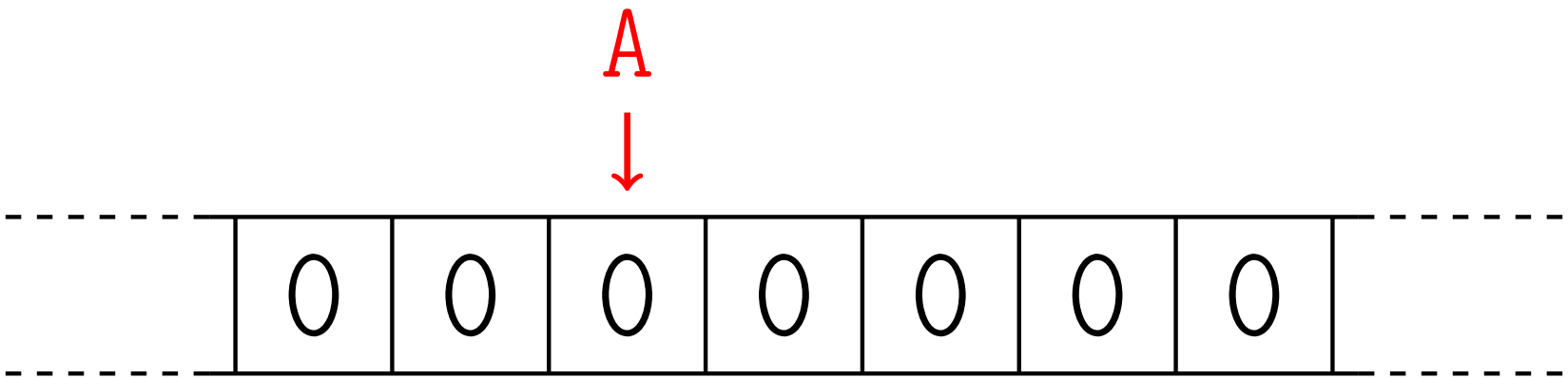
Les langages informatiques

Palais de la Découverte

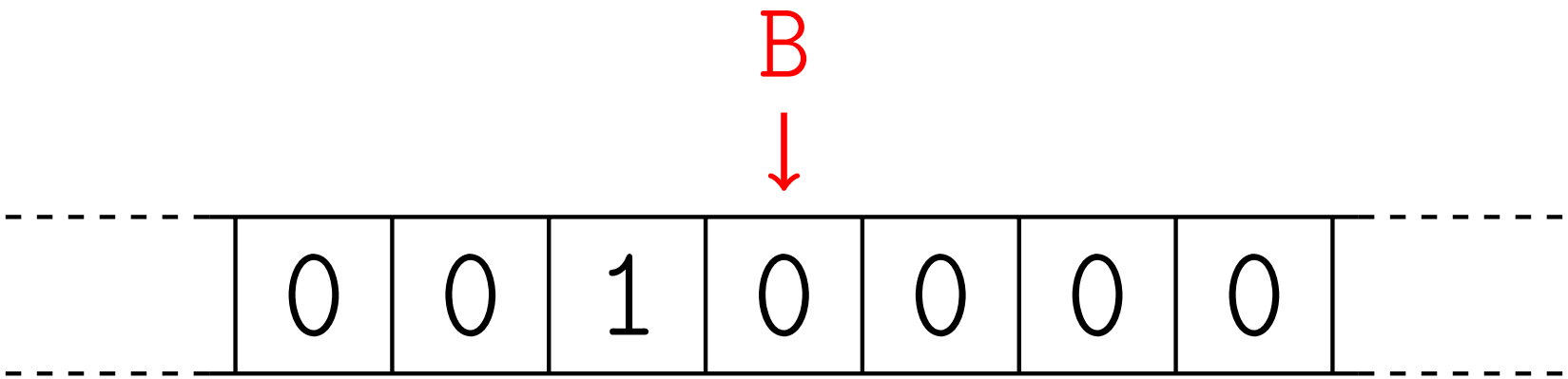
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



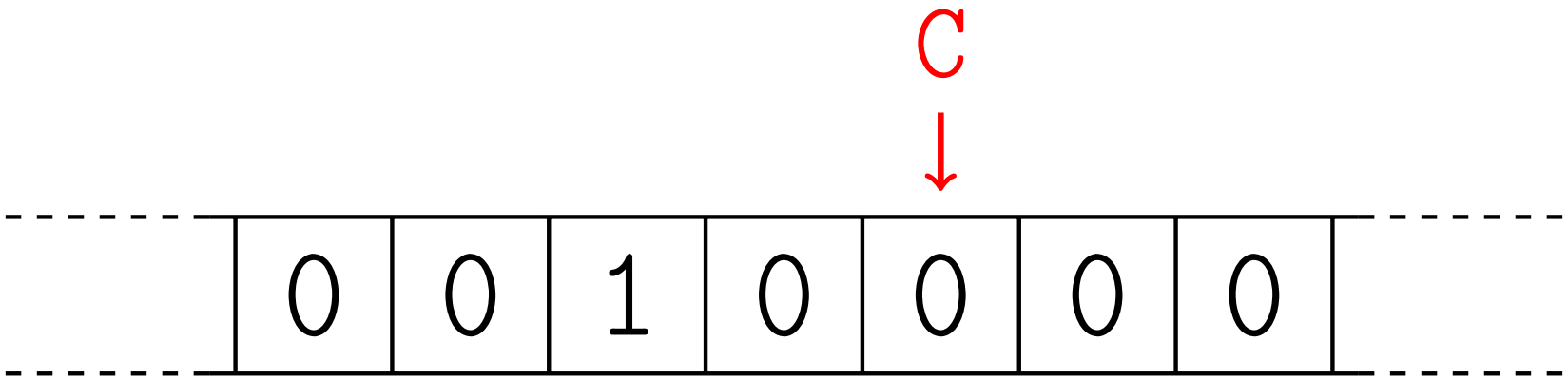
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



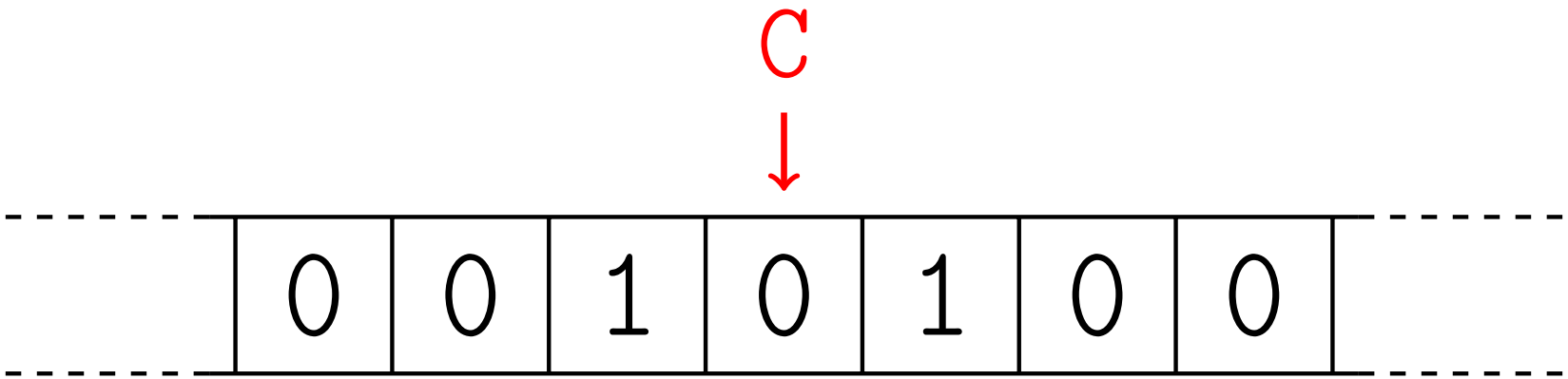
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



