 別のデータでは動かないことが起きる. とくに例外的なデータのチェックに必要である. チェック用のプリントは, 完成しても外さず条件文に作り, 不日入り用の時, 起動できるようにしておきたい.

同様にコメントもなるべく消さない. Lisp の Pretty Print は S 式しかプリントせず, セミコロンで始まるコメントは消えてしまうので, 特殊形式 comment を使おう.

どこかに似たプログラムがあるという場合, それをコピーし, 挿入, 再利用するのもいいが, つい修正したくなり, 元のプログラムと整合がとれなくなるので, サブルーチンにしたり, include するのがよい.

プログラミングもデバッギングでも, 成功の鍵は (息抜きも必要だが) 集中にあると思う. 私は本格的なプログラムを書く時は (笑うな !), 会議などはさぼることにしていた. あるいはなるべく講義などない時期まで作業を延ばすのである. まぁこれは原稿を書くのに, 〆切が近付いたことによるプレッシャーを利用し一気に書くのと似ているともいえる.

そういえば, 以前はデバッグに「身の上相談の技」があった. プログラムの虫が見つからぬ時, プログラムリストを見ながら友人に説明するのだ. するとあら不思議, 聞き手がさっぱり要領を得ないうちに, 話し手の側は突然ミスがわかり, 去っていってしまう. これは効果抜群で, 聞き手は案山子でもいいか, という笑い話もあったが, 生身の人間が必要みたいである. このところ, あまり見かけなくなったのは如何なるわけか.

(わだ えいいち (株) 富士通研究所)