De l’algorithmique au C

Nicolas Delestre
Nicolas.Delestre@insa-rouen.fr
Michel Mainguenaud
Michel.Mainguenaud@insa-rouen.fr

Plan...

- Notes
- Un langage compilé
- Un langage basé sur des modules
- Traduire les types de bases
- Traduire les constantes
- Traduire les opérateurs usuels
- Traduire les instructions simples et composées
- Traduire les conditionnelles
- Traduire les itératifs
- Les pointeurs
- Traduire les fonctions et les procédures
- Traduire `lire` ou `écrire`
- Traduire un programme
- Un exemple

Notes...

- Ce cours n’est pas un cours présentant tous les concepts du C
- Il présente seulement la manière de traduire proprement un programme algorithmique en programme C
- Vous ne trouverez donc pas dans ce cours toutes les subtilités du C, mais vous pouvez vous référer à des cours sur le langage C aux adresses suivantes :
  - `http://www.enstimac.fr/~gaborit/lang/CoursDeC/`
  - `http://www-rocq.inria.fr/codes/Anne.Canteaut/COURS_C/`
  - `http://www-ipst.u-strasbg.fr/pat/program/tpc.htm`
  - `http://www710.univ-lyon1.fr/~bonnev/CoursC/CoursC.html`
  - …

Un langage compilé...

- Le C est un langage compilé
- Les compilateurs C transforment un programme C (fichier suffixé par .c) en programme objet (fichier suffixé par .o) en deux phases :
  1. Le préprocesseur agit sur les macros (commandes précédées d'un #) et transforme le code source C en un autre code source C (ne contenant plus aucune macro) en remplaçant le code macro par son évaluation
     - Par exemple si le code C contient l'instruction `#define PI 3.14159`, le préprocesseur remplacera dans le code source la chaîne de caractères `PI` par la chaîne de caractères `3.14159` à partir de la position du `#define`
  2. Le compilateur transforme ce deuxième code source C en programme machine (nommé code objet)
- Sous Linux (et généralement sous unix), on utilise le compilateur gcc (GNU C Compiler) avec l'option -c pour compiler

Un langage compilé...

- Par exemple pour compiler le programme *helloworld.c* suivant (dont on expliquera le contenu plus tard), il faut taper *gcc -c helloworld.c*, on obtient alors le fichier *helloworld.o*

```
#include <stdio.h>

int main(){
   printf("Hello world");
}
```

Un langage basé sur des modules...

- Le langage C utilise le concept de module (ou de librairie) qui permet de créer et d'utiliser des bibliothèques de fonctions qui peuvent être utilisées dans plusieurs programmes
- De ce fait, le code objet produit par le compilateur n'est pas un programme exécutable car il utilise certainement des fonctions définies dans des modules
- Pour le rendre exécutable il faut le lier aux modules adéquates : c'est ce que l'on nomme l'édition des liens (ou *linkage*)
- Cette édition des liens s'effectue toujours avec gcc (sans option) suivi des codes objets du programme et des modules (non standards) utilisés

Un langage basé sur des modules...

- Par exemple pour linker *helloworld.o* on tape *gcc helloworld.o*
  - Il n'y a que *helloworld.o* car le seul module utilisé est le module *stdio.h* qui est un module standard
- Le programme exécutable *a.out* est alors créé
- Si à la place de l'exécutable *a.out*, on veut obtenir l'excutable *helloworld* il faut utiliser l'option *-o* suivi du nom du fichier exécutable désiré (ici *-o helloworld*)
- Remarque :
  - Si le programme n'utilise aucune librairie non standard, on peut compiler et linker directement en utilisant gcc suivi du fichier C (avec optionnellement l'option -o), par exemple :
    - *gcc -o helloworld helloworld.c*

Traduire les types simples...

- Le C propose les types simples suivants :
  - int, long, short, float, double, char
- On peut donc suivre les règles de traduction suivantes :

Algorithmique	C
Entier,Naturel	int, long, short
Réel	float, double
Caractère	char
Booléen	int (1=Vrai, 0=Faux)
Chaîne de caractères	*Voir cours sur les tableaux*

Traduire les constantes...

