EPITA - Practical Programming

LSE
Security
System
Laboratory of Epita

01 - Introduction

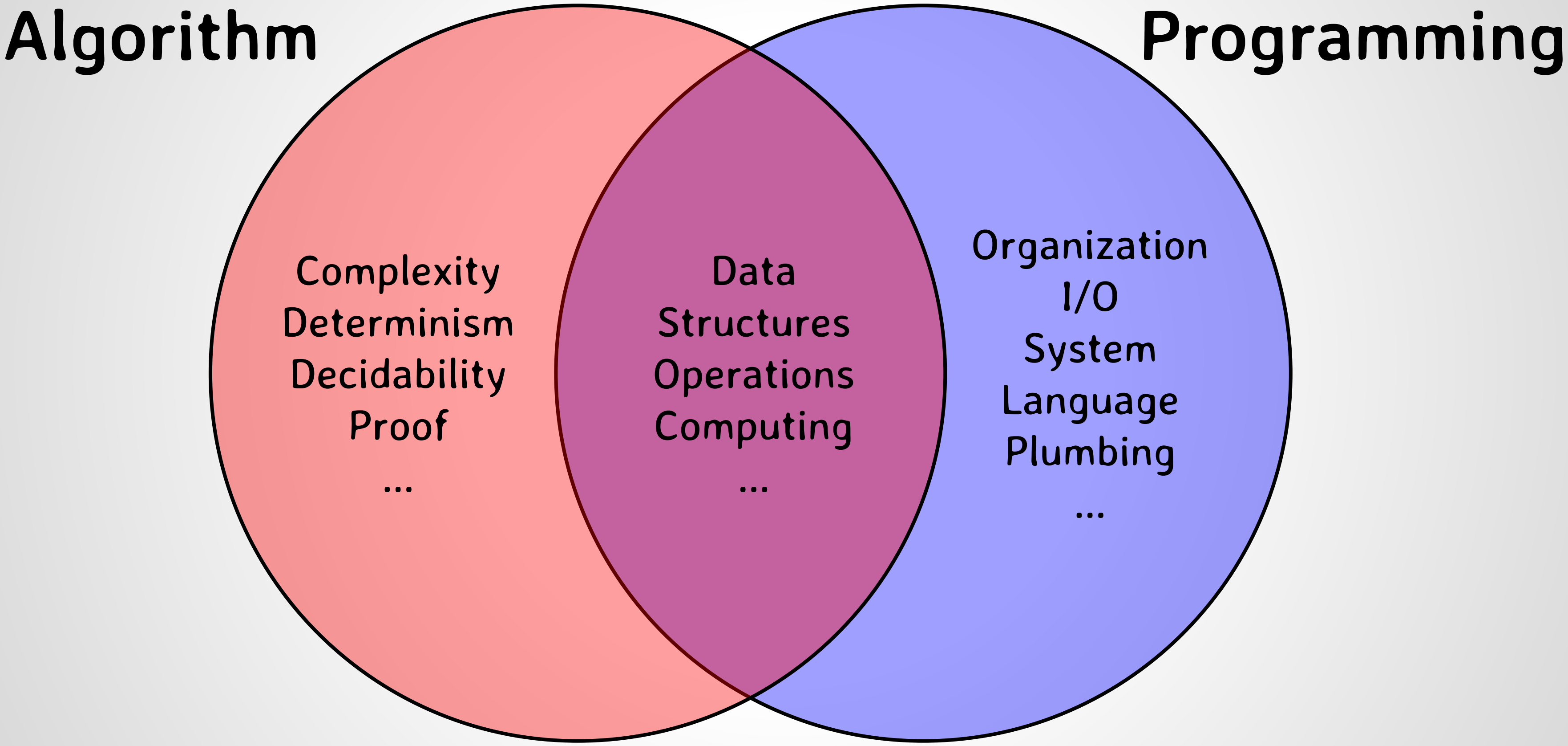
Marwan.Burelle@lse.epita.fr

http://wiki-prog.infoprepa.epita.fr

LSE
Security
System
Laboratory of Epita

“Everyone knows that debugging is twice as hard as writing a program in the first place. So if you’re as clever as you can be when you write it, how will you ever debug it?”

Brian Kernighan, "The Elements of Programming Style", 2nd edition, chapter 2



Algorithm

Programming

Complexity
Determinism
Decidability
Proof
...

