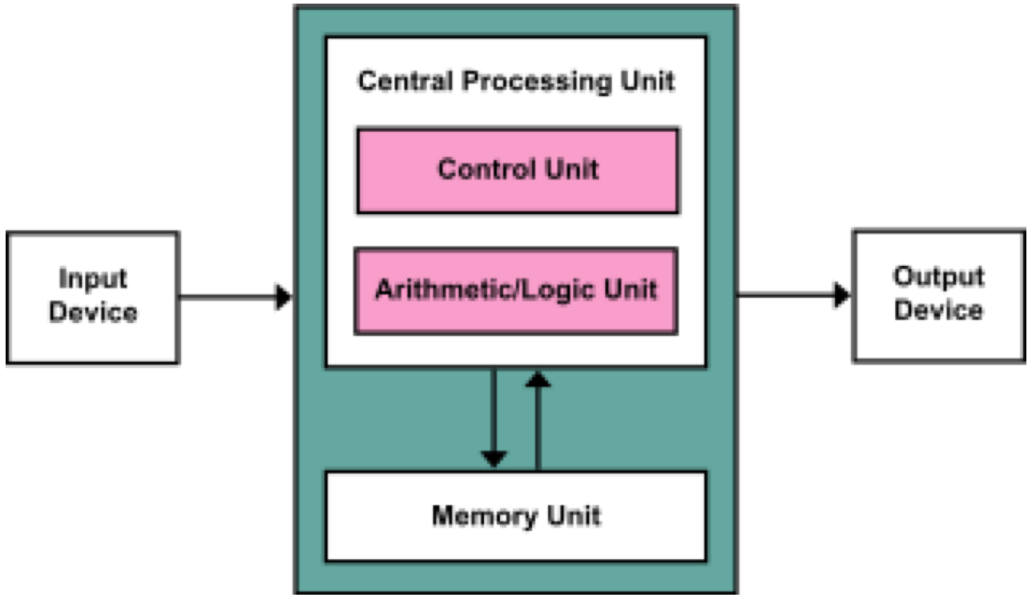
Systèmes d'exploitation

Introduction au language C

Architecture



Modelle Von Neumann

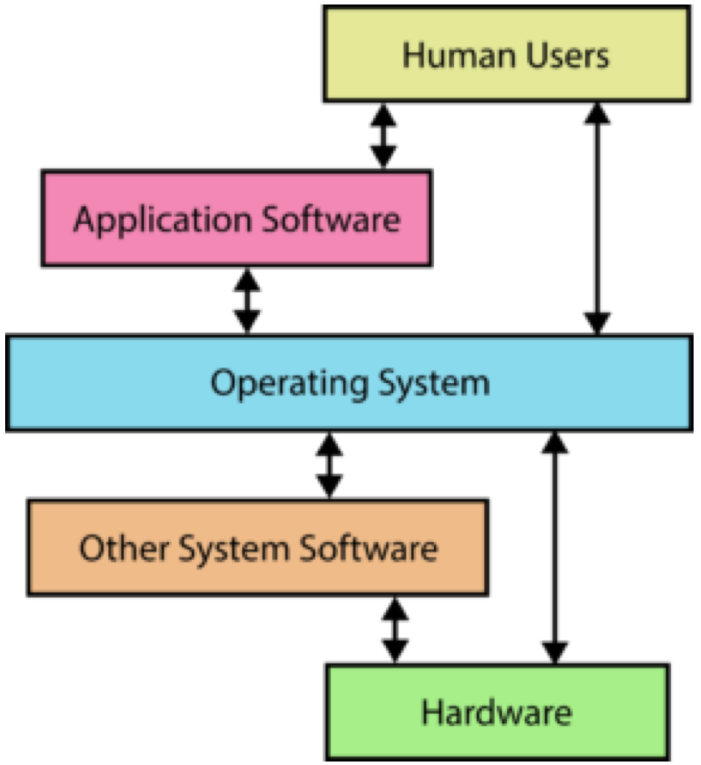
Questions

- Comment executer plus d'un programme à la fois?
- Comment partager de la memoire?
- Comment intéragir avec différents périphériques (IO Device)?
- ...

Les systèmes d'exploitation

Ils nous permettent de plus facilement executer des programmes.

On peut y penser comme une couche d'abstraction sur le matérielle.



