EECS 22L: Software Engineering Project in C Language

Lecture 9

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 9: Overview

- Software Evaluation and Optimization
  - Linux software development tools
  - Compiler tool chain, optimization and debugging
  - Software profiling and coverage testing
    - GNU profiler **`gprof`**
    - GNU coverage tester **`gcov`**
- Towards Object Oriented Programming in C++
  - Introduction to C++ concepts from the C perspective
  - Introduction to classes, objects and strings

Software Evaluation and Optimization

- GNU Compiler Tool Chain
  - C/C++ Compiler GCC [see EECS22 Lecture 7]
    - **`gcc [options…] [files…] [options…]`**
  - Language Dialect
    - **`-ansi`** selects ANSI-C language semantics
  - Warnings
    - **`-Wall`** enables all warnings
  - Compilation phases
    - **`-E`** preprocessing only, result is preprocessed file (**`.i`**)
    - **`-S`** code generation only, result is assembly file (**`.s`**)
    - **`-c`** compilation only, result is an object file (**`.o`**)
    - (none) all compilation phases, result is executable
  - Output File
    - **`-o name`** selects output filename (default **`a.out`**)

Software Evaluation and Optimization

- GNU Compiler Tool Chain
  - C/C++ Compiler GCC [see EECS22 Lecture 7]
    - **`gcc [options…] [files…] [options…]`**
  - Preprocessor definitions
    - **`-Dmacro`** command-line equivalent of **`#define macro`**
    - **`-Dm=def`** command-line equivalent of **`#define m def`**
    - **`-Umacro`** command-line equivalent of **`#undef macro`**
  - Support for debugging
    - **`-g`** generate symbol tables needed by debugger
    - **`–DDEBUG`** turn on conditional code for debugging
  - Optimization options
    - **`-O2`** optimize the generated code for speed (level 2)
    - **`-DNDEBUG`** turn off debugging support (i.e. assertions)

Software Evaluation and Optimization

- GNU Compiler Tool Chain
  - C/C++ Compiler GCC [see EECS22 Lecture 7]
    - **`gcc [options…] [files…] [options…]`**
  - Input files for different stages
    - Module header file (file suffix **`.h`**)
    - Module program file (file suffix **`.c`**)
    - Module object file (file suffix **`.o`**)
    - Static or shared library file (file suffix **`.a`**, or **`.so`**)
  - Support for performance profiler **`gprof`**
    - **`-pg`** turns on profile instrumentation
  - Support for coverage testing tool **`gcov`**
    - **`-fprofile-arcs`** turns on flow graph support
    - **`-ftest-coverage`** turns on coverage support

Software Evaluation and Optimization

- Execution Time Measurement
  - Linux kernel timing
    - **`/usr/bin/time command`**
- GNU Profiling Tools
  - GNU Profiler **`gprof`**
    - Find the critical performance bottleneck in a program
      - **`gcc –pg –g example.c …`**
      - **`./example`**
      - **`gprof example`**
  - GNU Coverage Testing Tool **`gcov`**
    - Determine code coverage when testing a program
      - **`gcc –fprofile-arcs -ftest-coverage –g example.c …`**
      - **`./example`**
      - **`gcov example`**

Software Evaluation and Optimization

- Performance Profiling Example

```
/* example.c: simple example to demonstrate profiling tools */
/* RD, 02/05/13. */

void CountN(int n)
{
    int i, c = 0;

    for(i=0; i<n; i++)
    {   c = c + 1;
    }
}

void FmulN(int n)
{
    int i;
    double d = 1.1;

    for(i=0; i<n; i++)
    {   d = d * 1.1;
    }
}
...
```


Software Evaluation and Optimization

- Performance Profiling Example

```
...
long Fibo(long n)
{
   if (n <= 1)
      { return n; }
   else
      { return Fibo(n-1) + Fibo(n-2); }
}

int main(void)
{   int x;
    for(x=0; x<100; x++)
    {   if (x % 3)
        {   CountN(1000); }
        else
        {   if (x % 77 == 0)
            {   Fibo(42); }
            FmulN(1000);
        }
    }
    return 0;
}
/* EOF */
```

