EE5240: Computer Modeling for Power Systems

Gowtham
Director of Research Computing, IT
Adj. Asst. Professor, Physics/ECE
EERC B39 · (906) 487-3593 · g@mtu.edu

Week #06: 2015/02/16
Programming etiquette

Programming etiquette

* Problem definition
  * Understand what is given
  * Understand the deliverables
  * Divide n’ conquer

* Literature search
  * Your own personal collection
  * Your friends’/colleagues’ collection
  * Computing literature
  * Object oriented learning
Programming etiquette

* A picture is still worth a thousand words

[Image of a comic strip from Dilbert.com]

Programming etiquette

* Language selection
  * More than one if you can
  * Features, specialty and caveats

* Divide n’ conquer
  * Describe each module in detail
  * Accomplish recurring tasks efficiently
  * Makes debugging easier

Examples: Scripting (BASH, PERL, Python), programming (C/C++, FORTRAN, Java, Mathematica, MATLAB), documentation (\LaTeX{}), database (SQL, Oracle), web design (CSS, HTML, PHP)
Programming etiquette

* Appearance
  * Must look pretty
  * Statements and modules in top-down/alphabetical order
  * Readability takes precedence over efficiency

* Communication
  * Meaningful nomenclature
  * Comments
  * Documentation with metrics
  * Revision control
Programming etiquette

// Factorial.c
//
// Computes factorial(n) where n is an integer (>=0) supplied by the user.
// Compilation/Execution takes less the one second on most modern hardware
// running any linux OS with GCC 4.4.7.
//
// Compilation and execution:
// gcc Factorial.c -o Factorial.x -lm
// ./Factorial.x

// Standard headers
#include <stdio.h>

// Function declaration
int factorial(int n);

// main()
int main() {

  // Variable declaration
  int n;
  long int N;

  // Accept user input
  printf("A non-negative integer: ");
  scanf("%d", &n);

  // Include user input verification
  // Compute factorial and print result
  N = factorial(n);
  printf("factorial(%d) = %ld\n", n, N);

  // Indicate the termination of main()
  return 0;
}

// factorial(n)
int factorial(int n) {

  // Variable declaration
  long int M;

  if (n == 0 || n == 1) {
    M = 1;
  }

  if (n > 1) {
    M = n * factorial(n - 1);
  }

  // Indicate the termination
  return M;
}
Programming etiquette

Premature optimization
Act of letting performance considerations affect the design.

* Testing
  * Check every line/step, and input/output
  * Be a devil’s advocate and check for extreme cases
  * Does the program do NOTHING when it is supposed to NOTHING?

It is better to design, then code from the design, and then profile/benchmark the resulting code to identify which parts should/can be optimized. A simple and elegant design is often easier to optimize, and profiling may reveal unexpected performance problems that would be hidden behind the curtain of premature optimization.
Programming etiquette

* Debugging
  * Identify the bug
  * Note down the solution and understand why it worked

<table>
<thead>
<tr>
<th>Programming etiquette</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Angry Spouse Bug</td>
</tr>
<tr>
<td>* Bloombug</td>
</tr>
<tr>
<td>* Bugfoot</td>
</tr>
<tr>
<td>* Common Law Feature</td>
</tr>
<tr>
<td>* Defensive Coding</td>
</tr>
<tr>
<td>* Heisenbug</td>
</tr>
<tr>
<td>* Higgs Bugson</td>
</tr>
<tr>
<td>* Hindenbug</td>
</tr>
</tbody>
</table>

http://blog.codinghorror.com/new-programming-jargon/
Errors

Errors

Approximation error
A discrepancy between an exact value and some approximation to it.

* Physical and mathematical
  * Point particle, spherical cow, ignore resistance and spin
  * Newtonian vs Einsteinian mechanics, special vs general relativity
  * Indiana $\pi$ Bill (1897)
  * Ignore smaller (or bigger) terms
Errors

Approximation error
A discrepancy between an exact value and some approximation to it.