Systèmes populaires

- Windows
- Unix
- Linux
- RTOS

Concepts des systèmes d'exploitations

Virtualization du CPU

- Processus et threads
- Ordonnancement (Sheduling)
- ...

Concurrence

- Primitives de concurrence (Locks, Condition Variables, Semaphores, …)
- …

Virtualization de la memoire

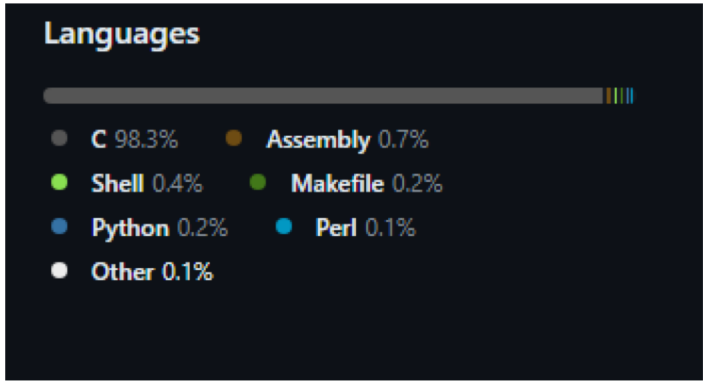
- Espace d'adressage (Address Space)
- Translation d'adresses (Address Translation)
- API de mémoire
- ...

Persistance

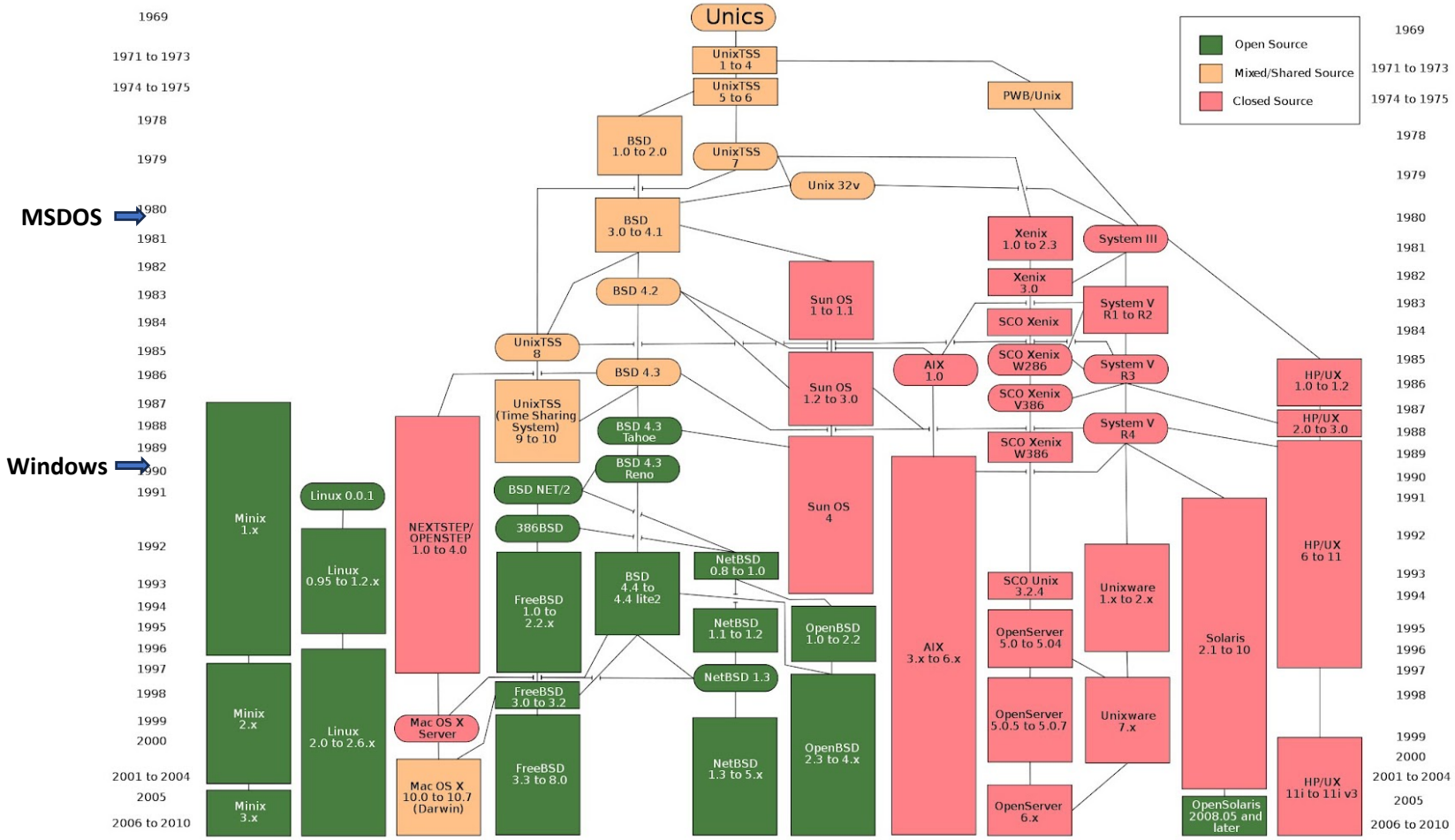
- Périphériques (IO Devices)
- Systèmes de fichiers
- ...

