C Programming Review & Productivity Tools

Giovanni Agosta

Piattaforme Software per la Rete – Modulo 2

Outline

Preliminaries

What are we looking at in this course?

- Using TCP/IP in application software
- The Client-Server model
- Application protocols
  - standard: remote login, file transfer, email, etc.
  - non-standard
  - using standard protocols for non-standard uses: telnet
    towel.blinkenlights.nl

Preliminaries

What are we looking at in this course?

- Using TCP/IP in application software
- The Client-Server model
- Application protocols

But we also need good programming skills!

- C programming language
- Programming *well*
- Shell scripting
- Programming productivity tools (`make`, `gdb`)

Organization of the course

Syllabus

- The Practice of Programming
- Network Administration and Security
- TPC/IP & Client-Server Applications
- Advanced topics (kernel modules, wireless networks)

Labs & Projects

- Both are *optional*
- Labs: try out SW development practices, building an application
- Project: a more advanced design & development task, replacing the exam

Evaluation

Written exam

- Each module is assessed separately
- Time for exam: 90'
- Structure: one/two exercises per syllabus item

Projects

- Multiple batches of projects
- First batch will be presented on 12/4
- Second batch will be presented near the end of the course
- Project replaces one (and only one) module
- Must be taken by two (or three) students, *all* are responsible for the *entire* work

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Function Pointers
Data Types and Qualifiers
Variadic Functions

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Function Pointers
Data Types and Qualifiers
Variadic Functions

Function Pointers
Overview

Generalities

- Functions are *not* variables, per se
- But, you can declare function pointers

A first example

```c
int plus(int a, int b){ return a+b; }
int apply(int x, int y,
          int (*funcptr)(int, int)){
    return funcptr(x,y);
}
```

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Function Pointers
Data Types and Qualifiers
Variadic Functions

Function Pointers
Usage

- Switch-like constructs
- Generic functions
- Callbacks

- calc.c
- qsort.c
- obj.c

Let's have a look at these examples...

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Function Pointers
Data Types and Qualifiers
Variadic Functions

Data Types and Qualifiers
Unions

What

- A type for representing multiple types
- Forces alignment to the longest type

Why

Need fixed size structures with variable content

Example

```
typedef union {
    char chr;
    int itg;
    char *str;
} _data;
```

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Function Pointers
Data Types and Qualifiers
Variadic Functions

Data Types and Qualifiers
Type Qualifiers

Volatile

- Forces all accesses to be in memory
- Needed when the compiler may be unaware of external accesses to a variable

volatile int a;

Const

- The variable is considered read-only by the compiler

const int a = 1;

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Function Pointers
Data Types and Qualifiers
Variadic Functions

Data Types and Qualifiers
Storage Class Specifiers

auto

- Standard automatic variable
- **register**: subclass where address cannot be taken

static

- Applied to variables: variable persist between function calls
- Applied to functions: function is not seen outside the compilation unit

extern

- Applied to variables: variable declared outside the function
- Applied to functions: function defined in another compilation unit

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Function Pointers
Data Types and Qualifiers
Variadic Functions

Variadic Functions
Handling variable parameters in C

stdarg.h

Defines the following macros:

- va_list : data type

- **void** va_start (va_list args , *last*): initialize scanning, starting from parameter *last*

- **void** va_end(va_list args): end scanning

- *type* va_arg(va_list args , *type*): get next argument, casting to type *type*

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Overview
Build Automation
Code Versioning
Debugging

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Overview
Build Automation
Code Versioning
Debugging

Tools for Productivity in Programming
Overview

Tasks

- Compiling
- Building
- Versioning
- Debugging

There are several solutions for each task!

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Overview
Build Automation
Code Versioning
Debugging

Tools for Productivity in Programming
Overview

Solutions

- Compiling: gcc, icc
- Building: make, SCons, autoconf, CMake
- Versioning: mercurial, git, svn, cvs
- Debugging: gdb, idb

We focus on the GNU tools

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Overview
Build Automation
Code Versioning
Debugging

Build Automation
GNU make

GNU `make` Basics

- Variables

- Rules

- Patterns, wildcards and much more

Today, we will look at a few basics only!

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Overview
Build Automation
Code Versioning
Debugging

Build Automation
GNU make

Automatic Variables

$@	The file name of the target
$<	The name of the first prerequisite
$?	The names of all prerequisites newer than the target
$^	The names of all prerequisites
$+	Like above, but keeps duplicates

Variable definition

objects = *.o
objects := $(wildcard *.o)

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Overview
Build Automation
Code Versioning
Debugging

Build Automation
GNU make

Rules

%.o: %.c
         $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

qsort: qsort.c
         gcc -o qsort qsort.c

Variable definition

- Generic vs specific rules
- Prefer specific to generic: shortest *stem* rule

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Overview
Build Automation
Code Versioning
Debugging

Versioning
Mercurial

A Quick Primer

create a repository	hg init *directory*
copy a (remote) repository	hg clone *address*
add files to repository	hg add *files*
commit changes to changeset	hg commit -m '*comment*'
push changes to other repository	hg push *address*
pull changes to other repository	hg pull *address*
merge different history lines	hg merge

Preliminaries
C Programming
Tools for Productivity in Programming
Conclusions

Overview
Build Automation
Code Versioning
Debugging

Debugging
GNU Debugger (gdb)

A Quick Primer

- Compiling for debugging: -g flag
- Setting arguments and running: `set args`, `run`
- Multiple threads: `info threads`, `thread` *n*
- Breakpoints: `break` at function, line or address, can be conditional or thread-specific
- Continuing execution: `continue` (to next breakpoint), `step` (to next source line), `next` (to next line in same stack frame)
- Get info about the program: `info`
- Examine the stack: `bt`, `up`, `down`
- Examine data: `print`

Concluding Remarks

What to do now?

- Programming in C with dynamic libraries
    - Write a dynamic library supporting a data type (e.g., lists or trees)
    - Use it in implementing a simple program (e.g., indexing
- Setting up a code project
    - Using versioning
    - Creating makefiles