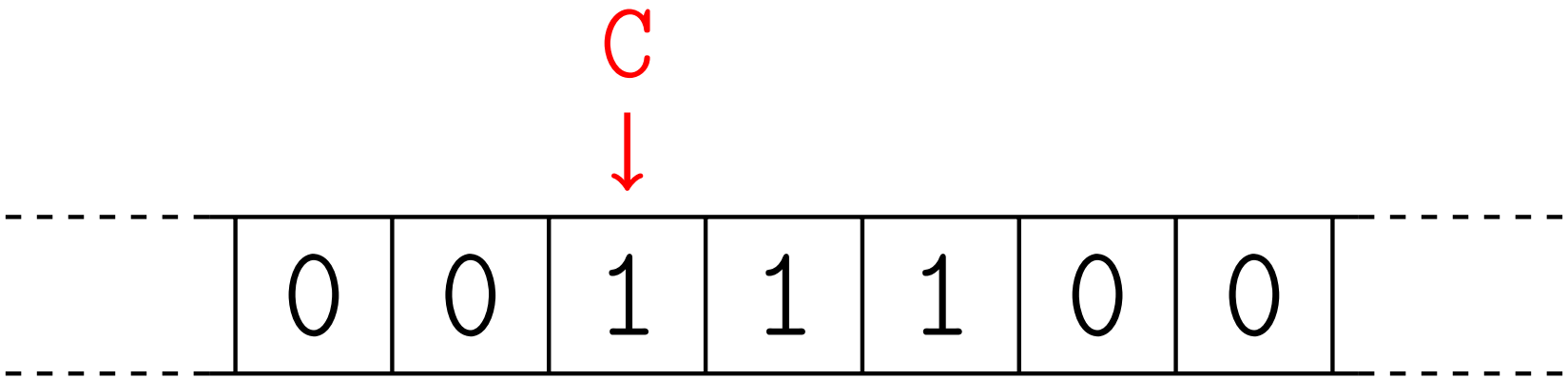
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



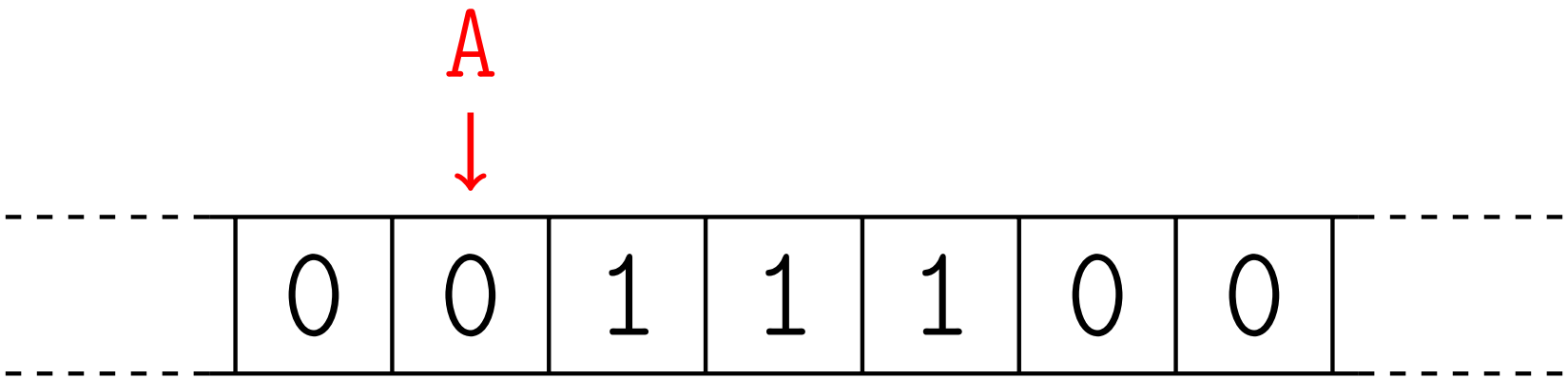
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



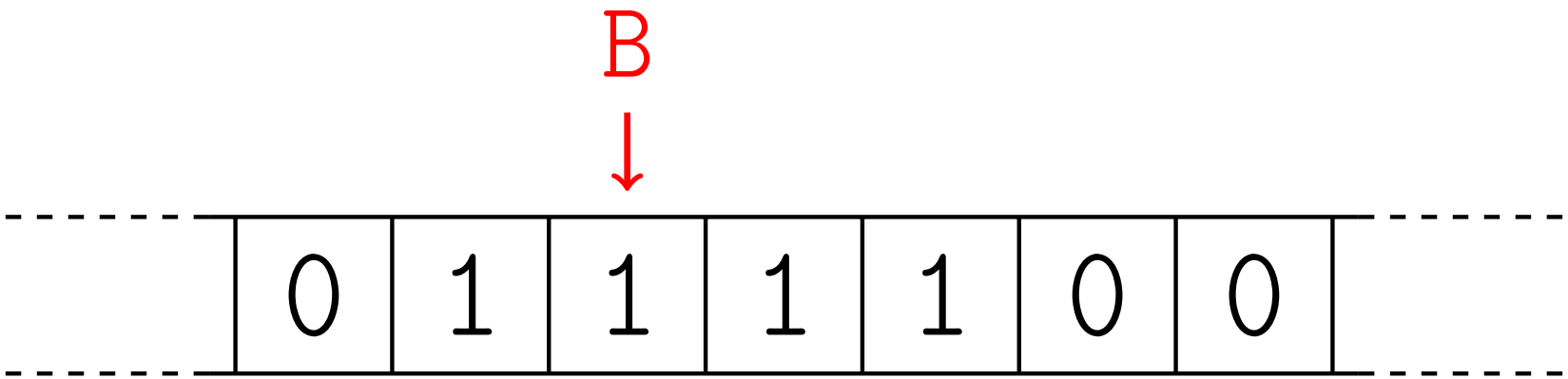
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