Unix et C

- SE écrit en assembleur était trop complex (Ex: MULTICS).
- Unix est le premier SE écrit majoritairement en C.
- Majorité des SE aujourd'hui son écrit en C (et un peu d'assembleur).



torvalds/linux: Linux kernel source tree (github.com)



History of Unix - Wikipedia

Pourquoi C?

Quelqu'un qui n'aime pas C:

- Vieux (Pas un language moderne!)
- Simple (Pas de cool features)
- Gestion de mémoire manuelle (Dangeureux)
- Pointeurs (Dangeureux)
- Undefined Behavior (Bugs!)
- ...

Quelqu'un qui aime C:

- Vieux (Compatible et prouvé)
- Simple (Facile a comprendre)
- Gestion de mémoire manuelle (Controle)
- Pointeurs (Controle)
- Undefined Behavior (Optimizations!)
- ...

C'est une question de perspective et utilité pour une communauté.

Web? Booo.
Sytèmes? Yey.

Pourquoi C?

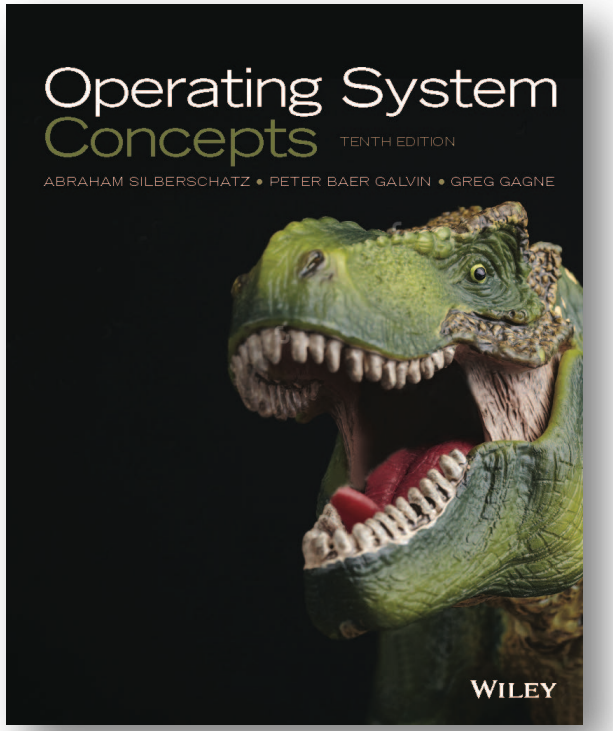
Linus Torvalds (Initiateur de Linux)

- "I like **interacting with hardware from a software prospective**."
- "Its not just that you can use C to generate good code for hardware. Its that, if you **think like a computer** writing C actually makes sense. "
- "When I read C, I **know what the assembly language will look like**, and that’s something I care about."

