

Mastering your Linux: C and Shell Programming

Martin Quinson <martin.quinson@loria.fr>

ESIAL

2009-2010

(compiled on: January 26, 2010)

Introduction

Course Goals

- ▶ Help you mastering your second programming language
 - ▶ Basics about the syntax
 - ▶ Caveats (of memory management, amongst other)
 - ▶ Get some good style
- ▶ Help you mastering your Linux box (or any other UNIX-based one)
 - ▶ Fluent use of the terminal
 - ▶ Non-trivial command lines
 - ▶ Simple scripts

Prerequisite

- ▶ Algorithmic Background: you cannot program without that
- ▶ Java Programming: we won't learn to program, but how to write it in C

Administrativae

Module Time Table

- ▶ One full-group lecture (today): introduction to the module
- ▶ 7 small group lectures / 6 practical labs (plus exam): The C language
- ▶ 2 small group lectures / 6 practical labs (plus exam): Shell Scripting

Evaluation

- ▶ Test on table (*partiel*) on C language
 - ▶ **What:** Content of lectures and labs (of course)
 - ▶ **When:** someday in march (check ADE agenda)
 - ▶ **Allowed material during test:** one A4 sheet of paper only
 - ▶ Hand-written (not typed)
 - ▶ From you (no photocopy)
- ▶ Test on table on Shell Scripting
 - ▶ **When:** someday in may (check ADE agenda)
 - ▶ (Ask Suzanne Collin for details)

About me

Martin Quinson

- ▶ **Study:** Université de Saint Étienne, France
- ▶ **PhD:** Grids and HPC in 2003 (team Graal of INRIA / ENS-Lyon, France)
- ▶ **Since 2005:**
 - ▶ Assistant professor at ESIAL (Univ. Henri Poincaré–Nancy I, France)
 - ▶ Researcher of AIgorille team of LORIA/INRIA
- ▶ **Research interests:**
 - ▶ **Context:** Distributed Systems
 - ▶ **Main:** Simulation of Distributed Applications (SimGrid project)
 - ▶ **Others:** Experimental Methodology, Model-Checking, ...
- ▶ **Teaching duties:**
 - ▶ Responsible of first year cursus at ESIAL
 - 1A: **PPP:** introduction to Java; **TOP:** Technics and tOols for Programming; **CSH:** C as Second Language (and Shell)
 - 2A: **RS:** System Programming (and Networking)
 - 3A: Peer-to-Peer Systems and Advanced Distributed Algorithms (master)
- ▶ **More infos:**
 - ▶ <http://www.loria.fr/~quinson> (Martin.Quinson@loria.fr)

References: Courses on Internet

- ▶ **Introduction to Systems Programming** (C. Grothoff)
C covered, but not only.
<http://grothoff.org/christian/teaching/2009/2355/>
- ▶ **C / Shell** (A. Crouzil, J.D. Durou et Ph. Joly; U. Paul Sabatier, Toulouse)
Good coverage of the whole module (in French).
http://www.irit.fr/ACTIVITES/EQ_TCI/ENSEIGNEMENT/CetSHELL/
- ▶ **Support de Cours de Langage C** (Christian Bac; INT Evry)
The C Language (in French).
<http://picolibre.int-evry.fr/projects/svn/coursc/>

Table of Contents

▶ Introduction and Generalities

- ▶ Introduction; Motivation; History.

1 Part I: C as Second Language

▶ Syntax and Basic usage

- ▶ Introduction; First C program and compilation; Syntax, printf; C vs. Java.

▶ Memory Management in C

- ▶ Variable visibility, storage class; Malloc and friends; Debugging problems.

▶ Advanced C Topics

- ▶ Modularity in C; Makefile; Performance tuning; Game programming.

2 Part II: Shell Scripting

▶ Low Script-fu knowledge

- ▶ Introduction; First shell “scripts”; Redirecting I/O & Pipes; basic commands.

▶ Medium Script-fu knowledge

- ▶ More Syntax for Advanced Scripts; Not so basic commands.

▶ Advanced Script-fu knowledge

- ▶ Shell functions; Variable Substitutions; Sub-shells; Arrays.

Premier chapitre

C and Unix

- Introduction

 - C? UNIX? What is all this about?

 - Why do we need to study C?

 - Why do we need to study C and UNIX together?

- C as Second Language

 - C vs. Java

 - How to survive in C?

 - Your first C program

- First steps in Unix

 - Désignation des fichiers

 - Protection des fichiers

 - Using the terminal

C? UNIX? What is all this about?

