ECE 462
Object-Oriented Programming
using C++ and Java

Testing

Yung-Hsiang Lu
yunglu@purdue.edu
Unreachable Code

If a, b, and c are zeros or positive numbers
\[(a + c) < b \Rightarrow a > b \text{ is impossible}\]
\[\Rightarrow \text{problem in the logic?}\]

if ((x <= 0) && (x >= width)) // hit left or right wall
{
    // width > 0
    vx = -vx; // change direction
}
Testing Strategy

- Testing is one, not the only one, step to ensure quality.
- Before writing code, think about how to test it.
- Do not be surprised that you write more code for testing than for the project.
- Danger of using testing to ensure quality: you usually test what you suspect. The program usually breaks at places where you are confident.
- Sometimes, reading code line-by-line can find and fix problems faster than writing testing code, especially for multi-thread programs.
## Design and Testing

<table>
<thead>
<tr>
<th></th>
<th>Test Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>unit test</td>
</tr>
<tr>
<td>design</td>
<td>integration test</td>
</tr>
<tr>
<td>requirements</td>
<td>validation test</td>
</tr>
<tr>
<td>system engineering</td>
<td>system testing</td>
</tr>
</tbody>
</table>

- Importance of unit testing: well-designed software should allow only **limited visibility** (encapsulation) for better consistency. Hence, testing from outside is difficult.
- Build software in **layers**. A lower layer should be fully tested before building a higher layer.
Test Coverage

• How much code is exercised in a test? How many possible paths are traversed?
• Many tools exist for reporting code coverage.
• Low coverage: not fully tested ⇒ bad test
• High coverage: can hardly test each possible path ⇒ quality unclear
• Testing discover many problems? good or bad?
• "dead code": code that is impossible to reach, usually indicates design or coding errors

```c
return 0;
x ++;
}
```
Every node has been visited (100% test coverage) but C2→C6 has not been tested.
Quantify Testing Quality

- coverage: (% code and % paths) of the test
- efficiency: evenly distribute? time to cover 99%?
- progression: % new code tested
- discovery rate: % bugs found for every line of test code
- configurability: selections of features to test
- ratio: how much testing code needed to test actual code
- expandability: amount of efforts needed to test new features
- degree of automation: can it be fully automated, semi-automated, or complete manual?
Testing Steps

• unit testing, integration testing, regression
• unit testing:
  – individual components
  – often conducted by the developer
  – often using dedicated testing code to create input data, exercise the components
  – often traced by single steps
  – check boundary conditions and error handling
  – check interface correctness and responses to incorrect inputs
Unit Testing

- examine the performance
- should be performed before "cvs commit"
- should be put into the repository
- should be configurable for related components
- require careful planning **in advance**
- driver: code to call the component, stub: code to be called by the component. both are overhead
Integration Testing

• Interface incompatibility is often the reason software breaks. Incompatibility ≠ compiler error
• types of interface errors, even passing compilation, e.g.
  – wrong types (object of derived class or base class)
  – wrong assumptions, for example
    • who is responsible for allocating or releasing memory
    • who may modify the data, especially global variables
    • sorted or nearly sorted? wrong result or wasting time
  – wrong timing assumption for real-time software
• **incremental** integration: add one component (e.g. class) each time
• may still use drivers and stubs (how do you know they are correct?)
Top-Down Integration

- using control flow to determine the integration order
- starting from the main component ("main" in C/C++) as the driver
- integrate callees (replace stubs) of the main component
- depth-first integration: C1, C2, C5, C7
- breadth-first integration: C1, C2, C3, C4
- after one successful test, replace a stub by the real component
- regression test (later) to ensure tested components still work
Challenges in Top-Down Testing

- **control flow** and **call graph** are not downward only or acyclic.
- Many functionalities cannot be tested before the leaf components are built, e.g. C1 needs the data (or objects) generated in C7 to test C3.
- Depth-first or breath-first only may not represent normal execution paths.
Bottom-Up Integration

1. start from individual components (such as classes)
2. put several components together, use a driver to test them
3. replace the driver by a real component
4. repeat step 2-3

• difficulties:
  – which components to integrate first? They must have a common driver.
  – control may not be upward only or acyclic
  – required data (or objects) may be generated from a component that has not been integrated
Regression Testing

- re-test what has been tested after new components are integrated into the project
- (same) test after errors are corrected
- **expandable** as more components are added
- **configurable** so that new components can be exercised more
- should be automated as much as possible (consider using cron jobs)
- Most important / frequently used features should be test more thoroughly.
Test Documentation