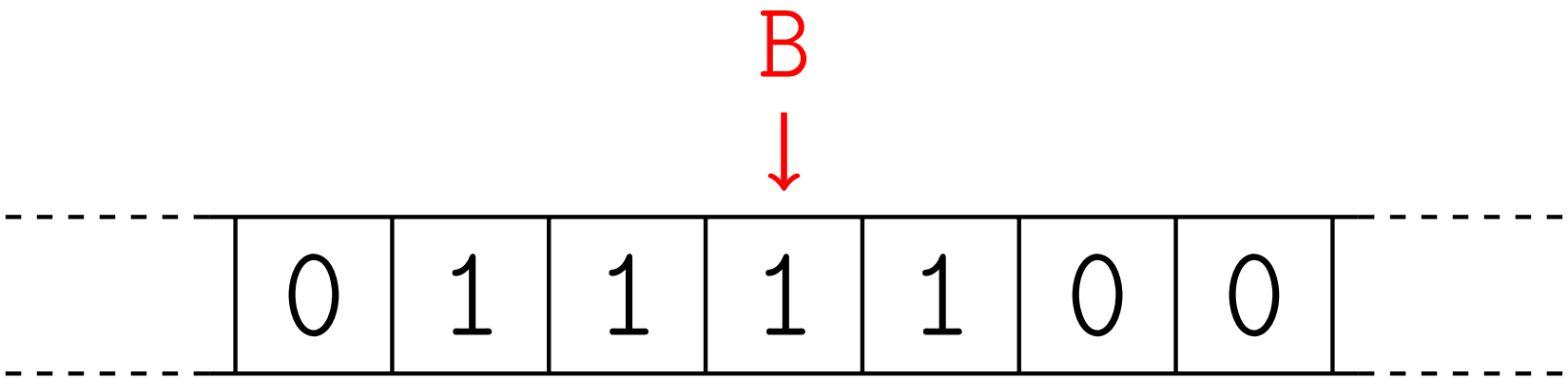
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



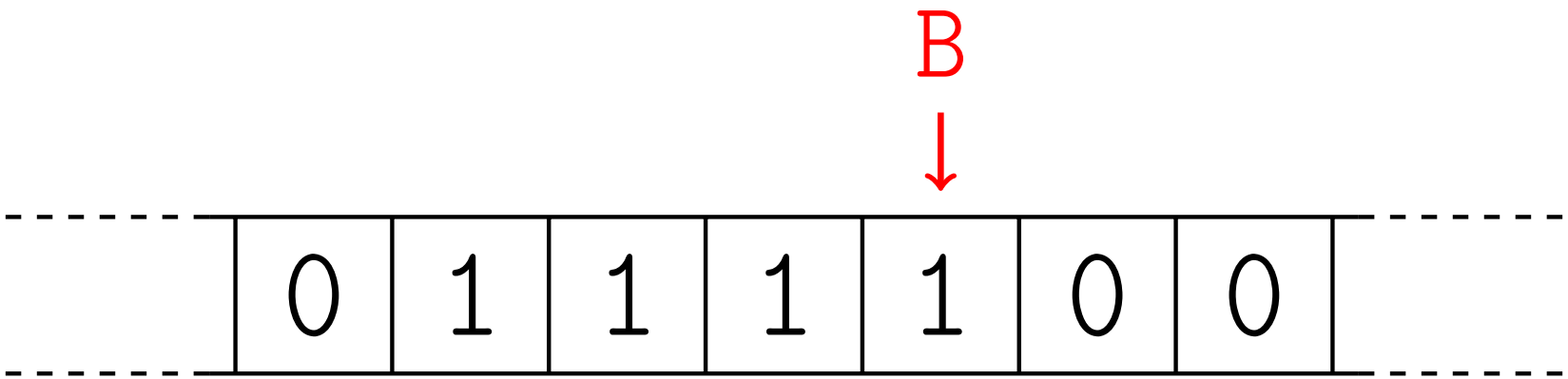
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



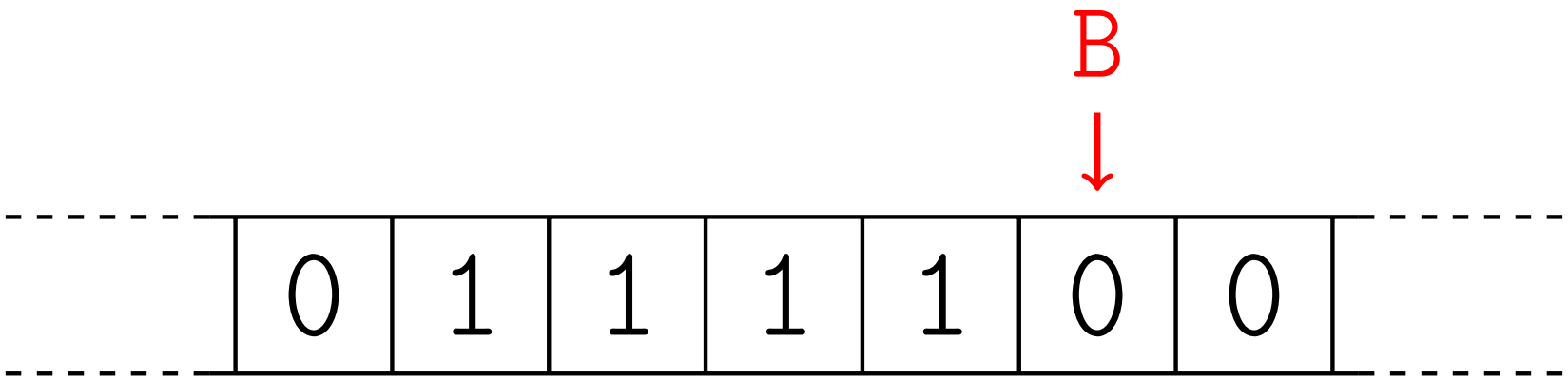
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



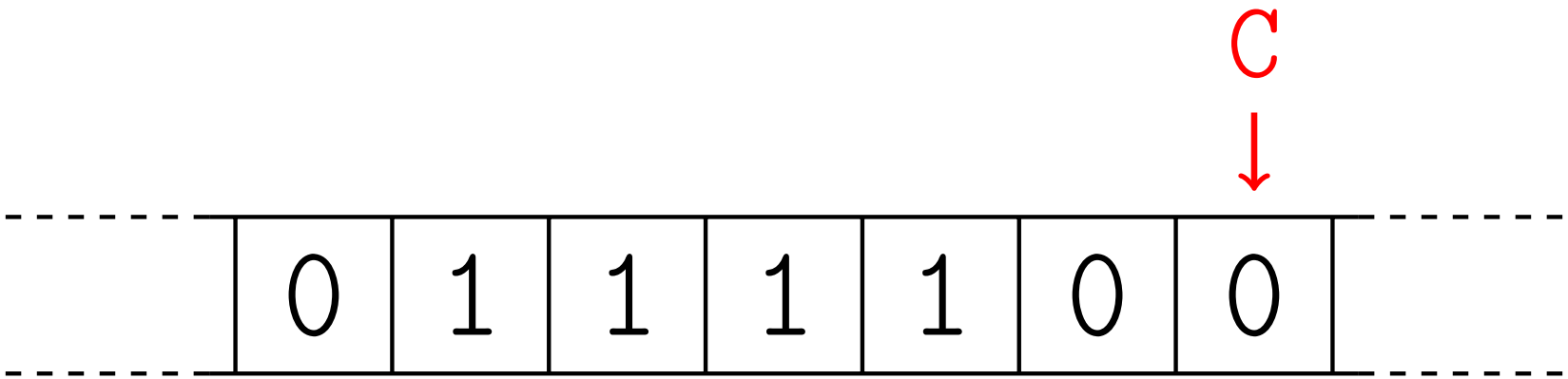
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