Let's go for a little pool, please

- ▶ Who never heard the word “Unix” *before arriving* at ESIAL?
- ▶ Who in the room have Linux installed on a computer at home?
- ▶ Who have a network of Linux boxes at home?

ESIAL population very heterogeneous

- ▶ Usually about $\frac{1}{3}$ didn't heard about Unix before arriving, and $\frac{1}{3}$ use it already
- ▶ We are here to level everybody
- ▶ Yep, some of you already know the first lectures (go get some maths)
- ▶ But be patient, soon, everyone will be lost (including YOU!)

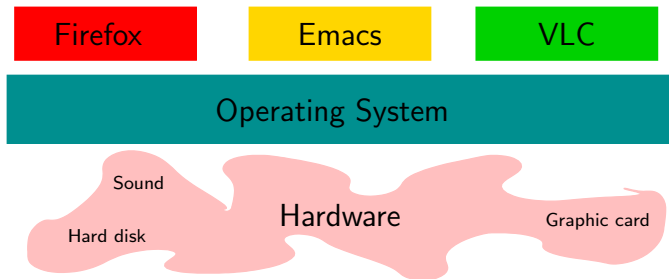
Further Quizz

- ▶ Could you define Unix in a word?

Operating System

What is an Operating System?

- ▶ That's the software between the applications and the hardware
- ▶ Handles (and protects) the resources
- ▶ Offers an unified interface to the applications

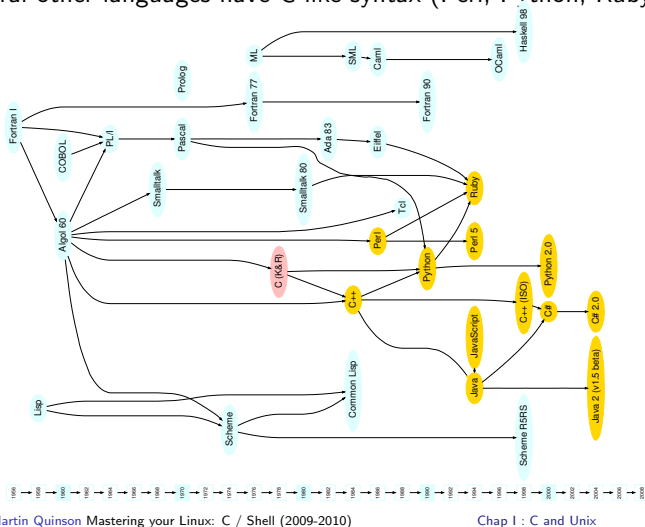


Operating System Basics

- ▶ What's the Operating System on neptune host (where you do your labs)?
- ▶ What's the difference between that and the Unix we spoke about before?
- ▶ What other Operating System you know?
- ▶ Any idea of the amount of existing Operating System? Guess the count
- ▶ What's the link between Mac OS and the other OSes?
- ▶ Why don't we speak of Windows instead?
- ▶ If so, why do we speak of Unix anyway?
 - 1.
 - 2.

Why should we study the C language? Huge impact

- ▶ C++ is an object extension of C (you cannot master C++ without C)
- ▶ Java is some sort of (safe) subset of C++; C# is a variation of Java
- ▶ Several other languages have C-like syntax (Perl, Python, Ruby, PHP)

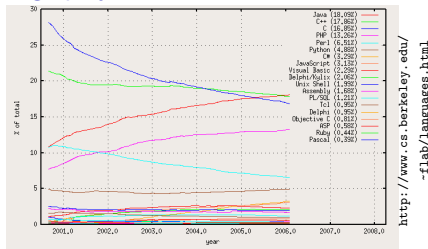


<http://merd.sourceforge.net/pixel/language-study/diagram.html>

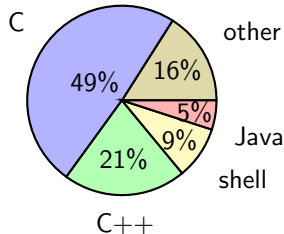
See also <http://www.digibarn.com/collections/posters/tongues/>

Why should we study the C language? Widespread

- ▶ De facto standard for System Programming: Windows, OS X, Linux, BSD in C
- ▶ Counting SourceForge projects. Java: 18%; C++: 17.9%; C: 15.9%



- ▶ Counting SLOC in Debian. Quite different numbers...



