



*Éléments d'informatique – Cours 1.*  
*Introduction*  
*L'instruction de contrôle if*

Pierre Boudes

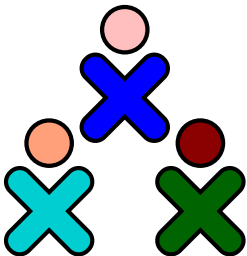
3 septembre 2012



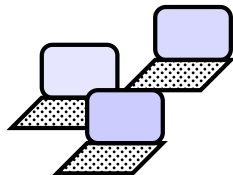


# Langages et programmes

*Humains*



*Ordinateurs*

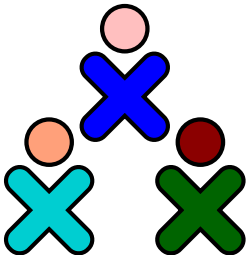




## Langages et programmes

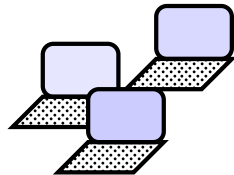
### *Humains*

communiquent entre eux en  
langage naturel



### *Ordinateurs*

obéissent à des instructions  
en langage machine

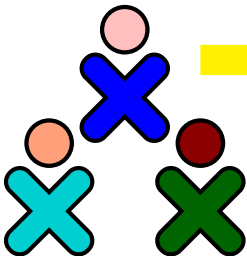




## Langages et programmes

*Humains*

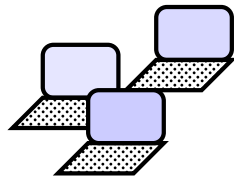
langage naturel



utilisent

*Ordinateurs*

langage machine





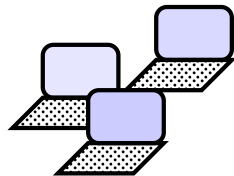
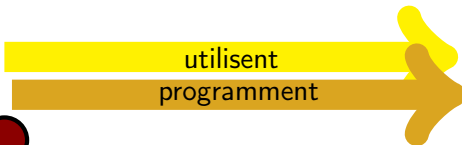
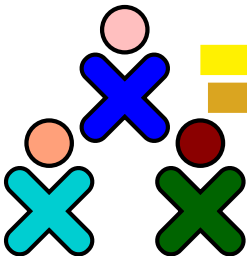
## Langages et programmes