Data
Structures
Operations
Computing
...

Organization
I/O
System
Language
Plumbing
...

Practical Programming Lecture - S3

LSE
Security
System
Laboratory of Epita

Overview

The C language:

- Syntax and basis
- Pointers
- Data Structures
- Pointers
- More structures
- Pointers
- ...

Organization:

- 2 tests on machine
- a group project
- practical sessions
- mini solo projects

More overview

- Unix (linux)
- C99/C11 (using gcc and/or clang)
- Program organization
- Understanding memory
- **Programming !**

Programmer's main law:

DO IT !

Marwan’s Programming Laws

1. *copy/paste* are **evil** !
2. Divide and Conquer !
3. The end justifies the mean !
4. "Often" is not enough for saving, almost enough to compile and just enough for testing.
5. **Keep It Simple Stupid**
6. **Code**, don’t procrastinate !

Optimization

Quotes:

- **Make it right before you make it fast.** Make it clear before you make it faster. Keep it right when you make it faster. *P. J. Plauger in [1]*
- We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil.** *from [2]*

Programming Style

- Indent your code
- Stay coherent
- Don’t waste time on aesthetic
- Identifiers should be explicit and short
- 80 columns is enough, more is unreadable

LSE Security System

Laboratory of Epita

About Comments

- Good code don’t need comments
- Comment interfaces, not code
- Keep comments in sync with code
- Don’t waste your time on comments

Quick C Introduction

LSE
Security System
Laboratory of Epita

First Code

```c
# include <stdio.h>
# include <stdlib.h>

int main()
{
  printf("Hello world !\n");
  return 0;
}
```

```
shell> ls
hello.c
shell> gcc hello.c
shell> ls
a.out  hello.c
shell> ./a.out
Hello world !
shell> gcc -Wall -Wextra -std=c99
-O3 -o hello hello.c
shell> ls
a.out  hello  hello.c
shell> ./hello
Hello world !
```

Using make(1)

```c
# include <stdio.h>
# include <stdlib.h>

int main()
{
  printf("Hello world !\n");
  return 0;
}
```

```
shell> ls
hello.c
shell> make hello
cc    hello.c   -o hello
shell> ls
hello hello.c
shell> ./hello
Hello world !
shell>
```

Using make(1)

```
# Simplest Makefile

# Compilers and options
CC=gcc
CPPFLAGS=
CFLAGS= -Wall -Wextra -std=c99 -O3
LDFLAGS=
LDLIBS=

# Empty default rule
all:

# Cleaning rule
clean:
	rm -f *.o *~

# END
```

```
shell> ls
Makefile  hello.c
shell> make hello
gcc -Wall -Wextra -std=c99 -O3 hello.c -o
hello
shell> ./hello
Hello world !
shell> make hello.o
gcc -Wall -Wextra -std=c99 -O3 -c -o
hello.o hello.c
shell> ls
Makefile  hello  hello.c  hello.o
shell> make clean
rm -f *.o *~
shell> ls
Makefile  hello  hello.c
shell>
```

Using make(1)

- Most of the time you don’t need more.
- Keep your Makefile as simple as possible.
- NO NEED FOR COMPILATION RULES!

main function

```
int main(int argc, char *argv[], char *envp[])
{

  return 0;
}
```

- ➢ Always returns int
- ➢ It's a function it must returns
  - 0 is the success default
  - EXIT_SUCCESS is more explicit
  - EXIT_FAILURE or not 0

- ➢ Parameters are optional
  - argc : size of argv
  - argv : command line
  - envp : POSIX extension

Command Line

- Arrays of strings (char pointers).
- Provided/parsed by your shell.
- First element is the program name.
- The shell splits cmd line on spaces.