More details:

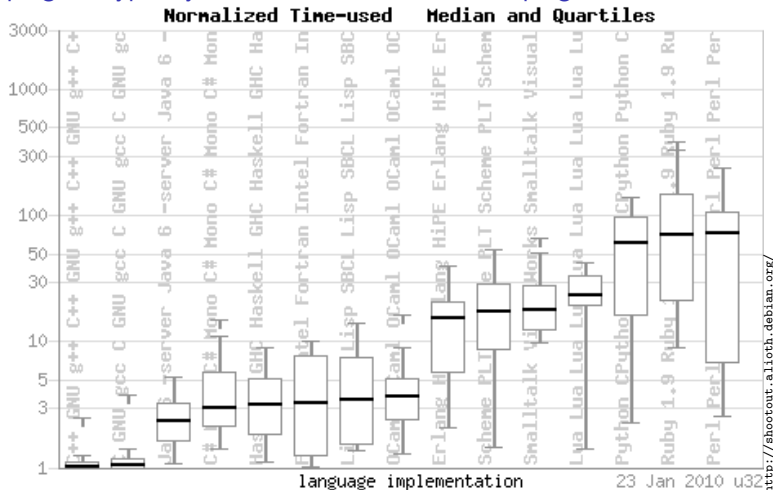
<http://debian-counting.libresoft.es/lenny/>

See also: <http://www.dwheeler.com/sloc/>

- ▶ Big codes are in C/C++
- ▶ Toy projects tend to be in Java (but things change)

Why should we study the C language? Fast

- ▶ C program typically execute faster than in other programs



- ▶ One could argue that this is because it has the best tools, but not only

Studying the C language for educational purpose

Understanding C helps you understanding the system as a whole

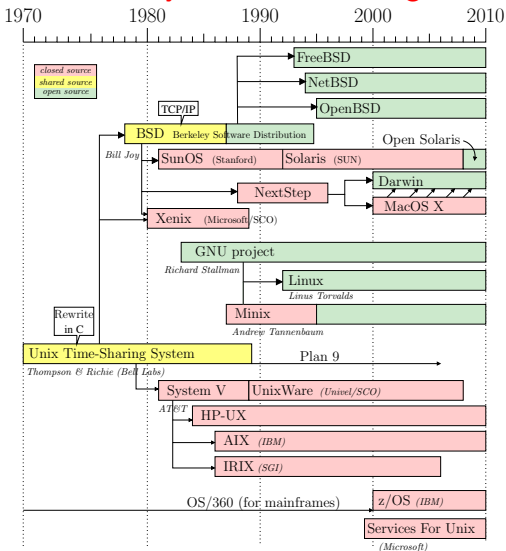
- ▶ C is the closest high-level language to the machine
- ▶ Every OS are written in C, so lower interface is in C/C++ too
- ▶ OS/hardware co-evolution: C conceptual model describes most hardware

Understanding C helps you writing better Java code

- ▶ Java/.Net/Perl/etc hide underlying low-level mechanism
- ▶ But these mechanism can be very important (to performance for example)
- ▶ To understand how objects get passed by ref, realize that they are pointers

Why do we need to study C and UNIX together?

Because they were invented together!



Unix history

- 1965 MULTICS: ambitious system project (Bell Labs)
- 1969 Bell Labs give up MULTICS, UNICS begins
- 1970 Unix: Official Bell Labs project
- 1973 Rewrite in C
Distribution to universities
Sold by AT&T
- 80-90 Unix War: BSD vs. System V
- 90-10 Normalization Effort (POSIX)

C history

- 1967 BCPL used at Bell Labs;
- 1968 B [Thompson]: simplification
- 1971 C K&R (somehow typed)
- 1983 C++: object oriented
- 1989 ANSI C; 1990: ISO C (C90)
- 1999 ISO C updated (C99)

What is Special About C?

Low Level: sort of abstract assembly language of historical processors

- ▶ Was invented on a PDP-11 with 24kb of memory: KISS!
- ▶ Process memory visible as an array of bytes
- ▶ Nothing in the language will prevent you from doing (really) stupid things

C combines the power of assembler with the portability of assembler.

Extensible: most higher-level features doable in C

- ▶ Self-modifying code, Introspection, Code migration, etc. (but all by yourself)
- ▶ (actually, JVM partially written in C/C++)

If you can't do it in ANSI C, it isn't worth doing.

Relatively Stable: almost backward compatible since seventies

- ▶ Other languages got heavily lifted too often (but some heritages unpleasant)

C has hardly any runtime system

- ▶ Small footprint, easily ported to new architectures (need to reinvent the wheel)

So, should we use C once studied?

