Algo/Prog – DUT 1

0. Introduction
1. Appeler un sous-programme
2. Définition de procédures et fonctions
3. Conditionnelle (vu en TP)
4. Instructions complexe (boucles)
5. Variables, types
6. Représentation des données en mémoire
7. Boucles imbriquées et tableaux à doubles entrées
8. Pointeurs, etc
9. `struct`

Organisation du cours

En règle général :

- Découpage : 1H de cours : (Pierre Courtieu)
  2H30 de TP : (Pierre Courtieu + Rachid Rebiha)
- Petit projet en fin de semestre pour ceux qui sont en avance ?

N.B. Naturellement la présence est obligatoire à toutes les séances sauf avis médical.

Bibliographie

Kernighan et Ritchie (1988, 2^{e} édition).
The C Programming Language.
La bible.
fr.wikipedia.org/wiki/The_C_Programming_Language.

Anne Canteaut.
La programmation en C.
Poly en ligne bien fait (derniers chapitres hors sujet pour nous).
www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/.

Interactive C tutorial.
Cours interactif en anglais.
www.learn-c.org/en/.

Des programmes partout

Où ?

- ordinateurs (word, excel, call of duty, SAP)
- serveurs (apache, sendmail, etc)
- smartphone, hifi, photo, cafetière, pacemaker, prothèses
- automobile : jusqu'à 70 calculateurs et 1 million de lignes de code
- avions, trains...
- bientôt serrures, radiateurs, ampoules,...

≪ Le citoyen d'un pays développé utilise quotidiennement et de manière transparente en moyenne 100 processeurs. ≫

Qu'est-ce qu'un programme ?

Wikipedia :

> ≪ *Un programme informatique est un ensemble d'opérations destinées à être exécutées par un ordinateur.* ≫

- Programme source (langage de programmation).
  Compilable (⇒ binaire) (C,fortran) ou interprétable (Java,Python).
- Programme binaire (langage machine).
  exécutable par un microprocesseur.

Le système d'exploitation [1] ≪ démarre ≫ un programme binaire : ordonne au processeur d'exécuter les instructions du programme. Par exemple lorsqu'on double-clique sur une icône, ou bien lorsqu'on tape le nom du ficher exécutable en ligne de commande.

1. (windows, macos, ios, linux, android,...)

Qu'est-ce qu'un programme (séquentiel) ?

Une recette, comme en cuisine. Une étape après l'autre.

1. Pelez et hachez finement 1/4 oignon(s).
2. Dans un bol :
    1. battez 2 oeufs entiers
    2. ajoutez 1 branche de basilic ciselé
    3. ajoutez 1/4 cube de bouillon de volaille.
    4. ajoutez sel et poivre à votre goût
    5. mélangez bien au fouet.
3. Dans une poêle :
    1. faites chauffer l'huile d'olive
    2. faites revenir les oignons hachés pendant 2 min, en remuant
4. Versez dans la poêle les oeufs et le fromage râpé laissez cuire 7 min.

Écrire une recette $\neq$ faire la cuisine

- Une recette écrite par un auteur :
  - est un morceau de texte
- Une exécution par un cuisinier :
  - lit la recette
  - réalise les actions décrites

Écrire une recette $\neq$ faire la cuisine

- Un programme écrit par un programmeur :
  - est un morceau de texte
- Une exécution par un processeur :
  - lit le programme
  - réalise les actions décrites
- 1 programme, ∞ exécutions différentes
  - paramètres différents
  - entrées différentes (clavier, souris, fichier, etc)

1 programme, ∞ exécutions différentes

Ne pas confondre :

- Le programmeur qui programme :
  - prévoit chaque cas possibles à l'avance
  - décrit la (sous-)recette à suivre dans chacun des cas
- Le processeur qui exécute
  - ne fait que suivre la « recette » pas à pas
  - ne fera pas tous les cas lors d'une exécution