*Humains*

langage naturel

*Ordinateurs*

langage machine





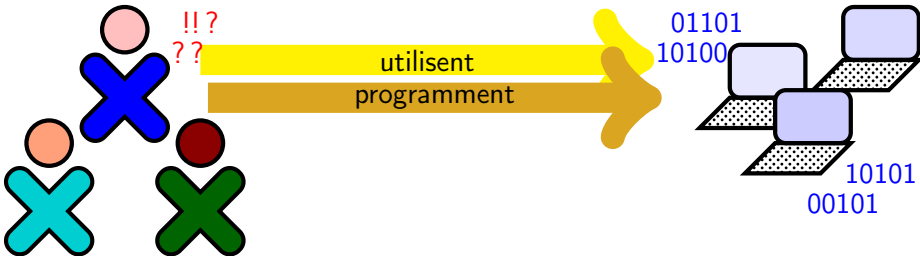
## Langages et programmes

*Humains*

langage naturel

*Ordinateurs*

langage machine

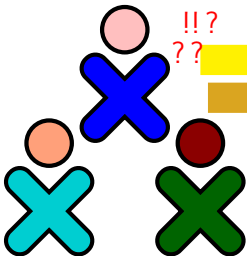




# Langages et programmes

*Humains*

langage naturel



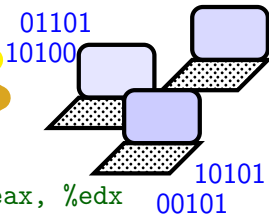
utilisent

programment

*Ordinateurs*

langage machine

`movb 0x65, %eax`



`movl %eax, %edx`



## Langages et programmes

*Humains*

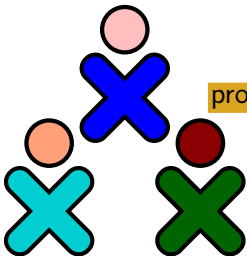
langage naturel

Langage de

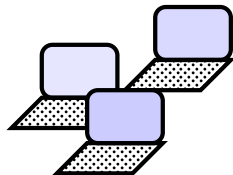
programmation

*Ordinateurs*

langage machine



programmation







## Langages et programmes

*Humains*

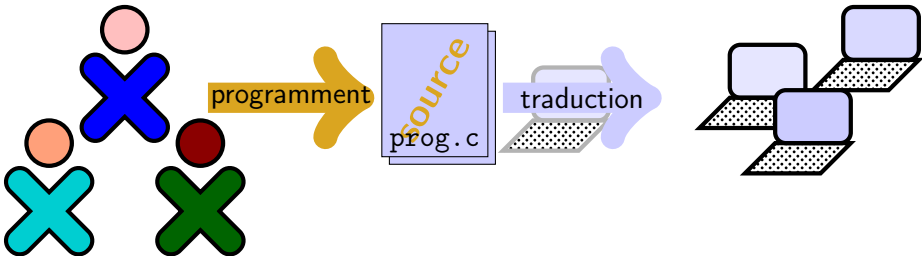
langage naturel

Langage de

programmation

*Ordinateurs*

langage machine





## Langages et programmes

*Humains*

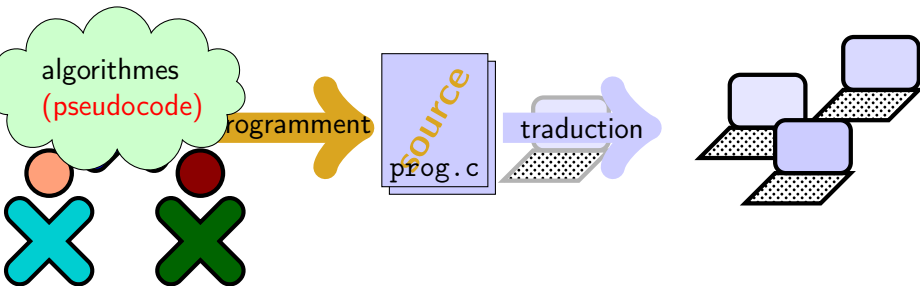
langage naturel

Langage de

programmation

*Ordinateurs*

langage machine





## Langages et programmes

*Humains*

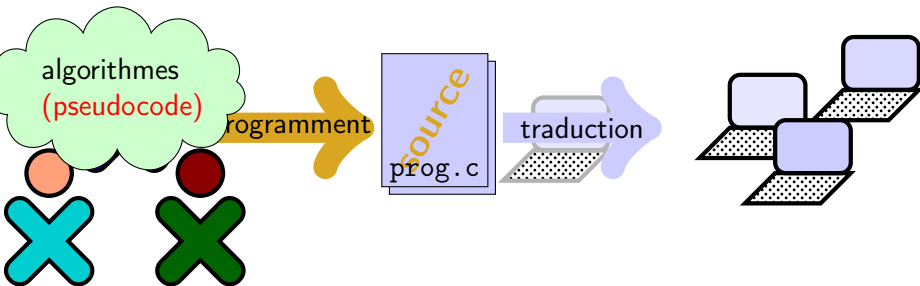
langage naturel

Langage de

programmation

*Ordinateurs*

langage machine



Dans ce cours : surtout du **langage C**, du pseudocode pour mettre au point les algorithmes et un petit peu d'un langage assembleur *jouet* (amil) pour expliquer le langage C.



## *Introduction : langages et programmes*

### *Architecture matérielle et langage machine*

- Architecture de von Neumann
- Représentation des informations
- Exécution des instructions

### *Premiers pas en langage C*

- La programmation structurée
- Variabes impératives et affectation
- L'instruction de contrôle if

### *Démos et fin*

### *Liens utiles*



*« L'informatique n'est pas plus la science des ordinateurs  
que l'astronomie n'est celle des télescopes. »*

*E. W. Dijkstra*



## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN



## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?



## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**





## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de tous nos processeurs



## *Architecture de von Neumann*

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de tous nos processeurs
- De quoi cette machine est-elle faite ?