Benefits

- ▶ More control over the execution behaviour of programs
- ▶ More opportunity for performance tuning
- ▶ Access to low-level features of system

Disadvantages

- ▶ Need to do your own memory management
- ▶ Typically takes more lines of code to accomplish each task
- ▶ More opportunity to make mistakes

Summary

- ▶ C is a powerful programming tool for experts
 - ▶ Presents many potential hazards for novices
 - ▶ Helps you to understand low-level execution ideas
 - ▶ Helps transforming you from a novice to an expert
- ↪ Use it when you need it, avoid it when you don't need it

Premier chapitre

C and Unix

- Introduction

- C? UNIX? What is all this about?

- Why do we need to study C?

- Why do we need to study C and UNIX together?

- C as Second Language

- C vs. Java

- How to survive in C?

- Your first C program

- First steps in Unix

- Désignation des fichiers

- Protection des fichiers

- Using the terminal

C as Second Language

Similarities between C and Java

- ▶ Operators
 - ▶ Arithmetic (+,-,*,/,% ; ++,- -,*=,...); Bitwise (&,|,^,!,<<,>>)
 - ▶ Relational (<,>,<=,>=,==,!); Logical &&, ||, !, (a?b:c)
- ▶ Language Constructs
 - ▶ `if(){ } else { }` ▶ `for(i=0; i<100; i++){ }`
 - ▶ `while(){ }` ▶ `do { } while()`
 - ▶ `switch() { case 0: ... }` ▶ `break, continue, return`
- ▶ Basic (primitive) types: void, int, short, long; float, double; char.
No boolean, use int instead (0=False; anything else=True)
- ▶ Function declarations: `int fact(int a){return a==0 ? 1 : a*fact(a-1);}`

Differences between C and Java

- ▶ No exception: usually rely on int error code instead (and usually a pain)
- ▶ No class/package/interface: code modularity different (not compiler-enforced)
- ▶ No garbage collector: alloc and free manually needed memory (incredible pain)

C as Second Language

C seems familiar when you know Java

- ▶ Actually, that's Java which is highly inspired from C/C++
- ▶ Feels like a Java without any object but with full access to everything

C is like Java without comfort and without any protections

- ▶ Standard library is poor (but huge amount of extensions)
- ▶ Compiler is incredibly permissive (by default)
- ▶ It's possible to shoot yourself in the foot in Java, that's common in C
- ▶ On error, Java displays a stack trace, C spits “segfault” or “invalid free” errors

Unix was not designed to stop people from doing stupid things, because that would also stop them from doing clever things.

– Doug Gwyn

C main specificities in a Nutshell

- ▶ Memory fully accessible through *pointers*
- ▶ Arrays are handled as pointer to memory
- ▶ Declaration syntax very similar to usage syntax (to the price of readability)

How to survive in C?

Use the tools that can help you

- ▶ Use the **compiler warning flags** `-Wall` mandatory, other useful
- ▶ Use a **proper editor** (able of colorization, auto-indent, compile easily)
 - ▶ **Good editors:** emacs & vi (historical), Eclipse/CDT (my personal favorite)
 - ▶ **Bad editor:** gedit (not good for text, BAD for code)
- ▶ The **debugger** (gdb) must become your friend quickly
- ▶ **valgrind** is a piece of magic (C coding without it is masochism)

Don't assume you're a genius (ie, don't do stupid things — yet)

- ▶ Pay attention to the modularity of your code (not compiler-enforced anyhow)
- ▶ Document your code (with readable comments, or doxygen for bigger projects)
- ▶ Get some discipline (coding convention), and stick to it
 - ▶ Symbol naming (`my_variable` or `myVariable`), indentation, etc.
 - ▶ Which one is not very important. Pick one, and stick to it
- ▶ Keep it simple: it's easy to write unreadable C code

Bad Style Coding as a Game

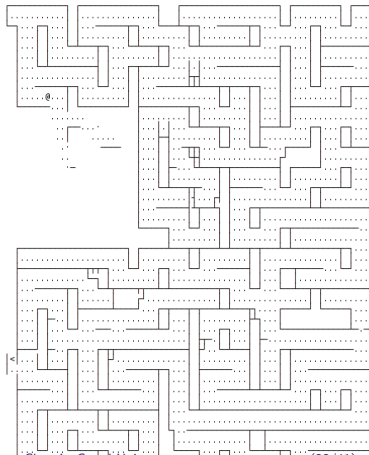
The International Obfuscated C Code Contest (www.ioccc.org)

- ▶ Yearly contest of intentionally obfuscated codes (in C; exist for other languages)