Exemple de programme machine (hexadécimal)

```
00000000  fc 31 c0 8e c0 8e d8 8e  d0 bc 00 7c 89 e6 bf 00  |.1.........|....|
00000010  06 b9 00 01 f3 a5 89 fd  b1 08 f3 ab fe 45 f2 e9  |.............E..|
00000020  00 8a f6 46 bb 20 75 08  84 d2 78 07 80 4e bb 40  |...F. u...x..N.@|
00000030  8a 56 ba 88 56 00 e8 fc  00 52 bb c2 07 31 d2 88  |.V..V....R...1..|
00000040  6f fc 0f a3 56 bb 73 19  8a 07 bf 87 07 b1 03 f2  |o...V.s.........|
00000050  ae 74 0e b1 0b f2 ae 83  c7 09 8a 0d 01 cf e8 c5  |.t..............|
00000060  00 42 80 c3 10 73 d8 58  2c 7f 3a 06 75 04 72 05  |.B...s.X,.:.u.r.|
00000070  48 74 0d 30 c0 04 b0 88  46 b8 bf b2 07 e8 a6 00  |Ht.0....F.......|
00000080  be 7b 07 e8 b2 00 8a 56  b9 4e e8 8e 00 eb 05 b0  |.{.....V.N......|
00000090  07 e8 b0 00 30 e4 cd 1a  89 d7 03 7e bc b4 01 cd  |....0......~....|
000000a0  16 75 0d 30 e4 cd 1a 39  fa 72 f2 8a 46 b9 eb 16  |.u.0...9.r..F...|
000000b0  30 e4 cd 16 88 e0 3c 1c  74 f1 2c 3b 3c 04 76 06  |0.....<.t.,;<.v.|
000000c0  2c c7 3c 04 77 c9 98 0f  a3 46 0c 73 c2 88 46 b9  |,.<.w....F.s..F.|
000000d0  be 00 08 8a 14 89 f3 3c  04 9c 74 0a c0 e0 04 05  |.......<..t.....|
000000e0  be 07 93 c6 07 80 53 f6  46 bb 40 75 08 bb 00 06  |......S.F.@u....|
000000f0  b4 03 e8 59 00 5e 9d 75  06 8a 56 b8 80 ea 30 bb  |...Y.^.u..V...0.|
00000100  00 7c b4 02 e8 47 00 72  86 81 bf fe 01 55 aa 0f  |.|...G.r.....U..|
00000110  85 7c ff be 85 07 e8 19  00 ff e3 b0 46 e8 24 00  |.|..........F.$.|
00000120  b0 31 00 d0 eb 17 0f ab  56 0c be 78 07 e8 eb ff  |.1......V..x....|
00000130  89 fe e8 03 00 be 85 07  ac a8 80 75 05 e8 04 00  |...........u....|
00000140  eb f6 24 7f 53 bb 07 00  b4 0e cd 10 5b c3 8a 74  |..$.S.......[..t|
00000150  01 8b 4c 02 b0 01 56 89  e7 f6 46 bb 80 74 13 66  |..L...V...F..t.f|
00000160  6a 00 66 ff 74 08 06 53  6a 01 6a 10 89 e6 48 80  |j.f.t..Sj.j...H.|
00000170  cc 40 cd 13 89 fc 5e c3  20 20 a0 0a 44 65 66 61  |.@....^.  ..Defa|
00000180  75 6c 74 3a a0 0d 8a 00  05 0f 01 06 07 0b 0c 0e  |ult:............|
00000190  83 a5 a6 a9 0d 0c 0b 0a  09 08 0a 0e 11 10 01 3f  |...............?|
000001a0  bf 44 4f d3 4c 69 6e 75  f8 46 72 65 65 42 53 c4  |.DO.Linu.FreeBS.|
000001b0  66 bb 44 72 69 76 65 20  00 00 80 8f b6 00 00 00  |f.Drive ........|
000001c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
```