• Testing should be planned and documented.
  – test plan: what to test, when to test, who runs the test, how to run the test, what data to use …
  – testing with integration: how components are integrated, regression testing procedure
  – procedure to test: manual, automated, or semi-automated? conditions, tools, special hardware …
  – test result analysis: what to expect, how to diagnose
  – test result management: providing a trace of integration and testing
  – re-test procedure after correction
Hypothesis-Test Debugging

• Most software developers take "hypothesis-testing" approach for debugging:
  – guess what is the cause ➡️ usually the hardest part
  – modify the code
  – run some tests
  – analyze results, if not fixed, guess another place
Time-Sensitive or Massive Data

• Single-step code is not always the best way to debug.
• Some programs cannot be single-stepped:
  – time-sensitive, interacting with the physical world. It does not wait for the program.
  – massive amount of data, too many steps. How do you single-step an image with 3 million pixels?
• Create increasingly complex and realistic testing data: smaller images, simpler images, single color, checkerboard …
• Detect error conditions before proceeding. **Always** check the return value of a system or library call, such as connect, read, write, **new**, fork …
Time-Sensitive or Massive Data

- Detect and handle errors before they propagate.
- Generate execution logs for post-execution analysis. Be careful about the impact on timing.
- Controversy of "assert": assert (something must be true);
  ```
  assert (x > 0);
  ```
  Program stops immediately if the condition fails
  ⇒ errors do not propagate.
  ⇒ users cannot recover anything, especially lost data.
Layered Structure

• Structure the program so that each file can be assigned a unique layer number.
• Layer 0: files from language, such as iostream
• Layer 1: library files, such as Qt's classes
• Layer 2: common files used in multiple projects in your organization
• Layer 3: stable files used for months
• Layer 4: recently developed and test files
• Layer 5: unstable files
• Layer n: depends on files in layer 0, 1, 2, …, n-1
File Layers

A file is assigned layer n if it depends on only files in layer 0, 1, 2, ..., n-1
Why to Layer Files / Classes

- A file with a lower layer number should be more stable and have a higher degree of correctness.
- Strictly layered structure allows unit testing a recently developed module in the program.
- Layering indicates the precedence of development. If a module is a foundation for some other modules, this module should be placed (physically) in a file that has a lower layer number.
- Cyclic dependence often suggests flaws in logical design.
Test Exceptional Cases

• Error / exception handling code, by definition, doesn’t execute often. In “normal” conditions, the code is not tested.

• Testing exception handling code requires a well-designed plan to create the conditions for triggering the code’s execution.

• Be careful about how to create the conditions and danger of “simulated disasters.”

ECE 462
Object-Oriented Programming
using C++ and Java

Automatic Testing

Yung-Hsiang Lu
yunglu@purdue.edu
Periodic Testing

- schedule periodic testing, for example, at 2AM everyday
- check out code from repository
- compile and link to create executable
- execute the program with known inputs
- direct the output to a (or several) file
- compare the output with expected output
- report test results
- inform project manager if serious errors are discovered
Periodic Tasks using cron
ls /etc/cron.daily/
00-makewhatis.cron*
0anacron*
logrotate*
[(qstruct04) ~/ ]
more /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly

minute hour day-of-month month day-of-week command
Cron Tasks

17 8 * * * echo "daily at 8:17 am"
17 20 * * * echo "daily at 8:17 pm"
00 4 * * 0 echo "at 4 am every Sunday"
01 * 19 07 * echo "hourly on the 19th of July"
1,15,30,45 * * * * echo "every 15 minutes"
Crontab

- `crontab -l`: list current cron tasks
- `crontab -r`: remove current cron tasks
- `crontab -e`: edit cron tasks using the editor specified by environment setting `EDITOR`

- `1,15,30,45 * * * * echo `date` >> LogFile`
[\texttt{\textbackslash (qstruct04) \textbackslash ~/ \textbackslash ] \texttt{crontab -e}]

1,15,30,45 * * * * echo `date` >> /home/shay/a/ee462b3\0/cronlog

-uu-::---F1 crontab.xxxx9ygsbl (Fundamental)---L2--
[(qstruct04) ~/] crontab -l
1,15,30,45 * * * * echo `date` >> /home/shay/a/ee462b30/cronlog