Example: Full (interactive) Maze Escape Game (arachnid, 2004 entry)

Screenshot

```
#include <ncurses.h>/*****
int m[256] [ 256 ],a
,b ;;; ;;; WINDOW*w; char*l="" "\176qx1" "q" "n" "k" "w\
xm" "x" "t" "j" "v" "u" "n" "n",Q[
]= "Z" "pt!ftd" "qdc!'eu" "dq!$c!nnwf"/** ***/"t\040\t";c(
int u , int v){ v7m [u] [v-
1] |=2,m[u][v-1] & 48?W[v-1] & 15]]):0:0;u7m[u -1][v]|=1 ,m[
u- 1][ v]& 48? W-1 ] [v ]&
15] ):0:0;v< 255 ?m[ u][v+1]|=8,m[u][v+1]& 48? W][ v+1]&15]]
):0 :0; u < 255 ?m[ u+1 ] [v ]|=
4,m[u+1][ v]&48?W+1][v]&15]]):0:0;W[ v]& 15] );};cu(char*q){ return
*q ?cu (q+ 1)& 1?q [0] ++:
q[0 ]-- :1; }d( int u , int/**/v, int/**/x, int y){ int
Y=y -v, X=x -u; int S,s =-Y =-Y ,s,
s=- 1:( s=1);X<0?X=-X,S =-1 :(S= 1); Y<= 1;X<=1; if(X>Y){
int f=Y -(X >>1 );; while(u!= x){
f>= 0?v+=s,f=-X:0;u +=S ;f+= Y;m[u][v]|=32;mvwaddch(w,v ,u, m[u
][ v]& 64? 60: 46) ;;if (m[ u][
v]&16){c(u,v); ;;; ;;; return;}} }else(int f=X -(Y>>1); while
(v !=y )){f >0 ?u +=S, f-= Y:0
;v +=s ;f+=X;m[u][v]|= 32;mvwaddch(w,v ,u,m[u][v]&64?60:46);if(m[u
][ v]& 16) {c( u,v );
; return;};}}Z( int/**/a, int b){ e( int/**/y,int/**/ x){
int i; for (i= a;i
+S;i++)d(y,x,i,b),d(y,x,i,b+L);for(i=b;i<=b+L;i++)d(y,x,a,i),d(y,x,a
); ;;; ;;; ;;; ;;; ;;; ;;;
mvwaddch(w,x,y,64); ;;; ;;; prefresh( w,b,a,0 ,L- 1,S-1
);}
main( int V, char *C[
] ) {FILE*f= fopen(V="1?arachnid.c"/**/ :C[ 1],"r");int/**/x,y,c,
(source code cut here)
Martin Quinson Mastering your Linux: C / Shell (2009-2010)
```



Recreational Obfuscation: Phillips entry of IOCC'88

Program code

```
#include <stdio.h>
main(t,_,a)char *a;{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_,+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{w+/w#cdnr/+,}{r/*de}+,/*{+*,/w{%/+,/w#q#n+,/#{1,+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;+*,/'r : 'd*'3,}{w+k w'K:'+'e#';dq#'l \
q#'+d'K#!/+k#;q#r}eKK#}w'r}eKK{nl}'/;#q#n'}){)#w'}){){nl}'+#n';d}rw' i;# \
){nl}!/n{n#'; r#{w'r nc{nl}'/#{1,+ 'K {rw' iK{;[{nl}'/w#q#n'wk nw' \
iwk{KK{nl}!/w{%'l##w#' i; :{nl}'/*{q#ld;r'}{nlwb!/*de}'c \
; ;{nl}'-}{rw}'/+,)##*}#nc,' #nw}'/+kd'+e}+;#rdq#w! nr'/ ') }+}{rl#}'{n' ')# \
}'+'}##(!/"/)
:t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
:0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{: \nuwloca-0;m .vpbks,fxntdCeghiry"),a+1);}

```

Output

On the first day of Christmas my true love gave to me
a partridge in a pear tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a pear tree.

On the third day of Christmas my true love gave to me
three french hens, two turtle doves
and a partridge in a pear tree.