Exemple de programme machine (assembleur)

```
   .data
UnNom :
   .long 43,54,32,76 /* 4 entiers. */
   .globl _start
_start:
   movl $5, %eax     /* EAX nombre d'entiers restant à additionner */
   movl $0, %ebx     /* EBX va contenir la somme de ces entiers */
   movl $UnNom, %ecx /* ECX << pointe >> sur l'élément à additionner */
top:
   addl (%ecx), %ebx  /* Additionne EBX ECX, résultat dans EBX */
   addl $4, %ecx     /* Déplace le <<pointeur>> sur le suivant */
   decl %eax         /* Décrémente le compteur EAX */
   jnz top           /* Si EAX non nul <<sauter>> à top: */
done:
   movl %ebx, UnAutre /* sinon, le resultat est stocké */
   movl    $0,%ebx     /* Ces instructions permettent d'invoquer de */
   movl    $1,%eax     /* terminer l'exécution d'un programme */
   int     $0x80       /* assembleur et sont indispensable */
```

Exemple de programme machine (assembleur)

```
   .data
UnNom :
   .long 43,54,32,76 /* 4 entiers. */
   .globl _start
_start:
   movl $5, %eax      /* EAX nombre d'entiers restant à additionner */
   movl $0, %ebx      /* EBX va contenir la somme de ces entiers */
   movl $UnNom, %ecx /* ECX << pointe >> sur l'élément à additionner */
top:
   addl (%ecx), %ebx  /* Additionne EBX ECX, résultat dans EBX */
   addl $4, %ecx      /* Déplace le <<pointeur>> sur le suivant */
   decl %eax          /* Décrémente le compteur EAX */
   jnz top            /* Si EAX non nul <<sauter>> à top: */
done:
   movl %ebx, UnAutre /* sinon, le resultat est stocké */
   movl    $0,%ebx     /* Ces instructions permettent d'invoquer de */
   movl    $1,%eax     /* terminer l'exécution d'un programme */
   int     $0x80       /* assembleur et sont indispensable */
```

Commentaire (sans effet)

Exemple de programme machine (assembleur)

```
   .data
UnNom :
   .long 43,54,32,76 /* 4 entiers. */
   .globl _start
_start:   Début des instructions
   movl $5, %eax      /* EAX nombre d'entiers restant à additionner */
   movl $0, %ebx      /* EBX va contenir la somme de ces entiers */
   movl $UnNom, %ecx /* ECX << pointe >> sur l'élément à additionner */
top:
   addl (%ecx), %ebx  /* Additionne EBX ECX, résultat dans EBX */
   addl $4, %ecx      /* Déplace le <<pointeur>> sur le suivant */
   decl %eax          /* Décrémente le compteur EAX */
   jnz top            /* Si EAX non nul <<sauter>> à top: */
done:
   movl %ebx, UnAutre /* sinon, le resultat est stocké */
   movl    $0,%ebx     /* Ces instructions permettent d'invoquer de */
   movl    $1,%eax     /* terminer l'exécution d'un programme */
   int     $0x80       /* assembleur et sont indispensable */
```

exemple de programme C

```c
#include <stdio.h>
void main(void)
{
  int i;
  char prenom[30];
  FILE *FICHIER1;
  FICHIER1=fopen("/home/chr/premierfichier.txt","w");
  for(i=1;i<=10;i++)
    {
      printf("Rentrez un prénom :\n");scanf("%s",prenom);
      fprintf(FICHIER1,"%s\n",prenom);
    }
  fclose(FICHIER1);
}
```

Premier programme

```
#include"inout.h"

int main(int nargs, char **args)
{
  ecrireString("Hello world!\n");
  ecrireString("Bye world!\n");
}
```

Premier programme

```c
#include"inout.h"

int main(int nargs, char **args)
{
  ecrireString("Hello world!\n");
  ecrireString("Bye world!\n");
}
```

Permet `ecrireString`, etc

Premier programme

```
#include"inout.h"

int main(int nargs, char **args)
{
  ecrireString("Hello world!\n");
  ecrireString("Bye world!\n");
}
```

