Peach 3

CS Colloquium at TU/e
29 November 2007

Tom Verhoeff

Eindhoven University of Technology
Department of Mathematics & Computer Science
Software Engineering & Technology

`http://www.win.tue.nl/~wstomv/`

Programming Education in the Past

- The year 2000: *over 150 first-year students in CS*
- Practical programming course: done on paper
- Five independent groups
- Process:
  1. collect programs
  2. evaluate programs
  3. administrate results

Programming Contests

- ACM ICPC: 1988–1990 European Finals, 1999 World Finals
- IOI: IOI'95 in Eindhoven, IOI'96–'07
- NIO
- Process:
  1. collect programs
  2. evaluate programs *automatically*
  3. administrate results *automatically*

Peach/vs

P rogramming **E** ducation **A** nd **C** ontest **H** osting verification system

Developed in the summer of 2001 by Erik Scheffers

First used in September 2001 for Programming 0

September 2007 (version 3): **Peach**

Feature Overview

- Web-based client-server system: `peach.win.tue.nl`
- Various user categories: student, grader, teacher, admin, observer
- Collect, store, evaluate, compare submitted work and results
- Supports multiple courses, with groups, over multiple years
- Evaluation configurable per assignment
- Supports multiple (programming) languages

Peach As Communication Aid

Who should do/did what when with what result?

- Register participants
- Provide assignments (configurable open/close period)
- Define and enforce deadlines and number of attempts
- Collect and store the work (web-viewable by submitter and staff)
- Provide feedback (automatic and/or manual)
- Administrate results

What Peach Is (Not)

Peach is not intended as a full-blown generic

- student administration system
- course management system (cf. Moodle.org)
- web content management system (WCMS)
- workflow management system
- program development environment (IDE)
- configuration/version management system (cf. Subversion)

Student View

- Register once (usercode/password; future: central login portal)
- Join course/group (once per course)
- Read assignment
- Submit work, check acceptance
- Read feedback/result

Repeat where necessary

Further support to interpret feedback (error messages) is desirable

Grader View

- View submission: files, checks
- Provide feedback
- Determine result

Grading scheme/criteria currently not stored in Peach

Teacher View

- Prepare assignments

  Can be developed stand-alone as a *Peach package*
- Make assignments (un)available
- Set deadlines and limits
- Inspect results
- View statistics

Further support for assignment preparation desirable

Manual Evaluation

Grading scheme covering

- Layout
- Comments, (formal) annotation
- Naming
- Definitions
- Modularization
- Coding patterns

Automated support imaginable

Automatic Evaluation

Typically, for submitted *programs*:

1. Preprocess (e.g. max. program length, language, TODOs)
2. Compile (possibly together with test framework)
3. Execute (with defined environment/input)
4. Check behavior/output
5. Determine score, repeat 2–5 as desired

Generally, can handle anything supported by analysis tools under Linux:
models, specifications, grammars, proofs, test cases, . . .

Assignment Preparation

- Descriptive text: problem, input, output, constraints, hints, ...
- Allowed programming language(s)
- What needs to be submitted, other preprocessing checks
- Compiler options, libraries, ...
- Run-time limits, environment
- Test cases: input, expected output or output checker
- Scoring function; accept/reject criteria
- Good and bad programs, to test the assignment configuration

Example Assignment: Candy (2IP05)

K kids together receive C candies. Your program must determine whether it is possible to divide all candies fairly , and if so, how many candies each kid receives. This is a integer Q such that $C = K * Q$.

Input : The first line contains two integers K and C, separated by one space, with $0 \le K, C < 10^9$.

Output : The first line must be the string '`Yes`' if it is possible to divide all candies fairly, and '`No`' otherwise. If it is possible, then there is a second line, containing integer Q (number of candies each kid receives), with $0 \le Q < 10^9$. *If there are multiple answers, then it does not matter which answer your program writes.*

Example:

input
3 15

output
Yes 5

Checker Issues

- Output format: whitespace, newlines, upper/lower case
- If input uniquely determines output: expected output
  - Can be generated by known-correct program
- If input does *not* uniquely determine output: checker program
  - Reads input, program's output, optional additional data
  - Verifies specified I/O relationship
  - Must be robust: program's output can be garbage
- Possibly no input/output, but *provide a service* or *use a service*
- GUI/web applications, distributed/parallel programs (not yet done)

Checker Example: Dice Game (2IP05)

- Players 1–4 each roll two regular dice (outcomes 1..6 + 1..6)

  Player 5 rolls a dodecahedron (outcomes 1..12)

  Unique maximum value wins, otherwise no winner

  Is Player 5 *better* off or *worse* off than the others? How much?
- Assignment: Randomly simulate multiple rounds

  Various programming errors possible
- Checker must test *statistical hypothesis*

Checker Example: Energy Pills (2IP05)

Example:

input	output
3 4	138
0 1 30 0	0* 1 30 0
2 10 1 3	2* 10* 1 3
4 20 7 99	4 20* 7* 99*

- Consider monotonic paths from *upper left* to *lower right* corner
- Maximize the path sum (total energy)
- Evaluation must “catch” greedy algorithms and other errors

Checker Example: Bounded Queue (2IP05)

- Assignment: Implement a bounded queue ADT, given its contract

  constructor *Create*(...);
  function *Count*;
  function *IsEmpty*;
  function *IsFull*;
  function *First*;
  procedure *Put*(...);
  procedure *RemFirst*;

  When precondition not satisfied, an *exception* must be raised

- Evaluation must verify *functionality* and *robustness*

  Done without using a known-correct bounded queue

Checker Example: Binary Search Test Driver (2IP10)

- Assignment: Write a test driver for a binary search routine

```
procedure Find ( const s: List; const x: Entry;
                 var found: Boolean; var pos: Index );
  { pre:  s is ascending (duplicates allowed)
    post: found == (E i : 0 <= i < s.len : s.item[i] = x)  /\
          found ==> 0 <= pos < s.len  /\  s.item[pos] = x }
```

- Evaluation based on coverage
- Compile with *instrumented* version of `Find`

  Log each call; check precondition

  Evaluate with various good and bad implementations of `Find`

  Check distribution of parameter values over all `Find` calls

Plagiarism

- *Correlate submissions* (same assignment, multiple years)
- No search on internet (only in its own database)
- False positives, false negatives, assignment dependence
- Subsequent investigation is time consuming
- Further tool support desirable
- Cannot detect that someone else did the work
- Alternative: give assignments *under exam constraints*

Availability

- Production environment on our own SET server
- Installable on other platforms (requirements . . .)
- Open source license (except: authorization/comparison modules)
- Also used in Finland, India

Usage Statistics

Until	# Courses + Contests	# Active Users	# Submissions
Aug. 2002	7	174	1808
Aug. 2003	18	483	5990
Aug. 2004	28	727	10509
Aug. 2005	38	937	14327
Aug. 2006	50	1158	18622
Aug. 2007	65	1673	24313
Dec. 2007*	68	1760	25747
Dec. 2007†	6	233	2917

*Peach 2
†Peach 3

Conclusions

- Peach is a success:
  1. Automated evaluation strictly enforces functional quality
  2. Support for manual evaluation of other qualities
  3. Uniform administration, easily accessible by all involved
  4. Enforced deadlines, individually extendible
  5. Plagiarism detection
- But automatic evaluation comes at a cost
- Future: interface to secure exam software, central login portal

Questions?

?