On the fourth day of Christmas my true love gave to me
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

Output (cont)

On the eighth day of Christmas my true love gave to me
eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the ninth day of Christmas my true love gave to me
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the tenth day of Christmas my true love gave to me
ten lords a-leaping,
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

Bad Style Coding as an Art

Another example: Computing Integer Square Roots

```
#include <stdio.h>
int l;int main(int o,char **0,
int I){char c,*D=0[1];if(o>0){
for(l=0;D[l]          ];D[l
++]--=10){D  [l++]--=120;D[l]--
=110;while  (!main(0,0,1))D[l]
+= 20; putchar((D[l]+1032)
/20  )  };putchar(10);}else{
c=o+  (D[I]+82)%10-(I>1/2)*
(D[I-1+I]+72)/10-9;D[I]+=I<0?0
:!(o=main(c/10,0,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

It actually works

```
$ ./cheong 1234
35
```

$(35 \times 35 = 1225; 35 \times 36 = 1296)$

```
$ ./cheong 112233445566
335012
```

$335012 \times 335012 = 112233040144$

$335013 \times 335013 = 112233710169$

Author claim: code self-documented...

```
#include <stdio.h>
int l;int main(int o,char **0,
int I){char c,*D=0[1];if(o>0){
for(l=0;D[l]          ];D[l
++]--=10){D  [l++]--=120;D[l]--
=110;while  (!main(0,0,1))D[l]
+= 20; putchar((D[l]+1032)
/20  )  };putchar(10);}else{
c=o+  (D[I]+82)%10-(I>1/2)*
(D[I-1+I]+72)/10-9;D[I]+=I<0?0
:!(o=main(c/10,0,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

*It is an old observation that the best writers sometimes disregard the rules of rhetoric. When they do so, however, the reader will usually find in the sentence some compensating merit, attained at the cost of the violation. **Unless he is certain of doing as well, he will probably do best to follow the rules.***

– William Strunk, Jr. (1918)

Your first C program

The classical Hello World

hello.c

```
#include <stdio.h>
int main(int argc, char *argv[]){
    printf("Hello, world\n");
}
```

For the record: same in Java

hello.c

```
class HelloWorld {
    public static void main(String[] arg){
        System.out.println("Hello, world");
    }
}
```

Compile and run it

```
$ gcc -Wall hello.c -o hello
$ ./hello
```

Compiling and running Java code

```
$ javac HelloWorld.java
$ java -cp . HelloWorld
```

Explanations

- ▶ #include can be seen as the equivalent of import directives
- ▶ main is the *entry point* of every program (same in C and Java)

C Compilation Process

Compiling a C program involves 3 separate tools

1. **Pre-processor:** Rewrites the code according to the defined *macros*
 - ▶ Lines beginning with "`#`" are macros
 - ▶ `#define name value`: declare a sort of automatic search/replace
 - ▶ `#define name(params) value`: search/replace but with arguments
 - ▶ `#include "file"`: inline the content of the given file
 - ▶ `#ifdef name/#else/#endif`: mask parts of the file if `name` is defined
2. **Compiler:** Translates the code into assembly
3. **Linker:** Take elements in assembly and resolve library dependencies
 - ▶ If your code uses function `cos()`, you need the math lib
 - ▶ The linker solves a puzzle to ensure that every used function get defined

This process is rather transparent to the user

- ▶ You edit your code (in `emacs/vi/eclipse`)
- ▶ You launch `gcc`, which launches mandatory tools automatically
- ▶ You mainly need to know that when you get error messages

What if you get error messages when compiling

Some examples

- ▶ `foo.c:71:2: error: invalid preprocessing directive #deifne`
The preprocessor is not happy: check file `foo.c`, line 71, column 2
- ▶ `foo.c:72: error: expected ')' before 'char'`
Compiler's not happy (syntax error)
- ▶ `foo.c:74: error: redefinition of 'myFunc'`
`foo.c:72: error: previous definition of 'myFunc' was here`
Defining the same function twice makes the linker unhappy
- ▶ `/usr/lib/crt1.o: In function '_start':`
`(.text+0x18): undefined reference to 'main'`
`collect2: ld returned 1 exit status`
A function is used, but never defined
(see RS lecture next year to understand the detail of the message)
- ▶ `Segmentation fault ./myProg`
Your program messed up the memory (valgrind knows where)

How to react when you get error messages (and you will)

- ▶ **Don't panic**, even if the message seem cryptic (they often are)
- ▶ **Read the message**: they are sometimes even understandable
- ▶ **Don't even read the second message**: the parser often gets lost after first error

Conclusion on C (for now)

C is the modern assembly language

- ▶ It's quite prehistorical
 - ▶ Compilation process not trivial (even with only one file)
 - ▶ Cryptic error messages
 - ▶ No fancy stuff in standard library
- ▶ Programs can be really fast
 - ▶ If you do them right; easy to code slow C programs too
- ▶ You have the full power of doing everything
 - ▶ No matter what you want to code, it's possible in C
 - ▶ A lot of code were already developed in C (check koders.com)
 - ▶ C poses no rule to limit your imagination...
 - ▶ ...but there is no barrier to prevent you doing stupid things

You need to master C to understand your machine

- ▶ The operating system is in C, just like the virtual machines
- ▶ And then, you're free to use it or not
Depending on whether you're seeking for fast programs or fast coding

Premier chapitre

C and Unix

- Introduction

 - C? UNIX? What is all this about?