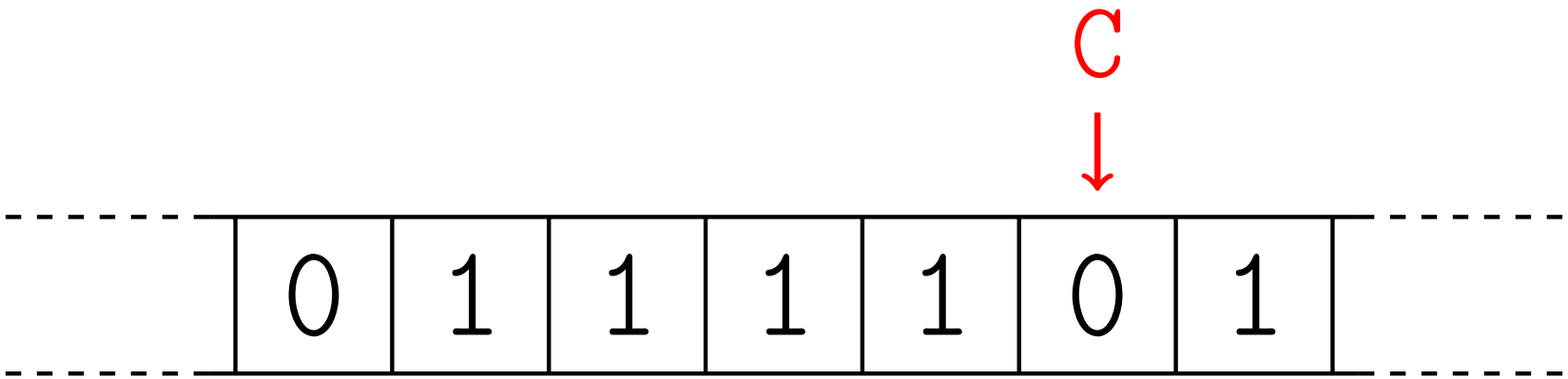
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



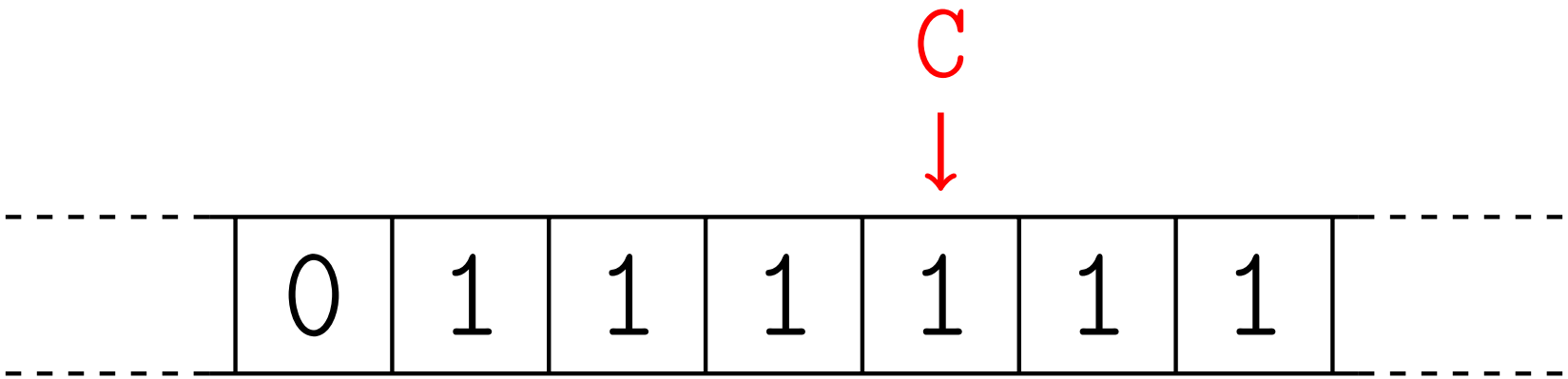
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



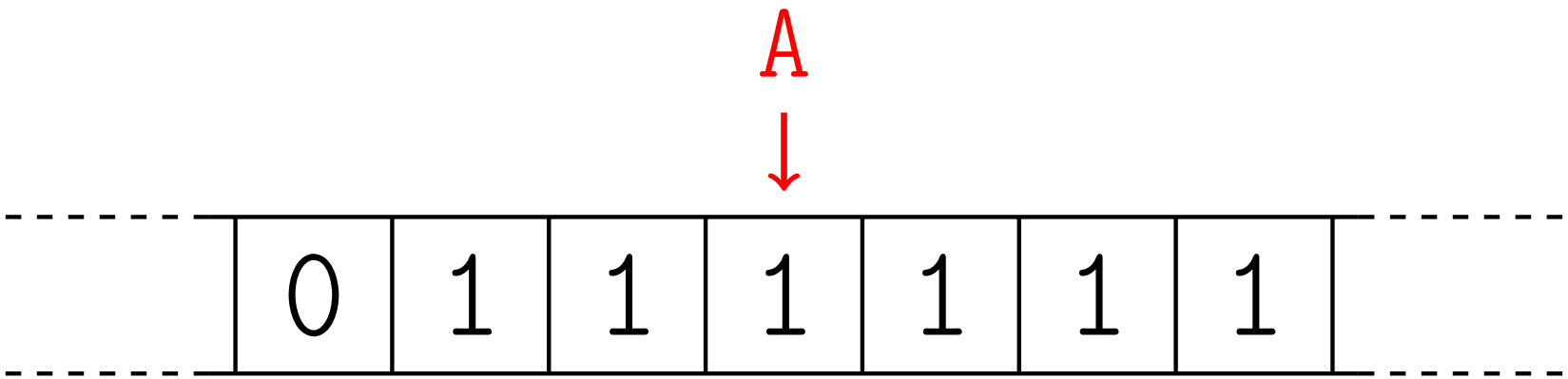
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