Fonction principale

Premier programme

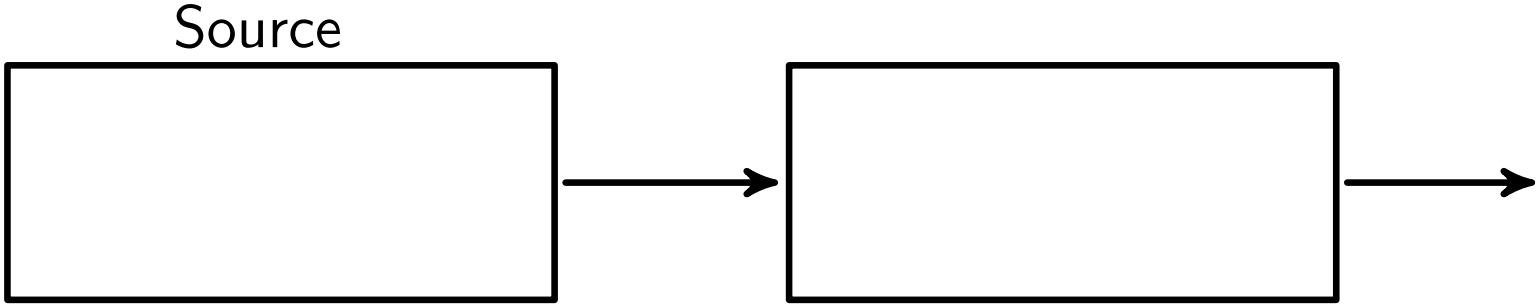
```
#include"inout.h"

int main(int nargs, char **args)
{
  ecrireString("Hello world!\n");
  ecrireString("Bye world!\n");
}
```

Début du programme

Premier programme

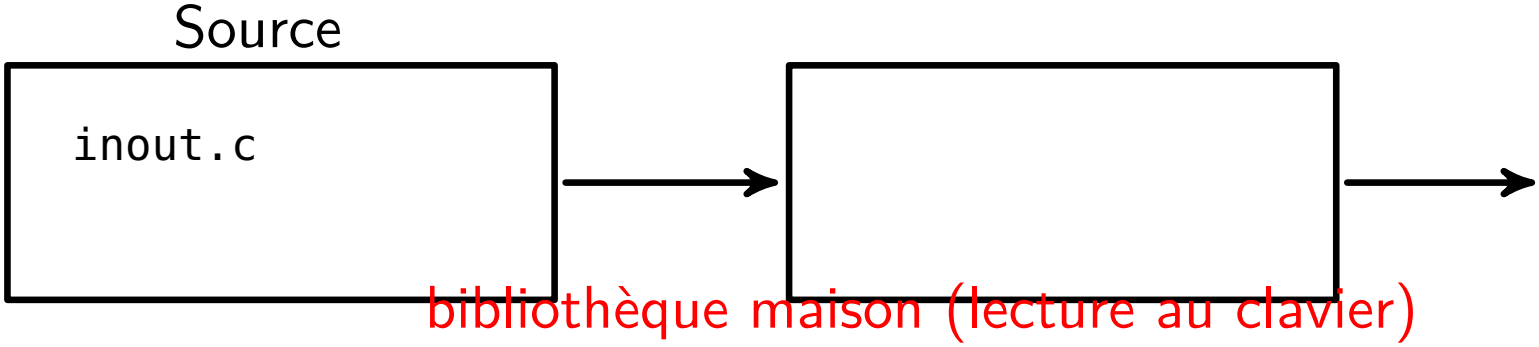
```
#include"inout.h"

int main(int nargs, char **args)
{
  ecrireString("Hello world!\n");
  ecrireString("Bye world!\n");
}
```

En ligne de commande (unix,gcc)



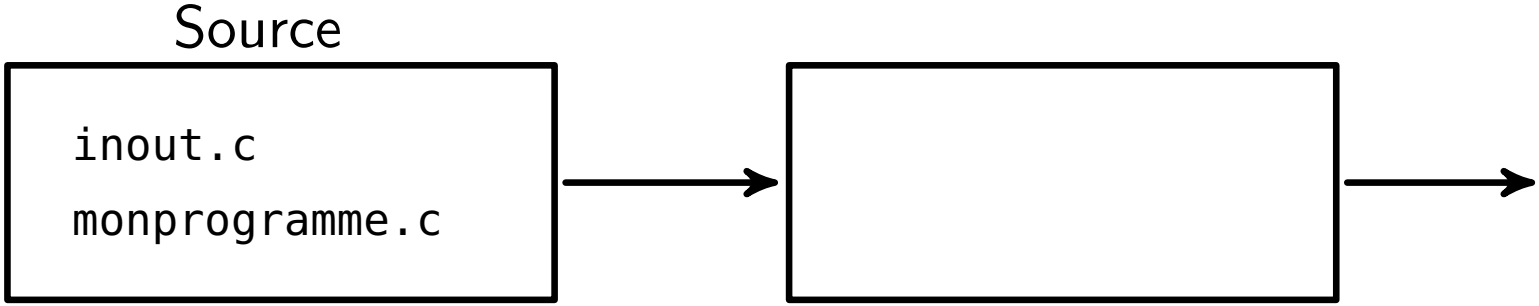

```
gcc -c inout.c
gcc -c premprogramme.c
gcc inout.o premprogramme.o -o monexe
```