 - Why do we need to study C?

 - Why do we need to study C and UNIX together?

- C as Second Language

 - C vs. Java

 - How to survive in C?

 - Your first C program

- First steps in Unix

 - Désignation des fichiers

 - Protection des fichiers

 - Using the terminal

First steps in Unix

This OS gives a central role to **files**

- ▶ Contains data and executable programs (quite usual)
- ▶ Communication with user : config files, stdin, stdout
- ▶ Communication between processes: sockets, pipes, etc.
- ▶ Interface to the kernel: /proc
- ▶ Interface to the hardware: peripheral in /dev

The **Terminal** is an interface of choice

- ▶ Graphical interfaces exist too, but I still prefer the terminal
- ▶ Lots of tricks make you more efficient with the terminal (more button on my keyboard than on my mouse)
- ▶ If you can't do it in one step, type a one-line script directly

Read The Fine Manual (RTFM)

- ▶ The command `man` gives you access to a large corpus of knowledge
- ▶ `man prog` or `man function` \leadsto documentation of that program or function

Désignation des fichiers

Désignation symbolique (nommage): Organisation hiérarchique

- ▶ Noeuds intermédiaires: répertoires (*directory* – ce sont aussi des fichiers)
- ▶ Noeuds terminaux: fichiers simples
- ▶ Nom absolu d'un fichier: le chemin d'accès depuis la racine

Exemples de chemins absolus :

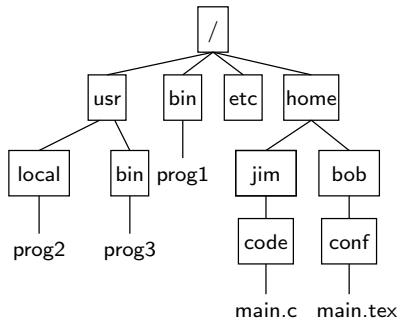
/

/bin

/usr/local/bin/prog

/home/bob/conf/main.tex

/home/jim/code/main.c



Raccourcis pour simplifier la désignation

Noms relatifs au répertoire courant

- ▶ Depuis `/home/bob`, `conf/main.tex` = `/home/bob/conf/main.tex`

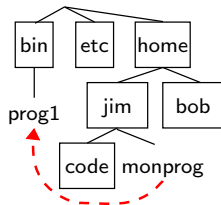
Abréviations

- ▶ Répertoire père: depuis `/home/bob`, `../jim/code` = `/home/jim/code`
- ▶ Répertoire courant: depuis `/bin`, `./prog1` = `/bin/prog1`
- ▶ Depuis n'importe où, `~bob/` = `/home/bob/` et `~/` = `/home/<moi>/`

Liens symboliques

Depuis `/home/jim`

- ▶ Création du lien: `ln -s cible nom_du_lien`
Exemple: `ln -s /bin/prog1 monprog`
- ▶ `/home/jim/prog1` désigne `/bin/prog1`
- ▶ Si la cible est supprimée, le lien devient invalide



Règles de recherche des exécutables

- ▶ Taper le chemin complet des exécutable (/usr/bin/lis) est lourd
- ▶ ⇒ on tape le nom sans le chemin et le shell cherche
- ▶ Variable environnement PATH: liste de répertoires à examiner successivement
/usr/local/bin:/usr/local/sbin:/sbin:/usr/sbin:/bin:/usr/bin:/usr/bin/X11
- ▶ Commande `which` indique quelle version est utilisée

Exercice : Comment exécuter un script nommé `gcc` dans le répertoire courant?

- ▶ Solution 1:
- ▶ Solution 2:

Utilisations courantes des fichiers

- ▶ Unix: fichiers = suite d'octets sans structure interprétée par utilisateur
- ▶ Windows: différencie fichiers textes (où \n est modifié) des fichiers binaires

Programmes exécutables

- ▶ Commandes du système ou programmes créés par un utilisateur
- ▶ Exemple: `gcc -o test test.c ; ./test`

- ▶ Question: pourquoi ./test ?