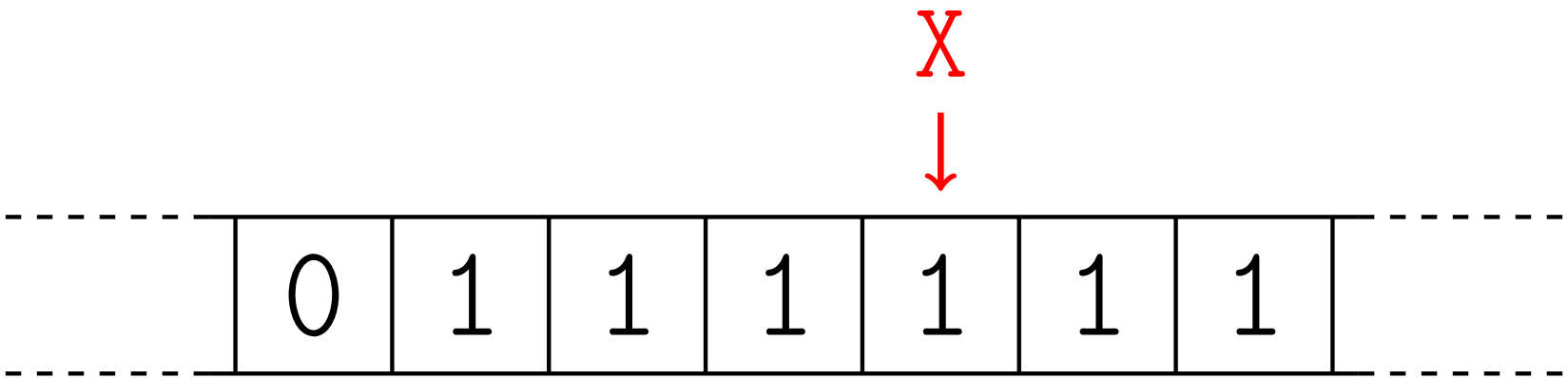
La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



La machine de Turing

I	A	B	C
0	1, →, B	0, →, C	1, ←, C
1	1, →, X	1, →, B	1, ←, A



Le lambda-calcul de Church

$$(\lambda xy.axbyx)\ ((\lambda z.zz)\ c)\ d$$

Le lambda-calcul de Church

$$(\lambda xy.axbyx)\ ((\lambda z.zz)\ c)\ d$$

$$(\lambda xy.axbyx)\ ((\lambda \mathbf{z}.\mathbf{zz})\ c)\ d$$

Le lambda-calcul de Church

$$(\lambda xy.axbyx)\ ((\lambda z.zz)\ c)\ d$$

$$(\lambda xy.axbyx)\ ((\lambda \mathbf{z}.\mathbf{zz})\ c)\ d$$

$$(\lambda xy.axbyx)\ (cc)\ d$$

Le lambda-calcul de Church

$$(\lambda xy.axbyx)\ ((\lambda z.zz)\ c)\ d$$

$$(\lambda xy.axbyx)\ ((\lambda \mathbf{z}.\mathbf{zz})\ c)\ d$$

$$(\lambda xy.axbyx)\ (cc)\ d$$

$$(\lambda \mathbf{x}y.a\mathbf{x}by\mathbf{x})\ (cc)\ d$$

Le lambda-calcul de Church

$$(\lambda xy.axbyx)\ ((\lambda z.zz)\ c)\ d$$

$$(\lambda xy.axbyx)\ ((\lambda \mathbf{z}.\mathbf{zz})\ c)\ d$$

$$(\lambda xy.axbyx)\ (cc)\ d$$

$$(\lambda \mathbf{x}y.a\mathbf{x}by\mathbf{x})\ (cc)\ d$$

$$(\lambda y.a(cc)by(cc))\ d$$

Le lambda-calcul de Church

$$(\lambda xy.axbyx)\ ((\lambda z.zz)\ c)\ d$$

$$(\lambda xy.axbyx)\ ((\lambda \mathbf{z}.\mathbf{zz})\ c)\ d$$

$$(\lambda xy.axbyx)\ (cc)\ d$$

$$(\lambda \mathbf{x}y.a\mathbf{x}by\mathbf{x})\ (cc)\ d$$

$$(\lambda y.a(cc)by(cc))\ d$$

$$(\lambda \mathbf{y}.a(cc)b\mathbf{y}(cc))\ d$$

Le lambda-calcul de Church

$$(\lambda xy.axbyx)\ ((\lambda z.zz)\ c)\ d$$

$$(\lambda xy.axbyx)\ ((\lambda \mathbf{z}.\mathbf{zz})\ c)\ d$$

$$(\lambda xy.axbyx)\ (cc)\ d$$

$$(\lambda \mathbf{x}y.a\mathbf{x}by\mathbf{x})\ (cc)\ d$$

$$(\lambda y.a(cc)by(cc))\ d$$

$$(\lambda \mathbf{y}.a(cc)b\mathbf{y}(cc))\ d$$

$$a(cc)bd(cc)$$

La thèse de Church-Turing

Quel langage exprime les meilleurs algorithmes ?

La thèse de Church-Turing

Quel langage exprime les meilleurs algorithmes ?

- Une seule machine de Turing simule tous les λ-termes...

La thèse de Church-Turing

Quel langage exprime les meilleurs algorithmes ?

- Une seule machine de Turing simule tous les λ-termes...
- ...un seul λ-terme simule toutes les machines de Turing.

La thèse de Church-Turing

Quel langage exprime les meilleurs algorithmes ?

- Une seule machine de Turing simule tous les λ-termes...
- ...un seul λ-terme simule toutes les machines de Turing.

Conclusion : Lambda-calcul $\Leftrightarrow$ Machines de Turing

La thèse de Church-Turing

Quel langage exprime les meilleurs algorithmes ?

- Une seule machine de Turing simule tous les λ-termes...
- ...un seul λ-terme simule toutes les machines de Turing.

Conclusion : Lambda-calcul $\Leftrightarrow$ Machines de Turing $\Leftrightarrow$ Autres

La thèse de Church-Turing

Quel langage exprime les meilleurs algorithmes ?

- Une seule machine de Turing simule tous les λ-termes...
- ...un seul λ-terme simule toutes les machines de Turing.