Software Evaluation and Optimization

- Performance Profiling Example

```
doemer@ladera:1 > cd eecs22l/lecture9/
doemer@ladera:2 > vi example.c
doemer@ladera:3 > gcc example.c -o example -ansi -Wall -g
doemer@ladera:4 > example
doemer@ladera:5 > /usr/bin/time example
4.39user 0.00system 0:04.40elapsed 99%CPU (0avgtext+0avgdata 0maxres.)k
0inputs+0outputs (0major+101minor)pagefaults 0swaps
doemer@ladera:6 > gcc example.c -o example -ansi -Wall -g -pg
doemer@ladera:7 > /usr/bin/time example
19.53user 0.00system 0:19.54elapsed 99%CPU (0avgtext+0avgdata
0maxresident)k
0inputs+0outputs (0major+126minor)pagefaults 0swaps
doemer@ladera:8 > ls
example*  example.c  gmon.out
doemer@ladera:9 >
```


Software Evaluation and Optimization

- Performance Profiling Example

```
doemer@ladera:9 > gprof example
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 93.31      3.50     3.50        1     3.50     3.50  Fibo
  7.53      3.78     0.28       34     0.01     0.01  FmulN
  0.00      3.78     0.00       66     0.00     0.00  CountN

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.
[…]
```

Software Evaluation and Optimization

- Performance Profiling Example

```
[…]
Call graph (explanation follows)

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    3.78                 main [1]
                3.50    0.00       1/1           Fibo [2]
                0.28    0.00      34/34          FmulN [3]
                0.00    0.00      66/66          CountN [4]
-----------------------------------------------
                                866988872             Fibo [2]
                3.50    0.00       1/1           main [1]
[2]     92.5    3.50    0.00       1+866988872 Fibo [2]
                                866988872             Fibo [2]
-----------------------------------------------
                0.28    0.00      34/34          main [1]
[3]      7.5    0.28    0.00      34          FmulN [3]
-----------------------------------------------
                0.00    0.00      66/66          main [1]
[4]      0.0    0.00    0.00      66          CountN [4]
-----------------------------------------------
This table describes the call tree of the program […]
doemer@ladera:10 >
```

Software Evaluation and Optimization

- Coverage Testing Example

```
[…]
doemer@ladera:12 > rm gmon.out
doemer@ladera:13 > ls
example*  example.c
doemer@ladera:14 > gcc example.c -o example -ansi -Wall -g -fprofile-arcs
-ftest-coverage
doemer@ladera:15 > ls
example*  example.c  example.gcno
doemer@ladera:16 > /usr/bin/time example
4.81user 0.00system 0:04.82elapsed 99%CPU (0avgtext+0avgdata 0maxres.)k
0inputs+0outputs (0major+132minor)pagefaults 0swaps
doemer@ladera:17 > ls
example*  example.c  example.gcda  example.gcno
doemer@ladera:18 > gcov example
File 'example.c'
Lines executed:100.00% of 22
example.c:creating 'example.c.gcov'

doemer@ladera:19 >
```

Software Evaluation and Optimization

- Coverage Testing Example

```
doemer@ladera:19 > ls
example*  example.c  example.c.gcov  example.gcda  example.gcno
doemer@ladera:20 > cat example.c.gcov
        -:    0:Source:example.c
        -:    0:Graph:example.gcno
        -:    0:Data:example.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        -:    1:/* example.c: simple example to demonstrate profiling */
        -:    2:/* RD, 02/05/13. */
        -:    3:
        -:    4:void CountN(int n)
       66:    5:{
       66:    6:    int i, c = 0;
        -:    7:
    66066:    8:    for(i=0; i<n; i++)
        -:    9:    {
    66000:   10:        c = c + 1;
        -:   11:    }
       66:   12:}
[…]
doemer@ladera:21 >
```


Software Evaluation and Optimization

- Observation and Debugging Tools
  - GNU Debugger [see EECS22 Lecture 10]
    - **gdb**
  - Data Display Debugger [see EECS22 Lecture 10 and 15]
    - **ddd**
  - Dynamic Memory Validator [see EECS22 Lecture 12]
    - **valgrind**

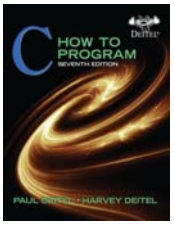
Object Oriented Programming

- Towards Object Oriented Programming in C++
  - C++ can be seen as “improved” C
  - C++ offers a number of new features, including:
    - Inline functions
    - References
    - Default arguments
    - Function and operator overloading
    - Classes and objects
    - Member functions (methods)
    - Constructor and destructor
    - Class and function templates
    - Class inheritance
    - Polymorphism
    - Exception handling

Object Oriented Programming

- “Crash Course” Introduction to C++
  - Selected slides from supplemental text book:

    Paul Deitel, Harvey Deitel,
    “C: How to Program”,
    Seventh Edition,
    Prentice Hall, 2013.



  - Excerpts from Chapter 16:
    Introduction to Classes, Objects and Strings