Fichiers de données

- ▶ Documents, images, programmes sources, etc.
- ▶ Convention:
Exemples : `.c` (programme C), `.o` (binaire translatable, cf. plus loin), `.h` (entête C), `.gif` (un format d'images), `.pdf` (Portable Document Format), etc.
Remarque: ne pas faire une confiance aveugle à l'extension (cf. `man file`)

Fichiers temporaires servant pour la communication

- ▶ Ne pas oublier de les supprimer après usage
- ▶ On peut aussi utiliser des tubes (cf. RS l'an prochain)

Protection des fichiers: généralités

Définition (générale) de la sécurité

- ▶ confidentialité :
- ▶ intégrité :
- ▶ contrôle d'accès :
- ▶ authentification :

Comment assurer la sécurité

- ▶ Définition d'un ensemble de règles (politiques de sécurité) spécifiant la sécurité d'une organisation ou d'une installation informatique
- ▶ Mise en place **mécanismes de protection** pour faire respecter ces règles

Règles d'éthique

- ▶ Protéger ses informations confidentielles (comme les projets et TP notés!)
- ▶ Ne pas tenter de contourner les mécanismes de protection (c'est la loi)
- ▶ Règles de bon usage avant tout:
La possibilité technique de lire un fichier ne donne pas le droit de le faire

Protection des fichiers sous Unix

Sécurité des fichiers dans Unix

- ▶ Trois types d'opérations sur les fichiers : lire (r), écrire (w), exécuter (x)
- ▶ Trois classes d'utilisateurs vis à vis d'un fichier:
propriétaire du fichier ; membres de son groupe ; les autres

rwx	rwx	rwx
propriétaire	groupe	autres

Granularité plus fine avec les Access Control List (peu répandus, pas étudiés ici)

- ▶ Pour les répertoires, r = ls, w = créer des éléments et x = cd.
- ▶ ls -l pour consulter les droits; chmod pour les modifier (cf. man chmod)

Mécanisme de délégation

- ▶ **Problème** : programme dont l'exécution nécessite des droits que n'ont pas les usagers potentiels (**exemple**: gestionnaire d'impression, d'affichage)
- ▶ **Solution** (**setuid** ou **setgid**): ce programme s'exécute toujours sous l'identité du propriétaire du fichier; identité utilisateur momentanément modifiée identité réelle (celle de départ) vs identité effective (celle après setuid)

Crash course on using the terminal

Main idea

- ▶ Your shell is somewhere in the filesystem tree (current directory)
- ▶ You issue commands to interact with the system

Commands Basic Syntax

- ▶ Every command follows this syntax: `<command name> <arguments>`
- ▶ Arguments are space separated
- ▶ Flags are specific arguments beginning usually with - (minus)

Minimal set of commands to remember

Action	Command	Memoing
Examine content of current dir	ls	listing
Know name of current dir	pwd	Print Working Directory
Change current dir	cd	change directory
Copy a file into another	cp	copy
Create a new dir	mkdir	make directory
Destroy a file, a dir	rm, rmdir	remove
Usual shorthand for files and dirs	. .. / ~ *	~user

Using the terminal efficiently

Common Tricks

- ▶ Typing everything is really too slow. You need to be lazy here.
- ▶ `Up`/`Down`: see commands typed previously. Edit it, and go!
- ▶ `Ctrl-A`/`Ctrl-E`: jump to begin/end of line
- ▶ `Tab`: auto-complete what you are currently typing

Medium Tricks

- ▶ `Ctrl-R`: begin to search a text pattern in the command history
- ▶ `!command`: directly relaunch the last command involving that `command`
- ▶ `!!`: directly relaunch the last command

Advanced Tricks

- ▶ Master your terminal (know the base commands)
- ▶ Assemble commands in pipe to get more advanced ones
- ▶ Write one-line scripts directly in the terminal
- ▶ Configure your environment: Declare aliases, write scripts, etc.

Conclusion on Unix (for now)

Unix is one of the most influent operating system

- ▶ Around since 40 years, still there for a long time
- ▶ Most of the OS research innovation go in Unix first (open source power)
- ▶ Other OSes become Unixes (OS X) or get portability layers (z/OS, windows)

You can use that powerful tool too

- ▶ Not as much game as on your Wii, but fully usable and free
- ▶ The interface may be different of what you're used to
- ▶ May be less intuitive at first glance, but there's a strong underlying philosophy
- ▶ Constitute a playground of choice for CS students

Mastering this system is the goal of that course