Using Command Line

```
int main(int argc, char *argv[])
{

  return 0;
}
```

argc : length of argv

argv : arrays containing the command line

Command line

```c
# include <stdio.h>
# include <stdlib.h>

int main(int argc, char *argv[])
{
  for (int i = 0; i < argc; ++i)
    printf("argv[%u] = \"%s\"\n", i, argv[i]);
  return 0;
}
```

```
shell> make cmdline
gcc -Wall -Wextra -std=c99 -O2
cmdline.c   -o cmdline
shell> ./cmdline
argv[0] = "./cmdline"
shell> ./cmdline a b 'c d'
argv[0] = "./cmdline"
argv[1] = "a"
argv[2] = "b"
argv[3] = "c d"
shell>
```

Numeric types

Integers:

- ➢ char, short, int, long and long long
- ➢ All integer constant default to int
- ➢ Use unsigned for natural numbers

Floating point numbers:

- ➢ float and double (maybe more)
- ➢ Floating point constant default to double

LSE Security System
Laboratory of Epita

Sizes

```c
#include <stdio.h>
#include <stdlib.h>
#define PRINT_SIZE(_TYPE_) printf(#_TYPE_ "     \t: %zu\n", sizeof (_TYPE_))
int main()
{
  PRINT_SIZE(char);
  PRINT_SIZE(short);
  PRINT_SIZE(int);
  PRINT_SIZE(long);
  PRINT_SIZE(long long);
  return 0;
}
```

```
shell> gcc -m32 -o int_sizes32 int_sizes.c
shell> ./int_sizes32
char          : 1
short         : 2
int           : 4
long          : 4
long long     : 8
```

```
shell> gcc -o int_sizes64 int_sizes.c
shell> ./int_sizes64
char          : 1
short         : 2
int           : 4
long          : 8
long long     : 8
```

Other types

- size_t : type for sizes, same length as pointers
- ssize_t : when you need negative sizes!
- int8_t, int16_t, int32_t, int64_t
- uint8_t, uint16_t, uint32_t, uint64_t

See the man page for stdint.h

LSE Security System

Laboratory of Epita

Size matters

```c
int fact_int(int n)
{
  int r = 1;
  for (; n > 0; n--)
    r *= n;
  return r;
}

unsigned fact_unsigned(unsigned n)
{
  unsigned r = 1;
  for (; n > 0; n--)
    r *= n;
  return r;
}

unsigned long fact_unsigned_long(unsigned long n)
{
  unsigned long r = 1;
  for (; n > 0; n--)
    r *= n;
  return r;
}
```

```
fact_int(20): -2102132736
fact_unsigned(20): 2192834560
fact_unsigned_long(20): 2432902008176640000
```

Warning: overflow for signed integers is undefined behavior

Code Sample

A Simple Program

```c
#include <stdio.h>
#include <stdlib.h>

unsigned long fact(unsigned long n)
{
  unsigned long r = 1;
  for (; n > 0; n--)
    r *= n;
  return r;
}
```

```c
int main()
{
  unsigned long r;
  r = fact(0);
  printf("fact( 0) = %19lu", r);
  if (r == 1) printf(" OK\n");
  else printf(" KO\n");
  for (unsigned long n = 1; n < 21; n++) {
    unsigned long tmp = fact(n);
    printf("fact(%2lu) = %19lu", n, tmp);
    if (tmp == r * n) printf(" OK\n");
    else {
      printf(" KO\n");
      return 1;
    }
    r = tmp;
  }
  return 0;
}
```

A Simple Program

```
shell> make fact
gcc -Wall -Wextra -std=c99 -O3 fact.c -o fact
shell> ./fact
fact( 0) =                   1 OK
fact( 1) =                   1 OK
fact( 2) =                   2 OK
fact( 3) =                   6 OK
fact( 4) =                  24 OK
fact( 5) =                 120 OK
fact( 6) =                 720 OK
fact( 7) =                5040 OK
fact( 8) =               40320 OK
fact( 9) =              362880 OK
fact(10) =             3628800 OK
fact(11) =            39916800 OK
fact(12) =           479001600 OK
fact(13) =          6227020800 OK
fact(14) =         87178291200 OK
fact(15) =       1307674368000 OK
fact(16) =      20922789888000 OK
fact(17) =     355687428096000 OK
fact(18) =    6402373705728000 OK
fact(19) =  121645100408832000 OK
fact(20) = 2432902008176640000 OK
```

Readable ?

```
unsigned long fact(unsigned long n)
{
  unsigned long r = 1;
  for (; n > 0; r *= n--);
  return r;
}
```

Security System

Laboratory of Epita

Bibliography

[1]: Kernighan and Plauger, The Elements of Programming Style by Brian W. Kernighan, P. J. Plauger , ISBN: 0070342075

[2]: Donald E. Knuth, Structured Programming with Goto Statements. Computing Surveys 6:4 (December 1974), pp. 261–301, §1.