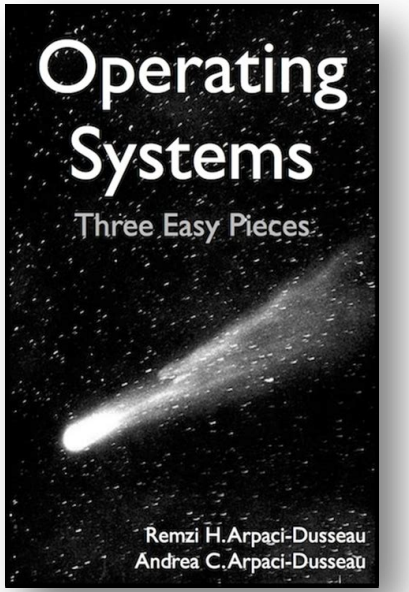
Linus Torvalds "Nothing better than C" (youtube.com)

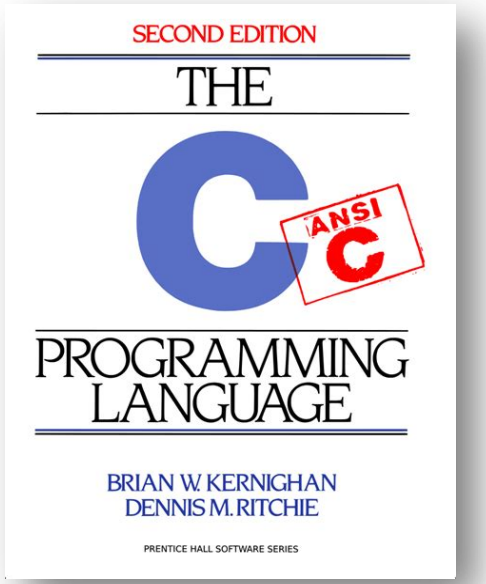
Recommendations

Manuelle du cour



Autres





Le language C

Introduction pour le cour de système d'exploitation

Manuelle de référence: C language - cppreference.com

Hello World

```c
#include <stdio.h>

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

Variables

```
int grade = 95;
grade = 98; // After convincing the teacher

float myfloat = 3.14159;

char myletter = 'A';
```

Type specifier	Equivalent type	Width in bits by data model				
		C standard	LP32	ILP32	LLP64	LP64
char	char	at least 8	8	8	8	8
signed char	signed char					
unsigned char	unsigned char					
short	short int	at least 16	16	16	16	16
short int						
signed short						
signed short int						
unsigned short	unsigned short int					
unsigned short int						
int	int	at least 16	16	32	32	32
signed						
signed int						
unsigned	unsigned int					
unsigned int						
long	long int	at least 32	32	32	32	64
long int						
signed long						
signed long int						
unsigned long	unsigned long int					
unsigned long int						
long long	long long int (C99)	at least 64	64	64	64	64
long long int						
signed long long						
signed long long int						
unsigned long long	unsigned long long int (C99)					
unsigned long long int						

Types de base

Bool? 0 = false, !0 = true

`1` == `sizeof(char)` ≤ `sizeof(short)` ≤ `sizeof(int)` ≤ `sizeof(long)` ≤ `sizeof(long long)`.

Tableaux

```
int n[10]; // integer constants are constant expressions
char o[sizeof(double)]; // sizeof is a constant expression


int a[5] = {1,2,3}; // declares int[5] initialized to 1,2,3,0,0
char str[] = "abc"; // declares char[4] initialized to 'a','b','c','\0'
```

Les tableau peuvent vivre sur la stack! C'est pas comme new int[100] en Java.

Pointeurs

```
float *p; // p is a pointer to float
float **pp; // pp is a pointer to a pointer to float
int (*fp)(int); // fp is a pointer to function with type int(int)
```

Les pointeurs sont simplement des types qui contiennes des **adresses** comme valeur.

On peut faire de l'**arithmetique** sur des pointeurs comme si c'était des entiers.

Operator	Operator name	Example	Description
`[]`	array subscript	`a[b]`	access the **b**th element of array **a**
`*`	pointer dereference	`*a`	dereference the pointer **a** to access the object or function it refers to
`&`	address of	`&a`	create a pointer that refers to the object or function **a**
`.`	member access	`a.b`	access member **b** of struct or union **a**
`->`	member access through pointer	`a->b`	access member **b** of struct or union pointed to by **a**

p[i] c'est du sucre syntaxique pour: *(p + i)

Qualifieurs de mutabilité

Augmente la lisibilité, et parfois la performance.

```
int a = 5; // Mutable
const float PI = 3.1415926; // Immutable

const int* p = &a; // Immutable, pointee mutable
*p = 6; // OK
const float* const q = &PI; // Immutable, pointee immutable
*q = 3.14; // Error
```

String literals

"Hello" construit une string intégré dans le code source!