En ligne de commande (unix,gcc)




```
gcc -c inout.c
gcc -c premprogramme.c
gcc inout.o premprogramme.o -o monexe
```

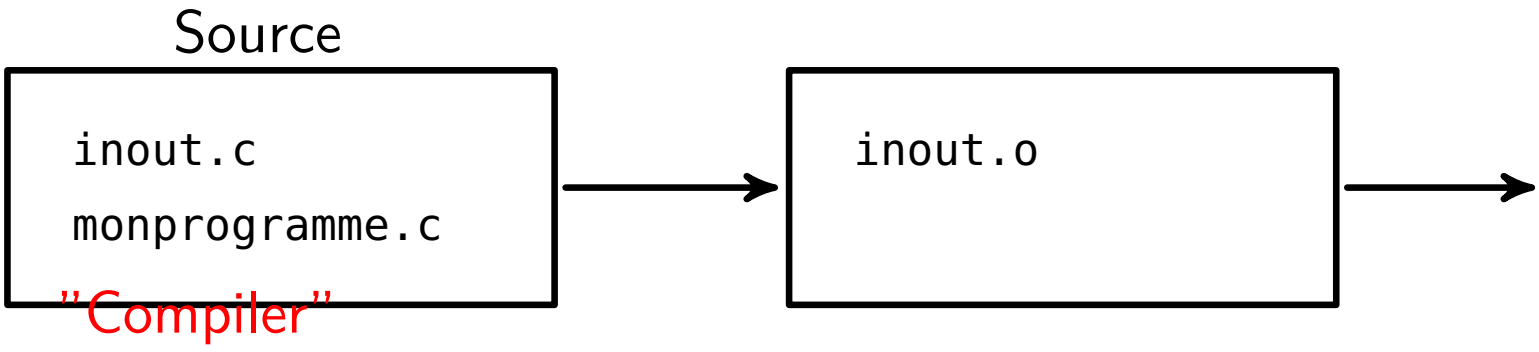
En ligne de commande (unix,gcc)



Écrit par vous

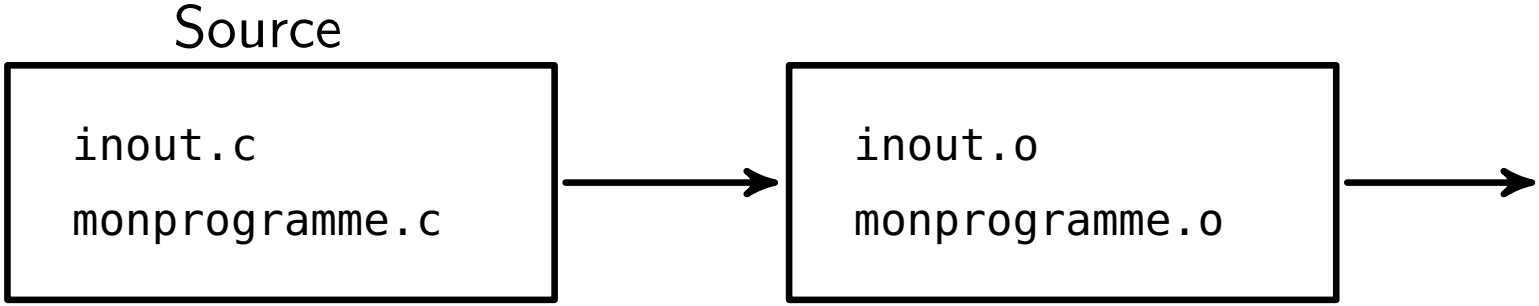
```
gcc -c inout.c
gcc -c premprogramme.c
gcc inout.o premprogramme.o -o monexe
```

