Program Testing and Correctness

An error-free program is almost never obtained on the first try. Programming is just too complicated a process for that to happen often. For a careful and competent programmer the probability of each possible error is very small, but there are so many opportunities that the probability of avoiding all errors is also very small. Prudence demands that we assume each new program is incorrect until we can demonstrate otherwise. We must accept the fact that testing is an integral part of the programming process - and that a program is not really "finished" until we have demonstrated its correctness.

The magnitude of the testing effort required for large problems might surprise you. Often at least half the manpower, cost and elapsed time of the total programming process is consumed in testing. In spite of this, the results are often not entirely satisfactory. Consequently, computers are not held in high public esteem for their reliability, when in fact they are exceedingly reliable machines - handicapped by inadequate programs.

Few beginning programmers take the trouble to learn to test programs efficiently. To plan testing in advance seems to be admitting inadequacy, so most beginners assume an ostrich-like, head-in-the-sand attitude. When it is obvious that something is wrong, they test in a haphazard manner. But testing does not have to be as time-consuming and unpleasant as many make it seem; effective testing is a skill that must be studied and learned.

The Meaning of Correctness

"Program correctness" is not easily defined. The programmer and user of a program may interpret "correctness" quite differently and hence have different expectations of program performance. Various interpretations of correctness are listed below in order of increasing difficulty of achievement:

1. The program contains no syntax errors that can be detected during translation by the language processor.

2. The program contains no errors, either of syntax or invalid operation, that can be automatically detected during translation or execution of the program.

3. There exists some set of test data for which the program will yield the correct answer.

4. For a typical (reasonable or random) set of test data the program will yield the correct answer.

5. For deliberately difficult sets of test data the program will yield the correct answers.

6. For all possible sets of data that are valid with respect to the problem specification, the program yields the correct answers.

7.   For all possible sets of valid test data, and for all likely conditions   of erroneous input, the program gives a correct (or at least reasonable) answer.

8.   For all possible input, the program gives correct or reasonable answers.

In the early stages of your programming experience you will feel harassed by error messages and feel a sense of relief and accomplishment when none appear. Later you will understand that the real work of testing starts after the error messages have been eliminated. The absence of error messages is only a necessary and not a sufficient condition for reasonable correctness.

Some programmers never mature beyond level 3 in their attitude toward correctness. Considering, however, the higher levels (say 4, 5 or 6), it is clear that satisfactory performance on any single set of test data is not sufficient grounds for an assertion of correctness. Moreover, failure on a single test is sufficient to demonstrate that the program is not correct. No matter how many tests the program may have passed successfully, just one test on which it fails is enough to show that it is not correct. This is not inherently a democractic process, and a program that works "most of the time" is a dangerous tool.

From the user's point of view a reasonable definition of correctness is certainly not less than level 6. Level 7 is better and level 8 is what he would really like. The programmer may maintain that a literal interpretation of problem specifications cannot demand more than level 6, while the user will maintain that certain implied requirements do not have to be explicitly stated. However, the primary responsibility rests with the programmer. A program is incorrect if it does not serve the user's purpose.

In summary, the situation is the following. The user would like to have level 8 correctness - but this is often unrealistic. Level 7 is a reasonable compromise, which is obviously going to lead to arguments since it leaves critical questions open to varying interpretations. The programmer's dilemma is that level 5 is the highest that can be achieved by purely empirical means - by running the program on test cases - so he must thoughtfully design test cases that permit a plausible assertion that level 6 has been achieved. To achieve level 7 the programmer must know enough about the intended use of the program to estimate what errors are likely to be encountered, and what response is appropriate.

The Deadly Sins of Programming

Debugging the program instead of the algorithm

    a)   Failing to use incremental program development methods
    b)   Attempting to validate more imperative statements than is
            intellectually feasible
    c)   Exceeding one page for a logical module

Insufficient, pre-prepared documentation

    a)   Insufficient comments in the program
    b)   Comments that explain the program rather than the methods
    c)   Lack of coordination between the documentation, the design
            and the program (i.e. lack of common keys or keywords)

Using an inappropriate language

    a)   Not conforming to the Standard
    b)   Choosing the wrong language for the particular solution

---

Poor use of identifiers

    a)   Using global (common) variables
    b)   Failing to protect non-used non-local variables
    c)   Relying on side effects
    d)   Not clearly distinguishing between input and output variables
            in a module

Poor Use of Functions

    a)   Using internal functions when external functions are
            available
    b)   Using the same parameters for both input and output
    c)   Using constants as arguments in a function invocation

Relying on default conditions

    a)   Insufficient explicit declarations
    b)   Assuming the existence of initial values
    c)   Using mixed mode expressions with default conversions

Programming Correctness & Sins A07. Sins   7

Insufficient verification assertions at critical points

    a)   Failing to provide sufficient redundancy for validation of
            program properties
    b)   Relying on the correctness of other programs or run-time
            routines

Not paying attention to input operations

    a)   Failing to echo the input data into the output file
    b)   Destroying the original data values
    c)   No domain checks on input data values
    d)   No recovery procedures for invalid input data

Not believing that output matters

    a)   Not providing a banner/heading giving program
            identification, author and version data
    b)   Too much output
    c)   Insufficient data identification
    d)   Giving numbers rather than diagrams

Computer Science Dept Va Tech Aug., 2001   Intro Data Structures & SE   ©1995-2001 Barnette ND, McQuain WD