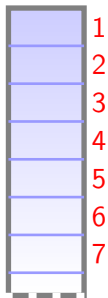


## Architecture de von Neumann ~~✗~~

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de tous nos processeurs
- De quoi cette machine est-elle faite ?

### *Machine de vN*

mémoire



- De mémoire (une suite de cases numérotées)

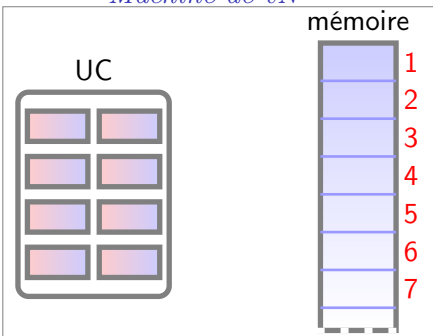


## Architecture de von Neumann ~~✗~~

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de tous nos processeurs
- De quoi cette machine est-elle faite ?

### *Machine de vN*

- De mémoire (une suite de cases numérotées)
- d'une **unité de calcul**, travaillant sur des **registres**

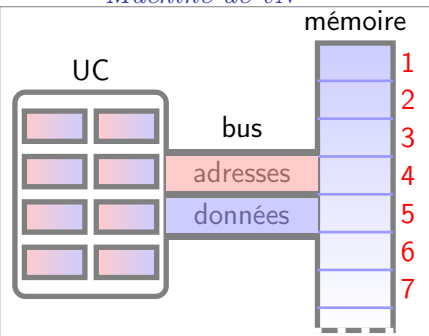




## Architecture de von Neumann ~~✗~~

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de tous nos processeurs
- De quoi cette machine est-elle faite ?

### *Machine de vN*



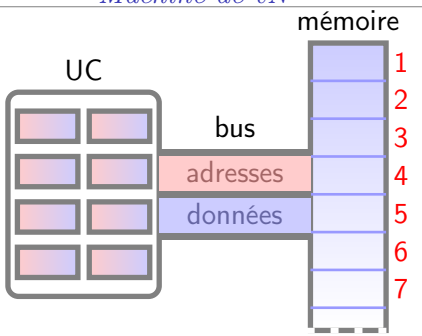
- De mémoire (une suite de cases numérotées)
- d'une **unité de calcul**, travaillant sur des **registres**
- d'un **bus** système (adresses et données) reliant mémoire et UC



## Architecture de von Neumann ~~✗~~

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de tous nos processeurs
- De quoi cette machine est-elle faite ?

### *Machine de vN*



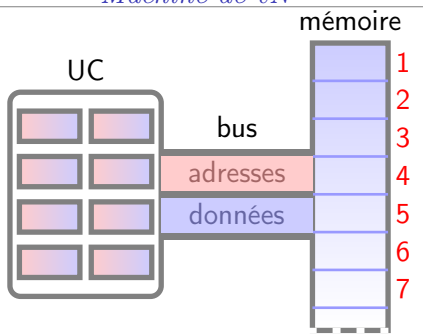
- De mémoire (une suite de cases numérotées)
- d'une **unité de calcul**, travaillant sur des **registres**
- d'un **bus** système (adresses et données) reliant mémoire et UC
- ~~De périphériques (on oublie !)~~



## Architecture de von Neumann ~~✗~~

- John William Mauchly et John Eckert autant (ou plus) que vN
- Qu'est-ce que c'est ?
  - L'idée d'une machine à **programme stocké**
  - Une machine réalisée, l'ancêtre de tous nos processeurs
- De quoi cette machine est-elle faite ?

### *Machine de vN*



- De mémoire (une suite de cases numérotées)
- d'une **unité de calcul**, travaillant sur des **registres**
- d'un **bus** système (adresses et données) reliant mémoire et UC
- ~~De périphériques (on oublie !)~~
- La mémoire contient le programme et les données.



## *Représentation en binaire des informations*

### *Definition (bit)*

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).





## *Représentation en binaire des informations*

### *Definition (bit)*

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est faite de bits.



## Représentation en binaire des informations

### Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est faite de bits.
- Les cases mémoires et les registres contiennent des **mots mémoire** : des suite de  $n$  bits, où  $n$  est fixé une fois pour toute par l'architecture matérielle.



