

# **Compilation**

---

Jean Goubault-Larrecq

LSV/CNRS UMR 8643 & ENS Cachan

## Plan

---

1. L'idée de la compilation un exemple en C.
2. L'optimisation l'exemple, suite.
3. Autres langages. Un langage fonctionnel simple et sa sémantique.

## Retour sur cat

---

Voici une façon (naïve) de programmer cat, en C:

```
main (int argc, char *argv[])           // ex: pour 'cat a b', argc=3,
{                                         // argv[0]="cat", argv[1]="a",
    int i, c;                             // argv[2]="b"

    for (i=1; i<argc; i++)               // pour chaque nom de fichier
    {
        FILE *f;

        f = fopen (argv[i], "r");        // ouvrir le fichier
        while ((c = fgetc (f))!=EOF)     // recuperer caractere suivant
            fputc (c, stdout);           // afficher le caractere
        fclose (f);                       // fermer le fichier
    }
    fflush (stdout);                       // *vraiment* afficher...
    exit (0);
}
```

## Compilons ça pour voir

---

```
gcc -ggdb cat.c
```

```
        ; main (int argc, char *argv[]) {
        ;   int i, c; FILE *f;   // 3x4 octets
0x8048150 <main>:      pushl   %ebp
0x8048151 <main+1>:    movl    %esp,%ebp
0x8048153 <main+3>:    subl   $0xc,%esp
        ; i=1
0x8048157 <main+7>:    movl   $0x1,-4(%ebp)
        ; while (i<argc)
0x804815e <main+14>:   movl   -4(%ebp),%eax
0x8048161 <main+17>:   cmpl   %eax,0x8(%ebp)
0x8048164 <main+20>:   jg     0x8048170 <main+32>
0x8048166 <main+22>:   jmp    0x80481f0 <main+160>
        ; empiler "r":   (gdb) x/s 0x8059d88
        ;                   0x8059d88 <_fini+8>:   "r"
0x8048170 <main+32>:   pushl  $0x8059d88
        ; calcul de argv[i]
```

```

0x8048175 <main+37>:   movl   -4(%ebp),%eax
0x8048178 <main+40>:   movl   %eax,%edx
0x804817a <main+42>:   leal   0x0(,%edx,4),%eax
                   ; %eax:=%edx*4
0x8048181 <main+49>:   movl   0xc(%ebp),%edx
0x8048184 <main+52>:   movl   (%edx,%eax,1),%eax
                   ; %eax:=mem(%edx+%eax)
0x8048187 <main+55>:   pushl  %eax
                   ; f = fopen (argv[i], "r")
0x8048188 <main+56>:   call   0x8048330 <fopen>
                   ; resultat dans %eax
0x804818d <main+61>:   addl   $0x8,%esp
0x8048190 <main+64>:   movl   %eax,%eax           ; ?
0x8048192 <main+66>:   movl   %eax,-12(%ebp)
                   ; c = fgetc (f)
0x8048195 <main+69>:   movl   -12(%ebp),%eax
0x8048198 <main+72>:   pushl  %eax
0x8048199 <main+73>:   call   0x80483a0 <fgetc>
0x804819e <main+78>:   addl   $0x4,%esp
0x80481a1 <main+81>:   movl   %eax,%eax           ; ?

```

```

0x80481a3 <main+83>:    movl    %eax,-8(%ebp)
                    ; if (c!=EOF)
0x80481a6 <main+86>:    cmpl    $-1,-8(%ebp)      ; EOF=-1
0x80481aa <main+90>:    jne     0x80481b0 <main+96>
0x80481ac <main+92>:    jmp     0x80481d0 <main+128>
                    ; fputc (c, stdout)
0x80481b0 <main+96>:    pushl  $0x805ed8c        ; =stdout
0x80481b5 <main+101>:   movl    -8(%ebp),%eax
0x80481b8 <main+104>:   pushl  %eax
0x80481b9 <main+105>:   call   0x80483f0 <fputc>
0x80481be <main+110>:   addl   $0x8,%esp
                    ; et on boucle (boucle interne)
0x80481c1 <main+113>:   jmp     0x8048195 <main+69>
                    ; ici on sort de la boucle interne
                    ; fclose (f)
0x80481d0 <main+128>:   movl    -12(%ebp),%eax
0x80481d3 <main+131>:   pushl  %eax
0x80481d4 <main+132>:   call   0x8048220 <_IO_fclose>
0x80481d9 <main+137>:   addl   $0x4,%esp
                    ; i++

```

```
0x80481dc <main+140>:   incl    -4(%ebp)
                   ; et on boucle (boucle externe)
0x80481df <main+143>:   jmp     0x804815e <main+14>
                   ; ici on sort de la boucle externe
                   ; fflush (stdout)
0x80481f0 <main+160>:   pushl   $0x805ed8c           ; =stdout
0x80481f5 <main+165>:   call    0x80482c0 <fflush>
                   ; exit (0)
0x80481fa <main+170>:   addl    $0x4,%esp
0x80481fd <main+173>:   pushl   $0x0
0x80481ff <main+175>:   call    0x8048540 <exit>
0x8048204 <main+180>:   addl    $0x4,%esp
                   ; et on retourne de main ():
0x8048210 <main+192>:   movl    %ebp,%esp
0x8048212 <main+194>:   popl    %ebp
0x8048213 <main+195>:   ret
```

## Comment ça marche

---

- Le compilateur **réserve** de la place pour chaque variable locale sur la pile `%esp`; garde l'adresse du `%esp` initial dans `%ebp`.
- Compilation de  $x=e$ :
  - par récurrence sur la structure de  $e$ , on calcule la valeur de  $e$  dans un registre (par ex. `%eax`);
  - on compile `mov%eax,⟨endroit réservé pour  $x$ ⟩`;
  - Défaut: produit des déplacements de valeurs **redondants**.
- Tests, boucles `while` et `for`: via `cmp`, `jge`, etc.
- Appels de fonction: empiler les arguments, `call`, dépiler; résultat dans `%eax`.



## Exercices

---

- En supposant que  $i = -12(\%ebp)$ ,  $argc = 8(\%ebp)$ ,  $argv = 12(\%ebp)$ ,  $len = -4(\%ebp)$ , proposer des suites d'instructions pour:
  - $len = 0$
  - $i++$  (incrémenter  $i$ )
  - $len += \dots$ , où  $\dots$  est dans  $\%eax$ .
- En déduire la forme compilée de:

```
len = 0;
for (i=1; i<argc; i++) len += strlen (argv[i]);
return len;
```

## Bootstrap

---

- Le compilateur `gcc` est lui-même un **programme**.
- En fait, `gcc` est écrit en C et compilé... avec `gcc`
  - ... permet en particulier de tester `gcc` sur un exemple non trivial.
  - ... plus vicieux, on peut tester si `gcc (gcc)=gcc`.