Suppose that $A$ is the exact value, and $A_{\text{approximation}}$ is the approximate value, then

\[ \epsilon_{\text{absolute}} = |A - A_{\text{approximation}}| \]

\[ \epsilon_{\text{relative}} = \frac{\epsilon_{\text{absolute}}}{|A|} \]

\[ \epsilon\% = \epsilon_{\text{relative}} \times 100 \]
Errors

**Approximation error**

A discrepancy between an exact value and some approximation to it.

Suppose that the exact value, $A$, is not known, then

\[
\epsilon_{\text{absolute}} = |A_{\text{current approximation}} - A_{\text{previous approximation}}|
\]

\[
\epsilon_{\text{relative}} = \frac{\epsilon_{\text{absolute}}}{|A_{\text{current approximation}}|}
\]

\[
\epsilon\% = \epsilon_{\text{relative}} \times 100
\]
Errors

**Truncation error**
Difference between the true result and the result produced by a given algorithm using exact arithmetic.

※ 3.141592653589793 = 3.1415 (i.e., rounding to zero)

※ Limiting the number of terms in series approximations

\[ \pi = 3 + \frac{4}{2 \times 3 \times 4} \]

\[ \pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} \]
Errors

Round-off error
Difference between the result produced by a given algorithm using exact arithmetic and that by the same algorithm using limited precision arithmetic.

* \( 3.141592653589793 = 3.1416 \)

* Fixed-point and floating-point representations (binary arithmetic)
  \[ 1 = 2^0 \]
  \[ 4.875 = 2^2 + 2^{-1} + 2^{-2} + 2^{-3} \]
  \[ 0.10 = \frac{1}{10} \approx 2^{-4} + 2^{-5} + 2^{-8} + 2^{-10} + 2^{-11} + 2^{-12} + \ldots \]
Errors

<table>
<thead>
<tr>
<th>Logic (semantic) error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditions or variables described incorrectly, and the result does not follow the rules of logic.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design error</th>
</tr>
</thead>
<tbody>
<tr>
<td>A flaw in the algorithm or a mistake in the flow of processing leads to determination of an incomplete or incorrect result.</td>
</tr>
</tbody>
</table>

Programs with such errors often compile successfully but yield undesired result without crashing.
Errors

**Compiler error**
Compiler fails to compile the program – either because of syntax errors in the code or errors in the compiler itself.

**Run time error**
A successfully compiled program fails to run completely or successfully.

**Propagation error**
An error in a later step of a process due to an error in an earlier step.

**Overflow/Underflow error**
Produced as a result of too big/small an exponent (±Inf, NaN, 0).
Debugging programs

The company pays me ten dollars for every bug I fix in my code, Ratbert.

I want you to do your little rat dance on my keyboard so I’ll have lots of bugs to fix.

How am I doing?

Not so good. You just authored a web browser.

Debugging programs

Debugging
Act of analyzing the code with the intention of finding, understanding and solving a bug, and make it run successfully to produce meaningful and reproducible result.

* Taking detailed notes
* Using `printf()` (or equivalent) statements
* Smarter text editors: `emacs` | `gedit` | `Sublime Text` | `vim`
* Free and open source tools: `ddd` | `eclipse` | `gdb` | `valgrind`
* Commercial tools: `IBM Rational Purify` | `IDB` | `MATLAB` | `pgdbg`

Hope is that you will develop good programming etiquette along with taking detailed notes and learn to minimize your reliance on debuggers. Error/Help messages from many debuggers often look cryptic to an untrained eye, and might require a fair amount of time to understand them.
Computer History Museum in Mountain View, CA, has a plethora of examples from legendary researchers – Seymour Cray, Robert Noyce and more – demonstrating the value of keeping notes.
Profiling programs

Profiling programs

Profiling

Act of analyzing the code with the intention of understanding where it is spending time and improving its performance.

