Correctness

A product is correct if it satisfies its output specifications when operated under permitted conditions.

Correctness of specifications

Incorrect specification for a sort (Figure 6.1):
- Input spec: p: array of n integers, n>0
- Output spec: q: array of n integers such that q[0] <= q[1] <= ... <= q[n-1]

Function `trickSort` which satisfies this specification (Figure 6.2):
```c
void trickSort (int p[], int q[])
{
    for (int i = 0; i < n; i++)
        q[i] = 0;
}
```

Correctness of specifications (2)

Corrected specification for the sort (Figure 6.3):
- Input spec: p: array of n integers, n>0
- Output spec:
  - q: array of n integers such that q[0] <= q[1] <= ... <= q[n-1]
  - The elements of array q are a permutation of the elements of array p, which are unchanged.

Correctness

Technically, correctness is not necessary:
- Example: C++ compiler that prints out spurious error messages

It is not sufficient:
- Example: `trickSort`

Testing versus Correctness Proofs

A correctness proof is an alternative to execution-based testing
It is a mathematical technique

Example of a Correctness Proof

The code segment to be proven correct (Figure 6.4): s contains the sum of the elements of y
```c
int k, s;
int y[n];
k = 0;
s = 0;
while (k < n)
{
    s = s + y[k];
    k = k + 1;
}
```
Example of a Correctness Proof (2)

- A flowchart equivalent of the code segment (Figure 6.5) – note that the loop condition has been inverted into a termination condition.

Example of a Correctness Proof (3)

- Add
  - Input specification – assumptions
  - Output specification – the result
  - Loop invariant – holds at the point before the loop condition for any number of iterations
  - Assertions – between statements, describe the effect of each statement

(See Figure 6.6 on next slide)

Annotated Flowchart

- Input specification – assumptions
- Output specification – the result
- Loop invariant – holds at the point before the loop condition for any number of iterations
- Assertions – between statements, describe the effect of each statement

(See Figure 6.6 on next slide)

Example of a Correctness Proof (4)

- An informal proof using induction
  - Show the correctness of each assertion
  - Follows that the output spec is correct

- A is the input spec and is assumed to be true: n \geq 1
- B and C follow from the definition of assignment: k = 0, s = 0

Example of a Correctness Proof (5)

- Base case at D: First time through the loop, k = 0 (by B) and n \geq 1 (by A)
  - k \leq n as required.
- Since k = 0, k-1 = -1, which is outside the range of the array of elements, so sum is empty and s = 0 (by C)
  - s = 0 as required.
- Therefore the loop invariant at D is true the first time the loop is entered.
Example of a Correctness Proof (6)

- Inductive hypothesis at D: Assume D holds for some $k_0$, $0 \leq k_0 \leq n$. That is, $k_0 \leq n$ and $s = y[0] + y[1] + \ldots + y[k_0-1]$
- At the loop condition, if $k_0 \geq n$, then because $k_0 \leq n$ (by hypothesis), then $k_0 = n$ implying $k_0 = n$ and $s = y[0] + y[1] + \ldots + y[n-1]$
- Thus proving the output spec is true at the termination of the loop

Example of a Correctness Proof (7)

- At the loop condition, if $k_0 \geq n$ fails, then at E, we know that $k_0 < n$ and $s = y[0] + y[1] + \ldots + y[k_0-1]$
- At F, the assignment statement has added $y[k_0]$ to the sum, so $k_0 < n$ and $s = y[0] + y[1] + \ldots + y[k_0-1] + y[k_0]$
- At G, since $k_0 < n$, the assignment statement adding 1 to $k_0$ means at most $k_0 < n$ and the index of the last term is now $k_0-1$, so $k_0 < n$ and $s = y[0] + y[1] + \ldots + y[k_0-1]$

Example of a Correctness Proof (8)

- The assertion at G is identical to assertion assumed at D.
- D is topologically identical to G, so if the assertion holds at D for $k = k_0$, then it again will hold for $k = k_0 + 1$
- Already shown that D holds for $k = 0$, so it follows that the loop invariant holds for all values of $k$, $0 \leq k \leq n$. 
Example of a Correctness Proof (9)

- Lastly, need to prove that the loop terminates
- At C, k = 0. Each iteration of the loop increases k by 1. Eventually, k must reach the value n, at which time the loop exits and the value of s is given by the loop invariant as shown.

Example of a Correctness Proof (10)

- Summary: Given input spec,
- proved loop invariant holds for any number of iterations, and
- proved the loop terminates, and
- when it does, k and s have values that satisfy the output spec,
- thus the code fragment is correct.

Correctness Proof Mini Case Study

- Dijkstra (1972):
  - "The programmer should let the program proof and program grow hand in hand"
- "Naur text-processing problem" (1969)

Naur Text-Processing Problem

- Given a text consisting of words separated by a blank or by newline characters, convert it to line-by-line form in accordance with the following rules:
- Line breaks must be made only where the given text contains a blank or newline
- Each line is filled as far as possible, as long as
- No line will contain more than maxpos characters

Correctness Proof Mini Case Study (2)

- Episode 1
  - Naur constructed a 25-line procedure
  - He informally proved its correctness
- Episode 2
  - 1970 — Reviewer in Computing Reviews
    - The first word of the first line is preceded by a blank unless the first word is exactly maxpos characters long

Correctness Proof Mini Case Study (3)

- Episode 3
  - 1971 — London finds 3 more faults, including:
    - The procedure does not terminate unless a word longer than maxpos characters is encountered
    - London corrects and proves formally
- Episode 4
  - 1975 — Goodenough and Gerhart find three further faults, including:
    - The last word will not be output unless it is followed by a blank or newline
Correctness Proof Mini Case Study (4)

- Four of seven discovered faults detected would have been uncovered had the procedure been tested on the illustrated data in Naur’s original paper!
- Lesson: Even if a product has been proven correct, it must still be tested!!

Correctness Proofs and Software Engineering

- Three myths of correctness proving:
  - Software engineers do not have enough mathematics for proofs
    - Most computer science majors either know or can learn the mathematics needed for proofs
  - Proving is too expensive to be practical
    - Economic viability is determined from cost–benefit analysis
  - Proving is too hard
    - Many nontrivial products have been successfully proven
    - Tools like theorem provers can assist us

Difficulties with Correctness Proving

- Can we trust a theorem prover? (Figure 6.7)
  ```c
  void TheoremProver ()
  {
  print "This product is correct";
  }
  ```
- How do we find input–output specifications, loop invariants?
- What if the specifications are wrong?
- We can never be sure that specifications or a verification system are correct

Correctness Proofs and Software Engineering (2)

- Correctness proofs are a vital software engineering tool, where appropriate:
  - When human lives are at stake
  - When indicated by cost–benefit analysis
  - When the risk of not proving is too great
- Also, informal proofs can improve the quality of the product
  - Use the `assert` statement

Software Project

- Every individual is responsible for his own project
- May collaborate with anyone in the class: team members are expected to do so at least for the interface analysis
- May not collaborate with anyone else, but may ask for assistance from instructor or Dr. Morse or other EECS faculty member
- Internet resources may be consulted for background material; no code may be copied with or without permission

Software Project (2)

- All project artifacts are to be archived on csprojects.evansville.edu
- Written documents are to be separate pages of the project website
- No classes next week; Monday will be HTML tutorial, if needed; everyone must sign up for at least one appointment with instructor by Wednesday
- Presentations on Friday after Fall Break; generally no class on Fridays after that