## Représentation en binaire des informations

### Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est faite de bits.
- Les cases mémoires et les registres contiennent des **mots mémoire** : des suite de  $n$  bits, où  $n$  est fixé une fois pour toute par l'architecture matérielle.
- les instructions du langage machine sont écrites en binaire.



## Représentation en binaire des informations

### Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est faite de bits.
- Les cases mémoires et les registres contiennent des **mots mémoire** : des suite de  $n$  bits, où  $n$  est fixé une fois pour toute par l'architecture matérielle.
- les instructions du langage machine sont écrites en binaire.
- le **langage assembleur** est une notation du langage machine plus pratique pour les humains.



## Représentation en binaire des informations

### Definition (bit)

- Le chiffre binaire, ou *bit*, est l'équivalent binaire de nos chiffres décimaux. Il peut valoir soit 0 soit 1. Un bit est une **quantité élémentaire d'information** (oui ou non, ouvert ou fermé, etc.).
- L'information manipulée par un ordinateur est faite de bits.
- Les cases mémoires et les registres contiennent des **mots mémoire** : des suite de  $n$  bits, où  $n$  est fixé une fois pour toute par l'architecture matérielle.
- les instructions du langage machine sont écrites en binaire.
- le **langage assembleur** est une notation du langage machine plus pratique pour les humains.

Vous en verrez plus sur les codages en binaire des données dans un autre cours.



## *Cycle d'exécution*



## *Cycle d'exécution*

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction



## *Cycle d'exécution*

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)





## Cycle d'exécution

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)
- CP est *incrémenté* (c'est à dire que sa valeur augmente de 1)



## Cycle d'exécution

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)
- CP est *incrémenté* (c'est à dire que sa valeur augmente de 1)
- le contenu de RI est décodé afin de déterminer l'opération à exécuter



## Cycle d'exécution

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)
- CP est *incrémenté* (c'est à dire que sa valeur augmente de 1)
- le contenu de RI est décodé afin de déterminer l'opération à exécuter
- l'opération est exécutée (le contenu d'un ou plusieurs registres est modifié, ou bien celui d'une case mémoire)



## Cycle d'exécution

- Le registre **compteur de programme** (CP) contient l'adresse du mot mémoire représentant la prochaine instruction
- le contenu de ce mot est transféré de la mémoire centrale dans le **registre d'instruction** (RI)
- CP est *incrémenté* (c'est à dire que sa valeur augmente de 1)
- le contenu de RI est décodé afin de déterminer l'opération à exécuter
- l'opération est exécutée (le contenu d'un ou plusieurs registres est modifié, ou bien celui d'une case mémoire)
- Fin du cycle d'exécution et démarrage d'un nouveau cycle



## *Instructions*

Une instruction machine type comporte un **code d'opération** et, si nécessaire, une ou deux *opérandes* (ou *arguments* de l'opération).



## *Instructions*

Une instruction machine type comporte un **code d'opération** et, si nécessaire, une ou deux *opérandes* (ou *arguments* de l'opération).

### *Vocabulaire*

Dans l'expression arithmétique usuelle  $3 + 5$ , le signe  $+$  est l'opérateur et les nombres 3 et 5 sont les opérandes.



## *Quelques instructions typiques (Amil) ~~✎~~*



## *Quelques instructions typiques (Amil)*

stop

Arrête l'exécution du programme.

noop

N'effectue aucune opération.





## *Quelques instructions typiques (Amil)*

stop	Arrête l'exécution du programme.
noop	N'effectue aucune opération.
lecture $i$ $r_j$	Charge, dans le registre $j$ , le contenu de la mémoire d'adresse $i$ .
écriture $r_i$ $j$	Écrit le contenu du registre $i$ dans la mémoire d'adresse $j$ .



## *Quelques instructions typiques (Amil) ~~✗~~*

stop	Arrête l'exécution du programme.
noop	N'effectue aucune opération.
lecture i rj	Charge, dans le registre $j$ , le contenu de la mémoire d'adresse $i$ .
écriture ri j	Écrit le contenu du registre $i$ dans la mémoire d'adresse $j$ .
saut i	Met CP à la valeur $i$ .
sautpos ri j	Si la valeur contenue dans le registre $i$ est positive ou nulle, met CP à la valeur $j$ .