```
char* p = "Hello";
p[1] = 'M'; // Undefined behavior
char a[] = "Hello";
a[1] = 'M'; // OK: a is not a string literal
char* p2 = "Hello";

char* p = "Hello";
char a[] = "Hello";
char* p2 = "Hello";
char[] a2 = "Hello";

p == "Hello"; // true
p == p2; // ?
a == "Hello"; // ?
p == a; // ?
```

X86-64

```
.L.str:
        .asciz  "Hello"
p:
        .quad   .L.str
a:
        .asciz  "Hello"
p2:
        .quad   .L.str
a2:
        .asciz  "Hello"
```

Expressions

Constants and literals (e.g. `2` or `"Hello, world"`)

Suitably declared identifiers (e.g. `n` or `printf`)

Common operators						
assignment	increment decrement	arithmetic	logical	comparison	member access	other

`a = b`
`a += b`
`a -= b`
`a *= b`
`a /= b`
`a %= b`
`a &= b`
`a |= b`
`a ^= b`
`a <<= b`
`a >>= b`
	`++a`
	`--a`
	`a++`
	`a--`
		`+a`
		`-a`
		`a + b`
		`a - b`
		`a * b`
		`a / b`
		`a % b`
		`~a`
		`a & b`
		`a | b`
		`a ^ b`
		`a << b`
		`a >> b`
			`!a`
			`a && b`
			`a || b`
				`a == b`
				`a != b`
				`a < b`
				`a > b`
				`a <= b`
				`a >= b`
					`a[b]`
					`*a`
					`&a`
					`a->b`
					`a.b`
						`a(...)`
						`a, b`
						`(type) a`
						`a ? b : c`
						`sizeof`
						
						`_Alignof`
						(since C11)

Commandes conditionnelles

```c
int value = 10;
if (value < 5) {
  printf("a");
} else {
  printf("b");
}

int value = 50;
if (value < 10) {
  printf("x");
} else if (value < 40) {
  printf("y");
} else {
  printf("z");
}
```

Commandes conditionnelles

```
int day = 4; // 0-6, 0 is Sunday

switch (day) {
  case 2: { // Tuesday
    printf("Demo!"); printf("Cour!"); break;
  }
  case 4: printf("Cour!"); break; // Thursday
  default: printf("Rien :(");
}

switch (day) {
  case 2: printf("Demo!"); // Fallthrough
  case 4: printf("Cour!"); break;
  default: printf("Rien :(");
}
```

Switch est souvent plus efficase que if-else!

Commandes itératives

```c
int i = 0;
while (i < 5) {
  printf("%d\n", i);
  i++;
}

int i = 0;
do {
  printf("%d\n", i);
  i++;
} while (i < 5);

for (int i = 0; i < 5; i++) {
  printf("%d\n", i);
}
```

Sauts non-conditionnelles

```
goto mylabel; // Jump to mylabel
mylabel:;

while (/* ... */) {
   // ...
   continue; // acts as goto contin;
   // ...
   contin:;
}

while (/* ... */) {
   // ...
   break; // acts as goto brk;
   // ...
}
brk:;
```

GOTO est généralement évité!

Déclarations et définitions de fonctions

```
int max(int a, int b); // Declaration of max function
void foo(); // Declaration of foo function


int max(int a, int b) // Definition of max function
{
  if(a > b){
    return a;
  }
  else {
    return b;
  }
}
```

Structures

```
struct point {
   int x;
   int y;
};

struct point p = { 1, 2 };

p.x = 3;
p.y = 7;

struct point *pp = &p;

(*pp).x = 4;
pp->x = 4; // equivalent to (*pp).x = 4;

struct point
{
  int x;
  int y;
} p = { 1, 2 };
```

Déclaration de type (Type alias)

```c
typedef int int_t;
typedef char* string_t;

int_t x;
string_t s;

typedef struct tnode node;

struct tnode {
    int count;
    node *left, *right; // tnode cant be used here, because it is not defined yet
                        // but we can use node, because it is defined before
};
node s;

typedef struct
{
  double hi, lo;
} range;

range x;
```

Gestion de mémoire dynamique

```
int *p1 = malloc(4*sizeof(int));  // allocates enough for an array of 4
int
int *p2 = malloc(sizeof(int[4])); // same, naming the type directly

struct t { int a[5]; };

struct t *p3 = malloc(sizeof(struct t)); // allocates enough for struct t

free(p1);
free(p2);
free(p3);
```

Autres fonctions: calloc, realloc

Dynamic memory management - cppreference.com

Manipulation et examination de strings