[(qstruct04) ~/]
<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Zone</th>
<th>Year</th>
<th>CRON Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon Jul 7</td>
<td>10:45:01</td>
<td>EDT</td>
<td>2008</td>
<td></td>
</tr>
<tr>
<td>Mon Jul 7</td>
<td>11:01:01</td>
<td>EDT</td>
<td>2008</td>
<td></td>
</tr>
<tr>
<td>Mon Jul 7</td>
<td>11:15:01</td>
<td>EDT</td>
<td>2008</td>
<td></td>
</tr>
<tr>
<td>Mon Jul 7</td>
<td>11:30:01</td>
<td>EDT</td>
<td>2008</td>
<td></td>
</tr>
</tbody>
</table>
0 2 * * * /path/to/your/crontask >> /path/to/your/logfile

#!/usr/bin/sh
CVSROOT=/your/CVS/root
CODEHOME=/here/to/store/the/checkout
cvs -d $CVSROOT co -d $CODEHOME program
echo `date`
#!/usr/bin/csh

set PROJPATH=/path/to/store/the/program

set CVSROOT=/path/to/CVSROOT

echo CVSROOT $CVSROOT
cvs -d $CVSROOT co project
cd project

set LOGFILE=testlog

make > $LOGFILE

./program arg1... >> $LOGFILE

analyze $LOGFILE
g++ -fprofile-arcs -ftest-coverage
int main(int argc, char * argv[]) {
    srand(time(NULL));
    for (int i = 0; i < 6; i++) {
        double a = 0.001 * (rand() % 1000);
        double b = 0.001 * (rand() % 1000);
        double c = 0.001 * (rand() % 1000);
        double d = 0.001 * (rand() % 1000);
        if (a > b) {
            cout << "a > b " << a << " " << b << endl;
        } else {
            cout << "a <= b " << a << " " << b << endl;
        }
        if (c > d) {
            cout << "c > d " << c << " " << d << endl;
        } else {
            cout << "c <= d " << c << " " << d << endl;
        }
        if ((a + c) < b) {
            if (a > b) {
                cout << "(a + c) < b and a > b " << endl;
            } else {
                cout << "(a + c) < b and a <= b " << endl;
            }
        }
    }
}
[`(qstruct04) ~/lecturecode/1110/ ] ls
coverage.cpp
[`(qstruct04) ~/lecturecode/1110/ ] g++ -fprofile-arcs -ftest-coverage coverage.cpp
[`(qstruct04) ~/lecturecode/1110/ ] ./a.out
a <= b 0.273 0.405
c > d 0.325 0.222
a > b 0.39 0.073
c > d 0.368 0.248
a <= b 0.352 0.534
c <= d 0.631 0.819
a > b 0.439 0.12
c <= d 0.1 0.766
a <= b 0.67 0.834
c <= d 0.748 0.833
a <= b 0.256 0.514
c <= d 0.005 0.937
(a + c) < b and a <= b
[`(qstruct04) ~/lecturecode/1110/ ] ls
a.out* coverage.cpp coverage.gcda coverage.gcno
[`(qstruct04) ~/lecturecode/1110/ ]
YHL C++ Test Coverage 17

```
[[qstruct04] ~/lecturecode/1110/ ] gcov coverage.cpp
File `/usr/lib/gcc/x86_64-redhat-linux/3.4.6/.../.../.../include/c++/3.4.6/bits/locale_facets.tcc'
Lines executed: 0.00% of 11
`/usr/lib/gcc/x86_64-redhat-linux/3.4.6/.../.../.../include/c++/3.4.6/bits/locale_facets.tcc:creating `locale_facets.tcc.gcov`

File `coverage.cpp'
Lines executed: 94.74% of 19
coverage.cpp: creating `coverage.cpp.gcov'

File `/usr/lib/gcc/x86_64-redhat-linux/3.4.6/.../.../.../include/c++/3.4.6/bits/stl_algobase.h'
Lines executed: 0.00% of 4
`/usr/lib/gcc/x86_64-redhat-linux/3.4.6/.../.../.../include/c++/3.4.6/bits/stl_algobase.h:creating `stl_algobase.h.gcov`

File `/usr/lib/gcc/x86_64-redhat-linux/3.4.6/.../.../.../include/c++/3.4.6/iostream'
Lines executed: 100.00% of 1
`/usr/lib/gcc/x86_64-redhat-linux/3.4.6/.../.../.../include/c++/3.4.6/iostream:creating `iostream.gcov'
```
ls
a.out* coverage.gcda locale_facets.tcc.gcov
coverage.cpp coverage.gcno stl_algo_base.h.gcov
coverage.cpp.gcov iostream.gcov