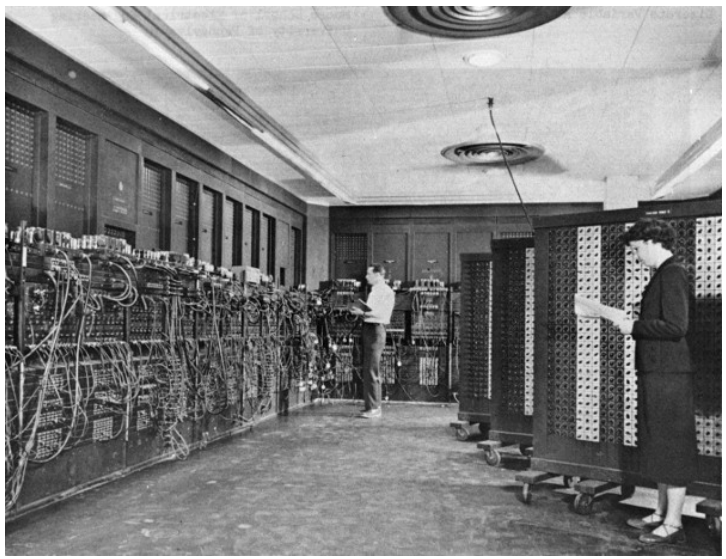
Conclusion : Lambda-calcul $\Leftrightarrow$ Machines de Turing $\Leftrightarrow$ Autres

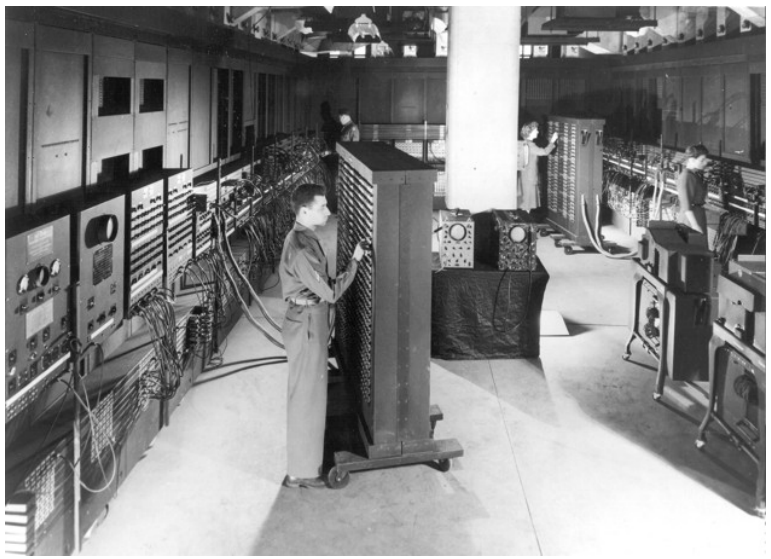
Thèse de Church-Turing

$\Rightarrow$ Tous les langages des algorithmes sont équivalents (...?)

Le code machine

Le code machine





Le langage assembleur

Le langage assembleur

```
0x0000178e <+28>:    mov    %rdx,0x8(%rax)
0x00001792 <+32>:    mov    -0x28(%rbp),%rax
0x00001796 <+36>:    mov    0x8(%rax),%rax
0x0000179a <+40>:    cmp    $0x7ffffffe,%rax
0x000017a0 <+46>:    jbe    0x17c1 <expand+79>
0x000017a2 <+48>:    lea    0x1876(%rip),%rcx
0x000017a9 <+55>:    mov    $0x84,%edx
0x000017ae <+60>:    lea    0x184f(%rip),%rsi
0x000017b5 <+67>:    lea    0x1851(%rip),%rdi
0x000017bc <+74>:    callq  0x1080 <__assert_fail@plt>
0x000017c1 <+79>:    mov    -0x28(%rbp),%rax
0x000017c5 <+83>:    mov    0x8(%rax),%rax
0x000017c9 <+87>:    shl    $0x3,%rax
0x000017cd <+91>:    mov    %rax,%rdi
0x000017d0 <+94>:    callq  0x1130 <malloc@plt>
```

Portabilité

Portabilité

Programme A $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1

Portabilité

Programme A $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
Programme A $\rightarrow$ Assembleur 2 $\rightarrow$ Machine 2

Portabilité

Programme A $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
Programme A $\rightarrow$ Assembleur 2 $\rightarrow$ Machine 2
Programme B $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1

Portabilité

Programme A $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
Programme A $\rightarrow$ Assembleur 2 $\rightarrow$ Machine 2
Programme B $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
Programme B $\rightarrow$ Assembleur 2 $\rightarrow$ Machine 2

Portabilité

Programme A $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
Programme A $\rightarrow$ Assembleur 2 $\rightarrow$ Machine 2
Programme B $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
Programme B $\rightarrow$ Assembleur 2 $\rightarrow$ Machine 2
Programme C $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1

Portabilité

$$
\begin{array}{ccccc}
\text{Programme A} & \rightarrow & \text{Assembleur 1} & \rightarrow & \text{Machine 1} \\
\text{Programme A} & \rightarrow & \text{Assembleur 2} & \rightarrow & \text{Machine 2} \\
\text{Programme B} & \rightarrow & \text{Assembleur 1} & \rightarrow & \text{Machine 1} \\
\text{Programme B} & \rightarrow & \text{Assembleur 2} & \rightarrow & \text{Machine 2} \\
\text{Programme C} & \rightarrow & \text{Assembleur 1} & \rightarrow & \text{Machine 1} \\
\ldots & \rightarrow & \ldots & \rightarrow & \ldots
\end{array}
$$

Portabilité

$$
\begin{array}{ccccc}
\text{Programme A} & \to & \text{Assembleur 1} & \to & \text{Machine 1} \\
\text{Programme A} & \to & \text{Assembleur 2} & \to & \text{Machine 2} \\
\text{Programme B} & \to & \text{Assembleur 1} & \to & \text{Machine 1} \\
\text{Programme B} & \to & \text{Assembleur 2} & \to & \text{Machine 2} \\
\text{Programme C} & \to & \text{Assembleur 1} & \to & \text{Machine 1} \\
\ldots & \to & \ldots & \to & \ldots
\end{array}
$$

$$
\left.
\begin{array}{c}
\text{Programme A} \\
\text{Programme B} \\
\text{Programme C} \\
\text{Programme D} \\
\text{Programme E} \\
\ldots
\end{array}
\right\}
$$

Portabilité

Programme A $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
Programme A $\rightarrow$ Assembleur 2 $\rightarrow$ Machine 2
Programme B $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
Programme B $\rightarrow$ Assembleur 2 $\rightarrow$ Machine 2
Programme C $\rightarrow$ Assembleur 1 $\rightarrow$ Machine 1
... $\rightarrow$ ... $\rightarrow$ ...

Programme A
Programme B
Programme C
Programme D
Programme E
...
$\rightarrow$ Langage $\rightarrow$

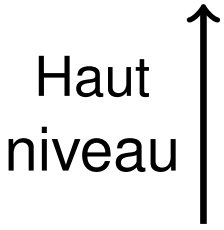
Portabilité

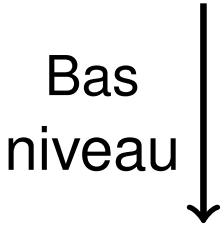
$$
\begin{array}{ccccc}
\text{Programme A} & \rightarrow & \text{Assembleur 1} & \rightarrow & \text{Machine 1} \\
\text{Programme A} & \rightarrow & \text{Assembleur 2} & \rightarrow & \text{Machine 2} \\
\text{Programme B} & \rightarrow & \text{Assembleur 1} & \rightarrow & \text{Machine 1} \\
\text{Programme B} & \rightarrow & \text{Assembleur 2} & \rightarrow & \text{Machine 2} \\
\text{Programme C} & \rightarrow & \text{Assembleur 1} & \rightarrow & \text{Machine 1} \\
\ldots & \rightarrow & \ldots & \rightarrow & \ldots
\end{array}
$$

$$
\left.\begin{array}{l}
\text{Programme A} \\
\text{Programme B} \\
\text{Programme C} \\
\text{Programme D} \\
\text{Programme E} \\
\ldots
\end{array}\right\} \rightarrow \text{Langage} \rightarrow \left\{\begin{array}{l}
\text{Machine 1} \\
\text{Machine 2} \\
\text{Machine 3} \\
\text{Machine 4} \\
\text{Machine 5} \\
\ldots
\end{array}\right.
$$