- Pour traduire les constantes entières, le C propose trois notations :
  1. La notation décimale, en base dix, par exemple 379
  2. La notation octale, en base huit, qui doit commencer par un zéro par exemple 0573
  3. Le notation hexadécimale, en base seize, qui doit commencer par un zéro suivi d'un x (ou X), par exemple 0X17B (ou 0x17B, 0X17b, 0x17b)
- Les constantes caractères sont entourées de quote, par exemple 'a'
  - Certains caractères (les caractères non imprimables, avec un code ASCII inférieur à 32, le \, le ') ne sont utilisables qu'en préfixant par un \:
    - le code ASCII du caractère (exprimé en octal), par exemple '\001'
    - un caractère tel que '\n', '\t', '\\', '\"

Traduire les constantes...

- Pour les constantes réelles, on utilise le “.” pour marquer la virgule et le caractère “e” suivi d’un nombre entier, ici a, pour représenter 10^a, par exemple :
  - 2. , .3, 2e4, 2.3e4
- les constantes chaîne de caractères doivent être entourées de guillemet, par exemple "une chaine"
  - Par défaut le compilateur ajoute à la fin de la chaîne de caractères ’\0’, qui est le marqueur de fin de chaîne
- Il n’y a pas de constante nommée en C, il faut utiliser la commande `#define` du préprocesseur

Traduire les opérateurs...

- On traduit les opérateurs en respectant les règles suivantes :

Algorithmique	C
$\leftarrow$	=
$=, \neq$	==, !=
$<, \leq, >, \geq$	<, <=, >, >=
et, ou, non	&&, ‖, !
+, -, *, /	+, -, *, /
div, mod	/, %

Traduire les opérateurs...

- Les opérateurs unaires `++` et `--` sont des opérateurs particuliers qui peuvent avoir jusqu'à deux effets de bord:
  - En dehors de toute affectation, elle incrémente l'opérande associée, par exemple
    - `i++` et `++i` sont équivalents à `i=i+1`
  - Lorsqu'ils sont utilisés dans une affectation, tout dépend de la position de l'opérateur par rapport à l'opérande, par exemple :
    - `j=i++` est équivalent à `j=i;i=i+1;`
    - `j=++i` est équivalent à `i=i+1;j=i;`

Les instructions...

- Il existe deux types d'instructions en C :
  - Les instructions simples (expression, appel de fonction, etc.)
    - Elles finissent toujours par un ';'
    - Par exemple :

```
a = a + 1;
```

  - Les instructions composées qui permettent de considérer une succession d'instructions comme étant une seule instruction
    - Elles commencent par "{" et finissent par "}"
    - Par exemple :

```
{ a = a + 1; b = b + 2; }
```

Traduire les variables...

- On déclare en C une variable en la précédant de son type, par exemple :

```
int a;
float x;
char c;
```

- Une variable globale est défi nie en dehors de toute fonction
  - Sa portée est le fi chier où elle est défi nie à partir de sa défi nition (pas au dessus)
- Une variable locale est défi nie à l'intérieur d'une fonction
  - Sa portée est uniquement la fonction ou elle a été défi nie

Les conditionnelles...

- L'instruction `si...alors...sinon` est traduit par l'instruction `if`, qui a la syntaxe suivante :

```
if (condition)
   // instruction(s) du if
[else
   // instruction(s) du else
]
```

  - Par exemple :

```
if (a<10)
   a=a+1;
else
   a=a+2;
```

Les conditionnelles...

- Remarque :
  - Lorsqu'il y ambiguité sur la porté du `else` d'une instruction `if`, le `else` dépend toujours du `if` le plus proche
  - Par exemple :

```
if (a>b) if (c<d) u=v; else i=j;
```

  - Le mieux étant toutefois de clarifi er cette ambiguité :

```
if (a>b)
   if (c<d)
      u=v;
   else
      i=j;
```

```
if (a>b) {
   if (c<d)
      u=v;
} else {
   i=j;
}
```

Les conditionnelles...

- L'instruction `cas..où` est traduite par l'instruction `switch`, qui a la syntaxe suivante :

```
switch (leSelecteur) {
   case cas1 :
      //instruction(s) du cas 1
      break;
   case cas2 :
      //instruction(s) du cas 2
      break;
   ...
   default :
      //instruction(s) du default
}
```

Les conditionnelles...

- Par exemple :

```
switch(choix) {
    case 't' : printf("vous voulez un triangle"); break;
    case 'c' : printf("vous voulez un carre"); break;
    case 'r' : printf("vous voulez un rectangle"); break;
    default : printf("erreur. recommencez !");
}
```

Traduire les itérations...