## Quelques instructions typiques (Amil) ~~✂~~

stop	Arrête l'exécution du programme.
noop	N'effectue aucune opération.
lecture i rj	Charge, dans le registre $j$ , le contenu de la mémoire d'adresse $i$ .
écriture ri j	Écrit le contenu du registre $i$ dans la mémoire d'adresse $j$ .
saut i	Met CP à la valeur $i$ .
sautpos ri j	Si la valeur contenue dans le registre $i$ est positive ou nulle, met CP à la valeur $j$ .
inverse ri	Inverse le signe du contenu du registre $i$ .
add ri rj	Ajoute la valeur du registre $i$ à celle du registre $j$ .
soustr ri rj	Soustrait la valeur du registre $i$ à celle du registre $j$ .
mult ri rj	Multiplie ...
div ri rj	Divise ...



## Quelques instructions typiques (Amil) ~~✎~~

stop	Arrête l'exécution du programme.
noop	N'effectue aucune opération.
lecture i rj	Charge, dans le registre $j$ , le contenu de la mémoire d'adresse $i$ .
écriture ri j	Écrit le contenu du registre $i$ dans la mémoire d'adresse $j$ .
saut i	Met CP à la valeur $i$ .
sautpos ri j	Si la valeur contenue dans le registre $i$ est positive ou nulle, met CP à la valeur $j$ .
inverse ri	Inverse le signe du contenu du registre $i$ .
add ri rj	Ajoute la valeur du registre $i$ à celle du registre $j$ .
soustr ri rj	Soustrait la valeur du registre $i$ à celle du registre $j$ .
mult ri rj	Multiplie ...
div ri rj	Divise ...
lecture *ri rj	Charge, dans $rj$ , le contenu de la mémoire dont l'adresse est la valeur du registre $i$



## *Trace d'exécution*

On simule pas à pas l'exécution du programme.



## Trace d'exécution

On simule pas à pas l'exécution du programme.

### *Programme*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?



## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12



## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?





## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				



## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			



## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			
soustr r2 r0	3	4	9				



## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			
soustr r2 r0	3	4	9				
sautpos r0 8	4	<b>8</b>					



## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			
soustr r2 r0	3	4	9				
sautpos r0 8	4	<b>8</b>					
ecriture r0 12	5	9					9



## Trace d'exécution

On simule pas à pas l'exécution du programme.

*Programme*

*Trace*

1. lecture 10 r0
2. lecture 11 r2
3. soustr r2 r0
4. sautpos r0 8
5. lecture 10 r2
6. add r2 r0
7. saut 4
8. ecriture r0 12
9. stop
10. 14
11. 5
12. ?

<i>Instructions</i>	Cycles	CP	r0	r2	10	11	12
Initialisation	0	1	?	?	14	5	?
lecture 10 r0	1	2	14				
lecture 11 r2	2	3		5			
soustr r2 r0	3	4	9				
sautpos r0 8	4	<b>8</b>					
ecriture r0 12	5	9					9
stop	6	10					



## *La programmation structurée*

### *Definition (Programmation structurée)*

Programmer par *blocs* d'instructions en combinant ces blocs de trois manières :



## *La programmation structurée*

### *Definition (Programmation structurée)*

Programmer par *blocs* d'instructions en combinant ces blocs de trois manières :

1. exécuter les blocs les uns à la suite des autres (*séquence*)





## *La programmation structurée*

### *Definition (Programmation structurée)*

Programmer par *blocs* d'instructions en combinant ces blocs de trois manières :

1. exécuter les blocs les uns à la suite des autres (*séquence*)
2. si une certaine condition est vraie, exécuter un bloc sinon en exécuter un autre (*sélection*)



## La programmation structurée

### Definition (Programmation structurée)

Programmer par *blocs* d'instructions en combinant ces blocs de trois manières :

1. exécuter les blocs les uns à la suite des autres (*séquence*)
2. si une certaine condition est vraie, exécuter un bloc sinon en exécuter un autre (*sélection*)
3. recommencer l'exécution d'un bloc tant qu'une certaine condition est vraie (*répétition*).