Langages de bas/haut niveau

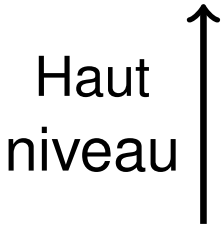
Langages de bas/haut niveau





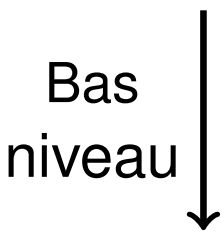
Langage	0100 1110	0101 1001
machine	0100 0101	0100 1000

Langages de bas/haut niveau



Langage assembleur

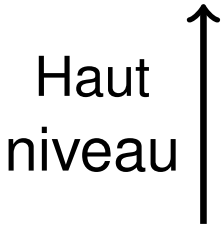
```
mov %eax, %esp
jmp %esp
```



Langage machine

0100 1110 0101 1001
0100 0101 0100 1000

Langages de bas/haut niveau



Langage portable

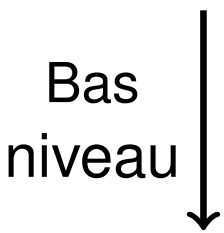
```
int len = 42;
char* str = malloc(len);
```

Langage assembleur

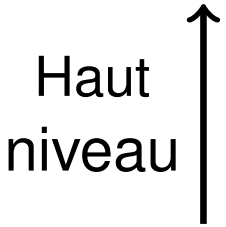
```
mov %eax, %esp
jmp %esp
```

Langage machine

0100 1110 0101 1001
0100 0101 0100 1000



Langages de bas/haut niveau



Langage abstrait

```
(defun fact (n)
  (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

Langage portable

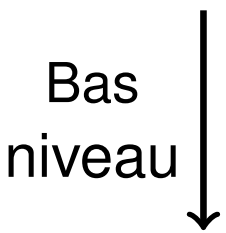
```
int len = 42;
char* str = malloc(len);
```

Langage assembleur

```
mov %eax, %esp
jmp %esp
```

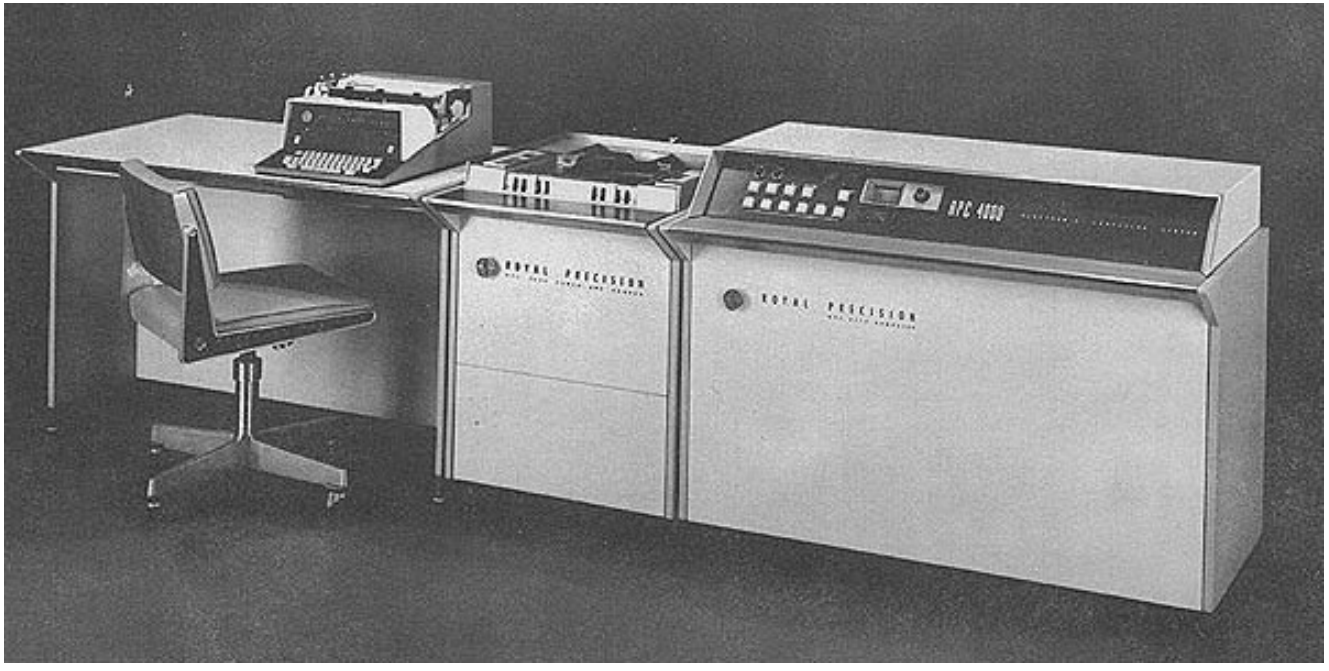
Langage machine