double a = 0.001 * (rand() % 1000);
double b = 0.001 * (rand() % 1000);
double c = 0.001 * (rand() % 1000);
double d = 0.001 * (rand() % 1000);

if (a > b) {
    cout << "a > b " << a << " " << b <<
}
else {
    cout << "a <= b " << a << " " << b <<
}

if (c > d) {
    cout << "c > d " << c << " " << d <<
}
else {
    cout << "c <= d " << c << " " << d <<
}

if ((a + c) < b) {
    cout << "(a + c) < b and a > b " <<
}
else {
    cout << "(a + c) < b and a <= b " <<
}
ECE 462
Object-Oriented Programming
using C++ and Java

3-Dimensional Graphics

Yung-Hsiang Lu
yunglu@purdue.edu
HQ: K1-84 reported in sector 16,15
*VF_2 :Entering the pattern.
*VF_2 :Change to trail formation.
Foreshortening
3D Graphics in C++ and Java

- libraries for 3D graphics:
  - C++
Show 3D on 2D Screen

- project 3D on 2D: farther objects look smaller
- calculate depth: a near non-transparent object blocks the visibility of a farther object
- show lighting and shading
Calculate Projection on Screen

\[ \frac{h_1}{x_1} = \frac{h_2}{x_2} \Rightarrow h_1 = \frac{x_1 h_2}{x_2} \]

A farther object (larger \(x_2\)) looks shorter (smaller \(h_1\)).
Projection

3D? It is a projection of 3D objects on a 2D screen.

If a vector is (x, y, z), its 2D projection on the X-Y plane is (x, y)
Projections

(a,0,c)  (a,b,c)

(a,b,0)

YHL 3-D Graphics 13
Why is 3D More Complex

The image on the screen depends on many factors

- location of the viewer
- locations of the objects, including their relative depths to the viewer
- shape of the objects
- locations of the lights
- surface materials of the objects, reflective (such as polished metal) or absorptive (such as dark-color cloth)
- motion
- ...

YHL 3-D Graphics

14
Geometric Representations

• Each point is represented by a 3-D coordinate
  \[(x, y, z)\]
• A vector is also represented by a 3-D coordinate
  \[v = (x, y, z)\]
• A line is represented by one point of the line and the direction of the line
  \[(x_0, y_0, z_0) + a (x_d, y_d, z_d), -\infty < a < \infty\]
• A plane is represented by one point of the plane and two vectors. The two vectors must not be parallel.
  \[(x_0, y_0, z_0) + a (x_1, y_1, z_1) + b (x_2, y_2, z_2), -\infty < a, b < \infty\]
3D Transformation

• Transformation: original location \((x, y, z)\)
  – translation, i.e. move it to \((x+dx, y+dy, z+dz)\)
  – scaling, move it to \((ax, by, cz)\)
  – rotation \(\theta\) along z axis \((x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta, z)\)

• All transformations can be expressed by matrices.
• Let \([x, y, z, 1]\) represent a 3-D location \((x,y,z)\), called **homogeneous coordinates**.
• translation
\[
\begin{bmatrix}
1 & 0 & 0 & dx \\
0 & 1 & 0 & dy \\
0 & 0 & 1 & dz \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= \begin{bmatrix}
x + dx \\
y + dy \\
z + dz \\
1
\end{bmatrix}
\]

• scaling
\[
\begin{bmatrix}
a & 0 & 0 & 0 \\
0 & b & 0 & 0 \\
0 & 0 & c & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= \begin{bmatrix}
ax \\
by \\
cz \\
1
\end{bmatrix}
\]

• rotation along z
\[
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= \begin{bmatrix}
x \cos \theta - y \sin \theta \\
x \sin \theta + y \cos \theta \\
z \\
1
\end{bmatrix}
\]
Transformation **not** Commutative

- $T_1 T_2 \mathbf{v} \neq T_2 T_1 \mathbf{v}$ in general
- $(3,0)$ translate $(3,0)$ then rotate $45^\circ \Rightarrow (4.2, 4.2)$
- $(3,0)$ rotate $45^\circ$ then translate $(3,0) \Rightarrow (5.1, 2.1)$