## La programmation structurée

### Definition (Programmation structurée)

Programmer par *blocs* d'instructions en combinant ces blocs de trois manières :

1. exécuter les blocs les uns à la suite des autres (*séquence*)
2. si une certaine condition est vraie, exécuter un bloc sinon en exécuter un autre (*sélection*)
3. recommencer l'exécution d'un bloc tant qu'une certaine condition est vraie (*répétition*).

Un bloc peut lui-même être une combinaison de blocs (ou juste une instruction).



## La programmation structurée

### Definition (Programmation structurée)

Programmer par *blocs* d'instructions en combinant ces blocs de trois manières :

1. exécuter les blocs les uns à la suite des autres (*séquence*)
2. si une certaine condition est vraie, exécuter un bloc sinon en exécuter un autre (*sélection*)
3. recommencer l'exécution d'un bloc tant qu'une certaine condition est vraie (*répétition*).

Un bloc peut lui-même être une combinaison de blocs (ou juste une instruction).

La sélection et la répétition sont assurées par des *instructions de contrôle*.



## La programmation structurée

### Definition (Programmation structurée)

Programmer par *blocs* d'instructions en combinant ces blocs de trois manières :

1. exécuter les blocs les uns à la suite des autres (*séquence*)
2. si une certaine condition est vraie, exécuter un bloc sinon en exécuter un autre (*sélection*)
3. recommencer l'exécution d'un bloc tant qu'une certaine condition est vraie (*répétition*).

Un bloc peut lui-même être une combinaison de blocs (ou juste une instruction).

La sélection et la répétition sont assurées par des *instructions de contrôle*. Tout programme en langage machine peut être transformé en un programme structuré (Böhm-Jacopini 1966).



## *Nécessité d'une traduction*

Les langages structurés, dits de haut niveau, nécessitent une **traduction en langage machine**.



## *Nécessité d'une traduction*

Les langages structurés, dits de haut niveau, nécessitent une **traduction en langage machine**.

Cette traduction est assurée par un programme particulier : un compilateur (traduction une fois pour toute) ou un interprète (traduction à chaque exécution).



## Nécessité d'une traduction

Les langages structurés, dits de haut niveau, nécessitent une **traduction en langage machine**.

Cette traduction est assurée par un programme particulier : un compilateur (traduction une fois pour toute) ou un interprète (traduction à chaque exécution).

La traduction suit des règles précises et systématiques. À chaque instruction du langage correspond un *schéma de traduction*. C'est ce schéma qui donne son sens (son effet) à l'instruction.





## Construction d'un programme C ~~✎~~

### Code source

```
/* Declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */

/* Declaration des constantes et types utilisateurs */

/* Declaration des fonctions utilisateurs */

/* Fonction principale */
int main()
{
    /* Declaration et initialisation des variables */
    ...
    /* valeur fonction */
    return EXIT_SUCCESS;
}

/* Definitions des fonctions utilisateurs */
```

Les commentaires sont ignorés lors de la traduction en langage machine.



## Traduction de l'affectation

### Code source

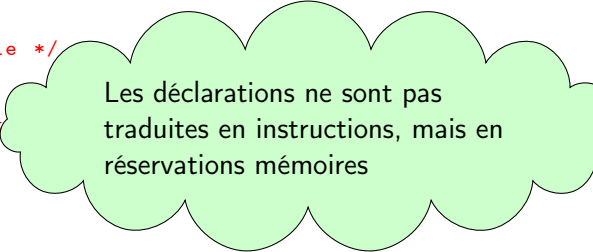
```
...  
/* Fonction principale */  
int main()  
{  
    /* Declaration et initialisation des variables */  
    int x = 5;  
  
    x = 3 * x + 1;  
    ...  
}
```

## Traduction de l'affectation

### Code source

```
...  
/* Fonction principale */  
int main()  
{  
    /* Declaration et  
    int x = 5;  
  
    x = 3 * x + 1;  
    ...
```

### Schéma de traduction



Les déclarations ne sont pas traduites en instructions, mais en réservations mémoires

## Traduction de l'affectation

### Code source

```
...  
/* Fonction principale */  
int main()  
{  
    /* Declaration et initialisation */  
    int x = 5;  
  
    x = 3 * x + 1;  
    ...  
}
```