- L'instruction `Pour` est traduite par l'instruction `for`, qui a la syntaxe suivante :

```
for(initialisation ;
    condition_d_arret ;
    operation_effectuée_à_chaque_itération)
  instruction;
```

  - Par exemple :

```
for(i=0;i<10;i++)
   printf("%d\n", i);
```

  - Remarque :
    - Contrairement à l'algortihmique le `for` est une itération indéterministe

Traduire les itérations...

- L'instruction `Tant...que` est traduite par l'instruction `while`, qui a la syntaxe suivante :

```
while(condition)
    instruction;
```

  - Par exemple :

```
i=0;
while(i<10){
    printf("%d\n", i);
    i++;
}
```

Traduire les itérations...

- L'instruction `répéter` est traduite par l'instruction `do..while`, qui a la syntaxe suivante :

```
do
    instruction;
while(condition);
```

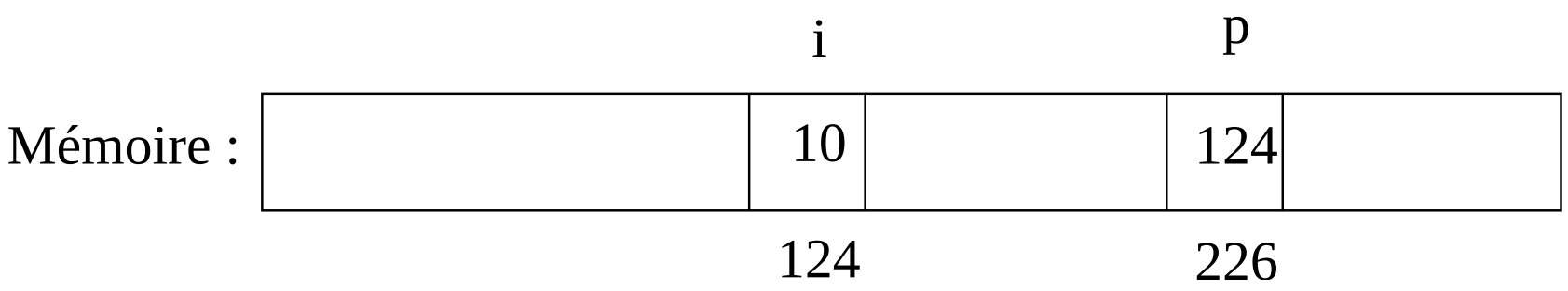
  - Par exemple :

```
i=0;
do {
   printf("%d\n", i);
   i++;
} while (i<=10);
```

  - Attention à la condition qui est la négagtion de la condition d'arrêt de l'instruction algorithmique `répéter...jusqu'à ce que`

Les pointeurs...

- Lorsque l'on déclare une variable, par exemple un entier `i`, l'ordinateur réserve un espace mémoire pour y stocker les valeurs de `i`
- L'emplacement de cet espace dans la mémoire est nommé adresse
  - Dans l'analogie de l'armoire que nous avons vu dans le deuxième cours, l'adresse correspond au numéro du tiroir
- Un pointeur est une variable qui permet de stocker une adresse
- Par exemple si on declare une variable entière `i` (initialisée à 10) et que l'on déclare un pointeur `p` dans lequel on range l'adresse de i (on dit que `p` pointe sur `i`), on a par exemple le schéma suivant :



Les pointeurs...

- On déclare une variable de type pointeur en suffi xant le type de la variable pointée par le caractère *, par exemple

```
int i, j;
int* p;
```

- Il existe deux opérateurs permettant d'utiliser les pointeurs :
  - & : permet d'obtenir l'adresse d'une variable, permettant donc à un pointeur de pointer sur une variable, par exemple

```
p=&i;  // p pointe sur i
```

  - * : permet de déréférencer un pointeur, c'est-à-dire d'accéder à la valeur de la variable pointée, par exemple

```
*p=*p+2; // ajoute 2 a i
j=*p; // met la valeur de i dans j (donc 12)
```

Les pointeurs...

- Par défaut lorsque l'on déclare un pointeur, on ne sait pas sur quoi il pointe
- Comme toute variable, il faut l'initialiser
  - On peut dire qu'un pointeur ne pointe sur rien en lui affectant la valeur NULL
  - Par exemple :

```
int i;
int* p1,p2;
p1=&i;
p2=NULL;
```

Traduire les fonctions...

- On déclare une fonction en respectant la syntaxe suivante :

```
typeRetour nomFonction(type1 param1, type2 param2, ...) {
  // variables locales

  // instructions avec au moins
  // une fois l'instruction return
}
```

- Par exemple :

```
int plusUn(int a){
  return a+1;
}
```

Traduire les procédures...