```
const char *src = "Take the test.";
//  src[0] = 'M' ; // this would be undefined behavior
char dst[strlen(src) + 1]; // +1 to accommodate for the null terminator
strcpy(dst, src);

int string_equals(char* lhs, char* rhs) {
  // lhs == rhs; // this would compare the pointers
  return strcmp(lhs, rhs) == 0;
}

char str[50] = "Hello ";
char str2[50] = "World!";
strcat(str, str2);
strcat(str, " ...");
strcat(str, " Goodbye World!");
```

Null-terminated byte strings - cppreference.com

Préprocesseur

```c
// Include
#include <stdio.h> // Include standard library
#include "myheader.h" // Include local header

// Remplacement de texte
#define MY_FAV_NUMBER 2
#undef MY_FAV_NUMBER
#define MY_FAV_NUMBER 1

#define str(x) #x
str(hello); // = "hello"
#define CONCAT(x,y) x##y
CONCAT(hello,world); // = helloworld

// Compilation conditionel
#ifdef PLATFORM_WINDOWS
  // Code pour windows
#elif PLATFORM_LINUX
  // Code pour linux
#elif PLATFORM_MACOS
  // Code pour macos
#else
  // Code pour autre
#endif
```

Fuite de mémore / Memory Leak

Se produit lorsqu'on est **incapable de libérer la mémoire** ou on ne le fait simplement pas.

```
void leak1()
{
    int *ptr = (int *)malloc(sizeof(int));
    // Leak
}

void leak2()
{
    int *ptr = (int *)malloc(sizeof(int));
    ptr = (int *)malloc(sizeof(int)); // Leak
    free(ptr);
}
```

Undefined Behavior

UB: Le compilateur suppose qu'une opération ne se produira jamais.
Si cette opération se produit, cela relève du comportement indéfini:

Présence de UB: n'importe quoi peut se produire, que ce soit
un fonctionnement correct,
un plantage ou une altération silencieuse des données.

Invalid read or write

```
int x;
if (x > 0) { // UB: true or false?
  // ...
}

int a[10] { 0 }; // Tableau de 10 entiers
initialisés à 0
a[10] = 1; // UB: out-of-bounds access
```

Le compilateur assume qu'un access memoire est valide.

Signed overflow

Source

```
int foo(int x)
{
    return x + 1 > x;
}
```

Output (x86)

```
foo:
        mov     eax, 1
        ret
```

Le compilateur assume que x + 1 ne sera jamais plus grand que 2147483647

S'applique pour les autres entiers aussi.

Compiler Explorer (godbolt.org)

Null pointer deference

Source

```
int foo(int* p)
{
    int x = *p;
    if (!p)
        return x;
    else
        return 0;
}
```

Output (x86)

```
foo:
        xor     eax, eax
        ret
```

Compilateur assume qu'une déréferenciation n'est jamais sur un pointeur null.

[Compiler Explorer (godbolt.org)]

Structure d'un projet C

Includes

On fait un fichier pour les déclarations (Header file), et un fichier pour les définitions (Source file).

foobar.h

```c
#ifndef _FOOBAR_H_
#define _FOOBAR_H_

int foo(int a, int b);
int bar(int a, int b);

#endif
```

foobar.c

```c
#include "foobar.h"

int foo(int a, int b) {
    return a + b;
}

int bar(int a, int b) {
    return a - b;
}
```

Linker

Le linker est un programme utilisé dans la compilation des programmes. Son rôle est de lier ou de combiner différents fichiers objets générés par le compilateur en un seul programme exécutable.

Vous allez probablement faire face à des erreurs de linker:

- **Undefined reference:** Le linker ne peut pas trouver l'implémentation d'une fonction utilisée dans le code.
- **Multiple definitions:** Il y a une ambiguïté et le linker ne peut pas choisir.

Build Systems

- Permet de specifier comment crée nos programes.
- Fichers sources, libraries, executables, etc...
- Plusieurs systèmes différents: Make, MSBuild, Ninja, …
- Build system generators: **Cmake**

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.11)
project(TP0 C)

set(CMAKE_C_STANDARD 99)

add_executable(TP0 main.c main.h)
```