En ligne de commande (unix,gcc)



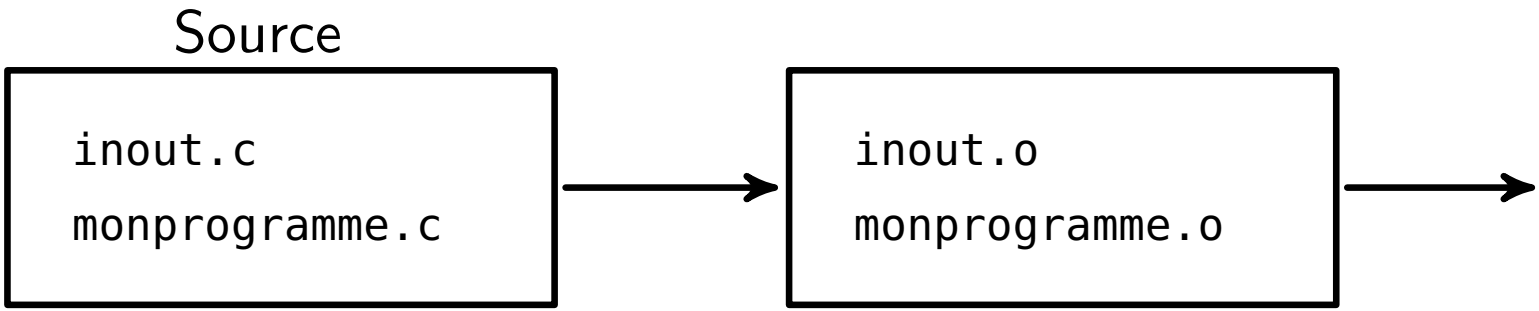

```
gcc -c inout.c ----------------------> inout.o
gcc -c premprogramme.c
gcc inout.o premprogramme.o -o monexe
```

En ligne de commande (unix,gcc)




```
gcc -c inout.c ------------------------> inout.o
gcc -c premprogramme.c----------------> premprogamme.o
gcc inout.o premprogramme.o -o monexe
```

En ligne de commande (unix,gcc)

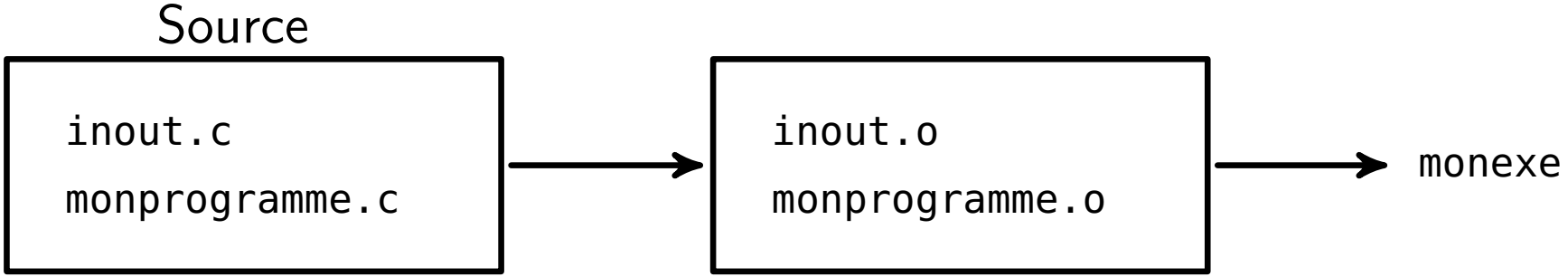



```
gcc -c inout.c ----------------------> inout.o
gcc -c premprogramme.c---------------> premprogamme.o
gcc inout.o premprogramme.o -o monexe
```

"construire un exécutable" (link)