- Free and open source tools: eclipse | gprof | valgrind
- Commercial tools: IBM Rational Performance Tester | Intel VTune MATLAB | pgprof

Hope is that you will develop good programming etiquette along with taking detailed notes. Even bigger hope is that you will learn not to prematurely optimize the code but write it for clarity, readability and understandability. Information from many profilers are more humane than those from debuggers, and can be used to pick one algorithm over the other.
Profiling programs

Which one should we use?

\[
\text{Sum for loop} = \text{increment the sum iteratively}
\]

\[
\text{Sum formula} = \frac{N (N + 1)}{2}
\]

* For \( N = 1000000 \)
  * \text{sum2Nloop} takes 99+\% time
  * \text{sum2Nformula} takes less than 1\% time
Profiling programs  Further improvements

\[
A = \begin{pmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{pmatrix}
\]

* C, C++, Python are row major languages

* A, laid out in linear fashion, would look like
  \[a_{11} \ a_{12} \ \ldots \ a_{1n} \ a_{21} \ a_{22} \ \ldots \ a_{2n} \ \ldots \ a_{m1} \ a_{m2} \ \ldots \ a_{mn}\]

* To loop through the array in above order
  * First, loop over rows
  * Next, loop over columns
Profiling programs  Further improvements

\[ A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \]

* FORTRAN, MATLAB, Octave, R are column major languages

* A, laid out in linear fashion, would look like

\[ a_{11} \ a_{21} \ \ldots \ a_{m1} \ a_{12} \ a_{22} \ \ldots \ a_{m2} \ \ldots \ \ldots \ a_{1n} \ a_{2n} \ \ldots \ a_{mn} \]

* To loop through the array in above order

  * First, loop over columns
  * Next, loop over rows
Do at home exercises

Memory and mathematics

If each double precision element requires 8 bytes, what is the maximum order of a square matrix that can fit within 1 GB RAM?

Computing power

Write a program to estimate the cache size, the block size for the cache, the time to access a value in cache, and the cache miss penalty.

Using the expression

\[
\text{FLOPS} = \text{# of processors} \times \\
\text{CPU clock speed (cycles/sec)} \times \\
\text{# of floating point operations/cycle}
\]

estimate the computing power of your laptop, and a workstation in any of the computing labs at Michigan Tech.
Do at home exercises

Single precision machine epsilon

Machine epsilon is defined as the smallest number number that can be stored. In 32-bit representation of floating point number, $\pm a \times 2^b$, 1 bit is used for sign ($\pm$), 23 bits for mantissa ($a$) and 8 bits for exponent ($b$).

Use this information and estimate machine epsilon for single precision floating point format. How would you accomplish this in code?

Accuracy of a 64-bit double precision number

In 64-bit double precision format, $\pm a \times 2^b$, 1 bit is used for sign ($\pm$), 52 bits for mantissa ($a$) and 11 bits for exponent ($b$).

Given this information, estimate the accuracy (i.e., number of digits after the decimal point) of such format.
Do at home exercises

**Maximum memory in 32- and 64-bit machines**

Unsigned $n$-bit integers have the range: 0 through $2^n - 1$.

Use this information and estimate the maximum memory that can be accommodated within a 32- and 64-bit machine.

**Time required for mathematical operations**

Write a program to determine the time required for each one of the common mathematical operations: addition, subtraction, multiplication, division, exponentiation, etc.

Is the answer different for integers and non-integers?
Is it in agreement with the manufacturer’s claim for such operations?
Do at home exercises

Design, code, debug and improve the performance

\[
\pi_1 = 2 \sum_{n=0}^{\infty} \frac{2^n (n!)^2}{(2n + 1)!} \quad \pi_2 = \sqrt{12} \sum_{n=0}^{\infty} \frac{(-3)^{-n}}{2n + 1}
\]

Write a program, in MATLAB (or a language of your choice), to compute the value of \( \pi \) using the above approximations. Use modular programming (i.e., each approximation is a function that can be repeatedly called for different values of \( n \)).

Keep a written log of bugs and how you resolved them. Estimate how much time the program spends in each function, and compare the computed value of \( \pi \) with the known value of 3.141592653589793.

Once the program runs successfully to reproduce meaningful results, tinker with the code to make it run faster.
Additional references

* hpc.mtu.edu
  Conferences, webinars, tips, and other opportunities

* MATLAB
  * Introduction
  * Speeding Up Applications
  * Top 10 Productivity Tools
  * Debugging
  * Profiling
Got questions?

EERC B39 · (906) 487-3593 · g@mtu.edu