```
0100 1110  0101 1001
0100 0101  0100 1000
```



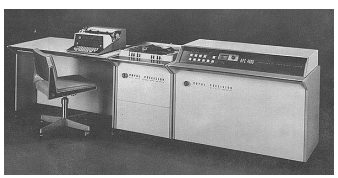
Un assembleur portable

Matériel



Un assembleur portable

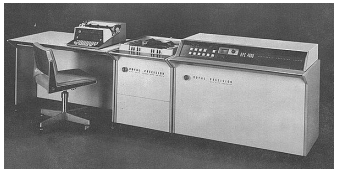
Matériel

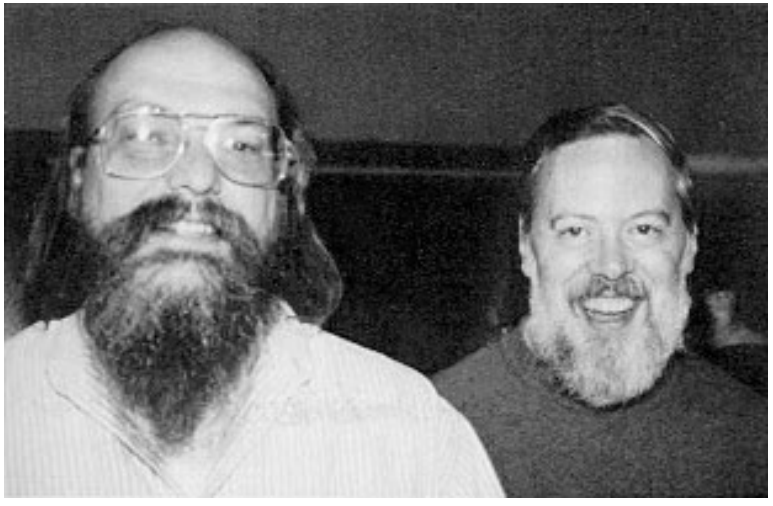


Un assembleur portable

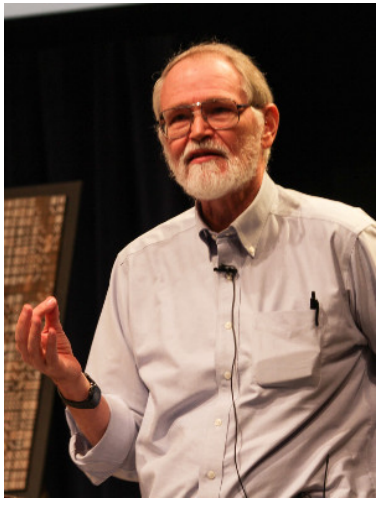
Utilisateur

Matériel





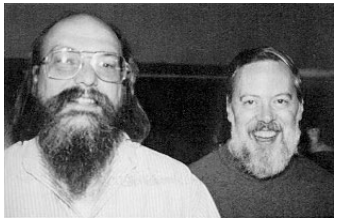
Ken Thompson

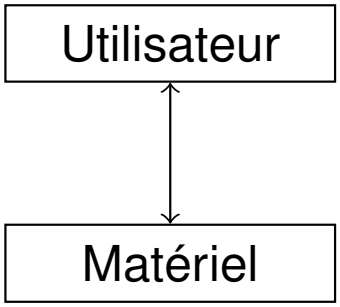


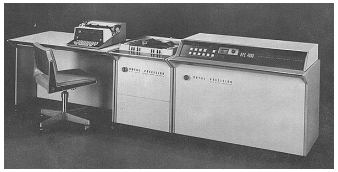
Dennis Ritchie

Brian Kernighan

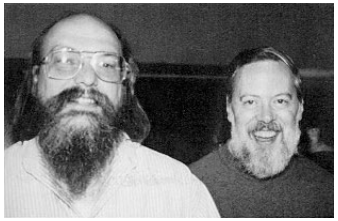
Un assembleur portable

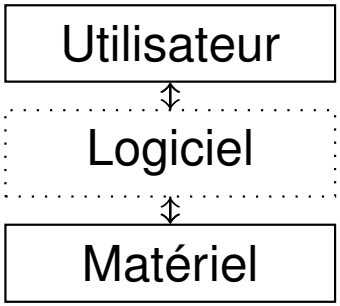


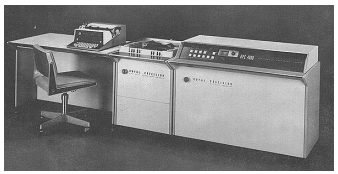




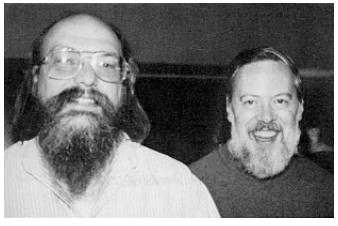
Un assembleur portable







Un assembleur portable



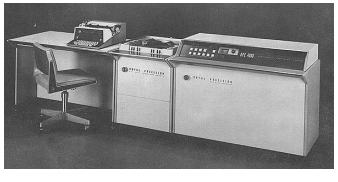
Utilisateur

↕

Logiciel

↕

Matériel



```
/* C command */

main(argc, argv)
char argv[][]; {
        extern callsys, printf, unlink, link, nodup;
        extern getsuf, setsuf, copy;
        extern tsp;
        extern tmp0, tmp1, tmp2, tmp3;
        char tmp0[], tmp1[], tmp2[], tmp3[];
        char glotch[100][], clist[50][], llist[50][], ts[500];
        char tsp[], av[50][], t[];
        auto nc, nl, cflag, i, j, c;

        tmp0 = tmp1 = tmp2 = tmp3 = "//";
        tsp = ts;
        i = nc = nl = cflag = 0;
        while(++i < argc) {
                if(*argv[i] == '-' & argv[i][1]=='c')
                        cflag++;
                else {
                        t = copy(argv[i]);
                        if((c=getsuf(t))=='c') {
                                clist[nc++] = t;
"ncc.c" 31L, 656C written                                1,1           Top
```

Typage

Que signifie `0010 1010` ?

Typage

Que signifie `0010 1010` ?

Ça dépend !

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ? 

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ? 
- Un pixel monochrome ? 

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ? 
- Un pixel monochrome ? 
- ...

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ?
- Un pixel monochrome ?
- ...

- `x = "4"`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ?
- Un pixel monochrome ?
- ...

- `x = "4"`
  `"4"`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ?
- Un pixel monochrome ?
- ...

- `x = "4"`
  `"4"`
- `x - 2 + 4`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ?
- Un pixel monochrome ?
- ...

- `x = "4"`
  `"4"`
- `x - 2 + 4`
  `6`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ? 
- Un pixel monochrome ? 
- ...

- `x = "4"`
  `"4"`
- `x - 2 + 4`
  `6`
- `x + 4 - 2`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ? 
- Un pixel monochrome ? 
- ...

- `x = "4"`
  `"4"`
- `x - 2 + 4`
  `6`
- `x + 4 - 2`
  `42`

Typage

Que signifie `0010 1010` ?

Ça dépend ! Avec 8 bits :

- Un nombre entier ? `42`
- Un caractère ? `:`
- Un nombre à virgule ? $+1.25 * 2^4$
- Une adresse mémoire ? `0x2a`
- Un booléen ? `vrai`
- Un pixel de couleur ?
- Un pixel monochrome ?
- ...

- `x = "4"`
  `"4"`
- `x - 2 + 4`
  `6`
- `x + 4 - 2`
  `42`

```
TypeError: can only
concatenate str
(not "int") to str
```

```
WARNING: Assignment
makes pointer from
integer without a
cast.
```

Ramasse-miettes

Paradigmes

Différentes conceptions de la programmation :

- Impérative

Paradigmes

Différentes conceptions de la programmation :

- Impérative
- Déclarative

Paradigmes

Différentes conceptions de la programmation :

- Impérative
- Déclarative
- Fonctionnelle

Paradigmes

Différentes conceptions de la programmation :

- Impérative
- Déclarative
- Fonctionnelle
- Orientée-objet

Paradigmes

Différentes conceptions de la programmation :

- Impérative
- Déclarative
- Fonctionnelle
- Orientée-objet
- Évènementielle

Paradigmes

Différentes conceptions de la programmation :

- Impérative
- Déclarative
- Fonctionnelle
- Orientée-objet
- Évènementielle
- ...

Langages formels 101

Syntaxe et sémantique

Une grammaire normale