### Schéma de traduction

*x est une case mémoire de type entier qui contient initialement 5*

## Traduction de l'affectation

### Code source

```
...  
/* Fonction principale */  
int main()  
{  
    /* Declaration et initialisation */  
    int x = 5;  
  
    x = 3 * x + 1;  
    ...  
}
```

### Schéma de traduction

*x est une case mémoire de type entier qui contient initialement 5*

l'affectation est traduite par l'évaluation d'une expression et une écriture en mémoire



## Traduction de l'affectation

### Code source

```
...  
/* Fonction principale */  
int main()  
{  
    /* Declaration et initialisation */  
    int x = 5;  
  
    x = 3 * x + 1;  
    ...  
}
```

### Schéma de traduction

*x est une case mémoire de type entier qui contient initialement 5*

*évaluation de l'expression  $3x + 1$  dans un registre*

## Traduction de l'affectation

### Code source

```
...
/* Fonction principale */
int main()
{
    /* Declaration et initialisation */
    int x = 5;
```

*x est une case mémoire de type entier qui contient initialement 5*

```
x = 3 * x + 1;
...
```

*évaluation de l'expression  $3x + 1$  dans un registre*

*écriture du résultat à l'adresse de x*



## *L'instruction de contrôle if*

*Syntaxe* : `if (condition) { bloc1 } else { bloc2 }.`





## *L'instruction de contrôle if*

*Syntaxe* : `if (condition) { bloc1 } else { bloc2 }.`

*Code source*

```
/* avant */
if (age < 18)
{
    permis = 0;
}
else
{
    permis = 1;
}
/* après */
```



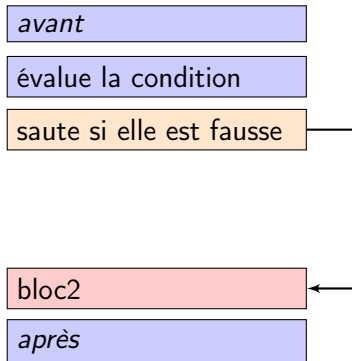
## L'instruction de contrôle if

*Syntaxe* : `if (condition) { bloc1 } else { bloc2 }.`

### Code source

```
/* avant */  
if (age < 18)  
{  
    permis = 0;  
}  
else  
{  
    permis = 1;  
}  
/* après */
```

### Schéma de traduction





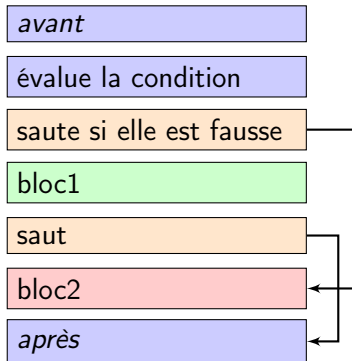
## L'instruction de contrôle if

*Syntaxe* : if (condition) { bloc1 } else { bloc2 }.

### Code source

```
/* avant */  
if (age < 18)  
{  
    permis = 0;  
}  
else  
{  
    permis = 1;  
}  
/* après */
```

### Schéma de traduction



○  
○  
○○○○

○○○  
○  
○

## *Démos et fin*



## *Liens utiles*

- ma page : <http://www-lipn.univ-paris13.fr/~boudes/>
- sites pour apprendre à développer :
  - <http://www.siteduzero.com/> (chercher langage C)
  - <http://www.developpez.com/> (chercher langage C)
- Le nouveau cours de terminal S :  
<https://science-info-lycee.fr>
- Un livre de la BU : *Le livre du C, premier langage (pour les vrais débutants en programmation)*, Claude Delannoy.



## *Liens utiles*

- Polycopiés :
  - le cours I3 de Laure Petrucci :  
http://www-lipn.univ-paris13.fr/~petrucci/polys.html
  - le cours de Anne Canteaut :  
http://www-roc.inria.fr/secret/Anne.Canteaut/COURS\_C/
  - le cours de Bernard Cassagne :  
http://clips.imag.fr/commun/bernard.cassagne/Introduction\_ANSI\_C.html
  - le cours de Henri Garreta :  
http://www.dil.univ-mrs.fr/~garreta/generique/
- Logiciels
  - codeblocks : <http://www.codeblocks.org/>
  - ubuntu : <http://www.ubuntu-fr.org/>
  - virtualbox : <http://www.virtualbox.org/>