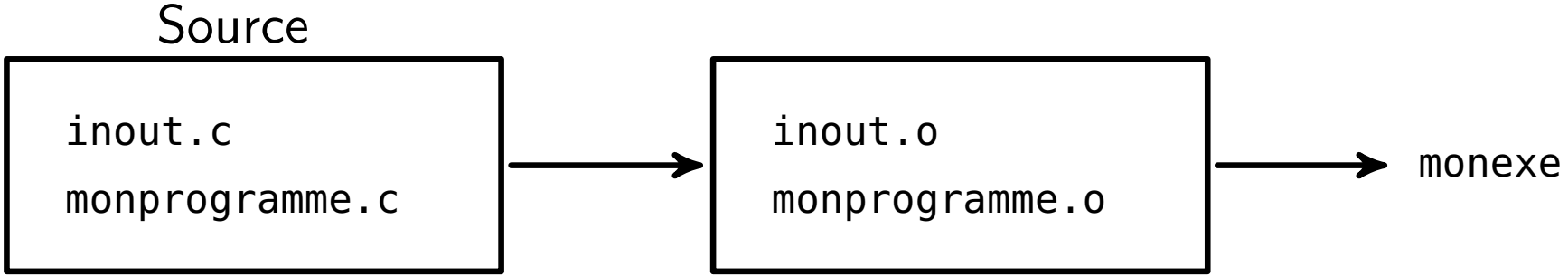
"nom de l'exécutable"

En ligne de commande (unix,gcc)




```
gcc -c inout.c ------------------------> inout.o
gcc -c premprogramme.c----------------> premprogamme.o
gcc inout.o premprogramme.o -o monexe ----> monexe
```

En ligne de commande (unix,gcc)




```
gcc -c inout.c ------------------------> inout.o
gcc -c premprogramme.c----------------> premprogamme.o
gcc inout.o premprogramme.o -o monexe ----> monexe
```

- répertoire courant : `inout.c`, `inout.h` et `premprogamme.c`
- link : tous les fichiers .o d'un seul coup, ordre important
- raccourci quand un seul fichier : `gcc monpogramme.c -o monexe`
- Exécution : `./monexe`

DÉMO

Instructions

```
ecrireString("Hello world\n");
```

Une instruction

- morceau de programme/recette = morceau de texte,
- se termine (en C, C++, Java) par un point-virgule : `;`
- opération réalisée à l'exécution,

Exemples :

- « battez 2 oeufs entiers »
- `ecrireString("Toto\n");`
- `x = y + 3;`
- `return (3);`

Contre-exemples :

- « 2 oeufs »
- `"Toto\n"`
- `y+3`
- `(3)`

Expressions

`"Hello world\n"` `45` `2 + 7 * 0xFF` `x` `2 + x * 0xFF`

Une expression

- morceau de programme/recette = morceau de texte,
- une partie d'une instruction
- qui correspondra à une valeur à l'exécution,

Contre-exemples :

- « 2 oeufs »
- `"Toto\n"`
- `y+3`
- `(3)`

Exemples :

- « battez 2 oeufs entiers »
- `ecrireString("Toto\n");`
- `x = y + 3;`
- `return (3);`

Instruction VS expression

- En principe :

Instruction	Expression
se termine par `;`	fait partie d'une instruction
aura un effet à l'exécution	aura une valeur à l'exécution
n'a pas de type (`void`)	a un type (`int`,`double`,etc)

- En C :
  - les expressions sont parfois aussi des instructions.
  - retournent une valeurs ET ont un effet.
  - ex : `x = 3`
    - ★ effet : met la valeur 3 dans la variable `x`
    - ★ valeur : 3.
    - ★ code bizarre : x = (y = f(x,y)) à éviter

Instruction VS expression

- En principe :

Instruction	Expression
se termine par `;`	fait partie d'une instruction
aura un effet à l'exécution	aura une valeur à l'exécution
n'a pas de type (`void`)	a un type (`int`,`double`,etc)

- En C :
  - les expressions sont parfois aussi des instructions.
  - retournent une valeurs ET ont un effet.
  - ex : `x = 3`
    - ★ effet : met la valeur 3 dans la variable `x`
    - ★ valeur : 3.
    - ★ code bizarre : x = (y = f(x,y)) à éviter