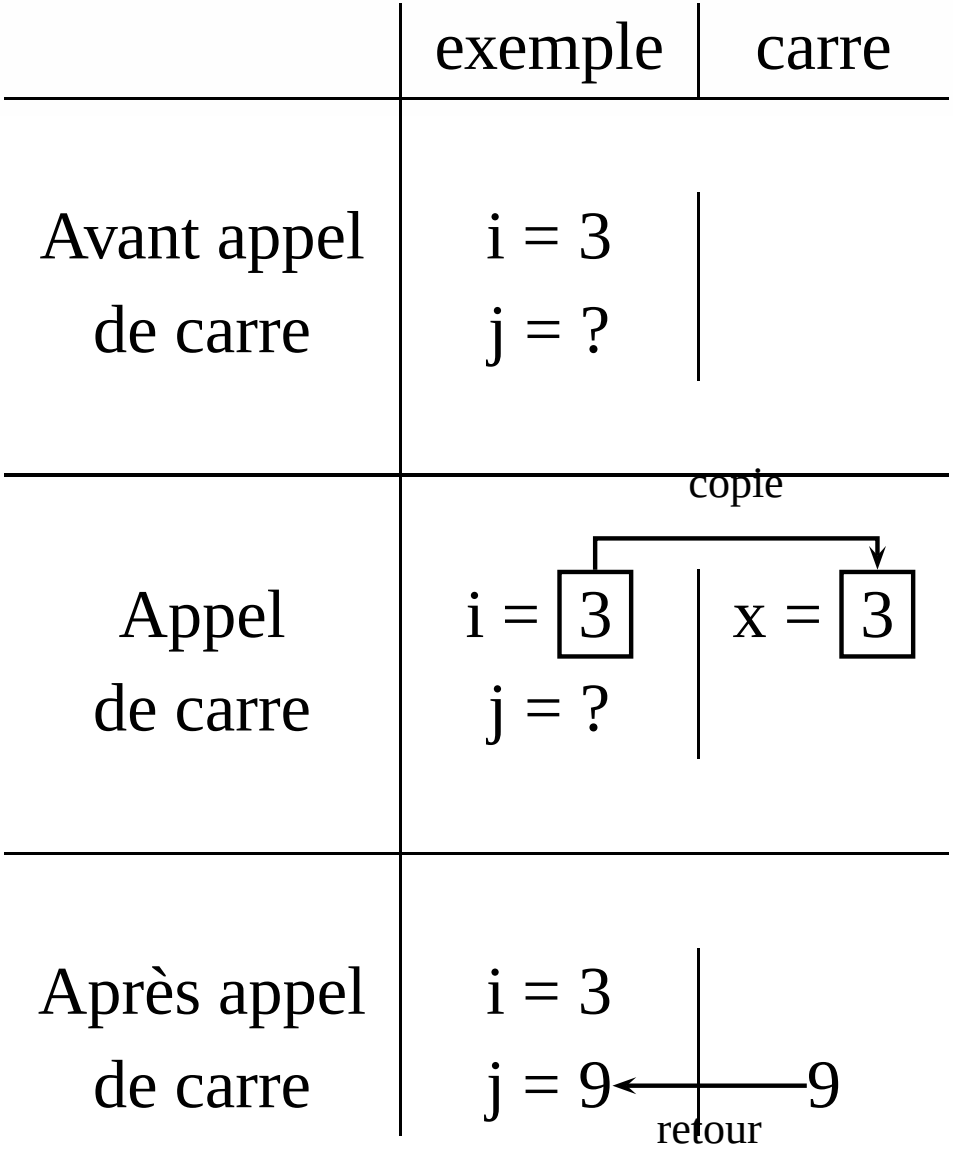
- Il n’y a pas de procédure en C
  - Pour traduire une procédure, on crée une fonction qui ne retourne pas de valeur, on utilise alors le type `void` comme type de retour
- Tous les passages de paramètres en C sont des passages de paramètres en entrée (on parle de passage par valeur)
  - Lorsque l’on veut traduire des passages de paramètres en sortie ou en entrée/sortie on utilise les pointeurs (on parle de passage de paramètre par adresse)
    - On passe le pointeur sur la variable en lieu et place de la variable

Passage de paramètre par valeur...

```
int carre (int x){
   return (x*x);
}

void exemple (){
   int i = 3;
   int j;

   j = carre (i);
   printf("%d",i2);
}
```



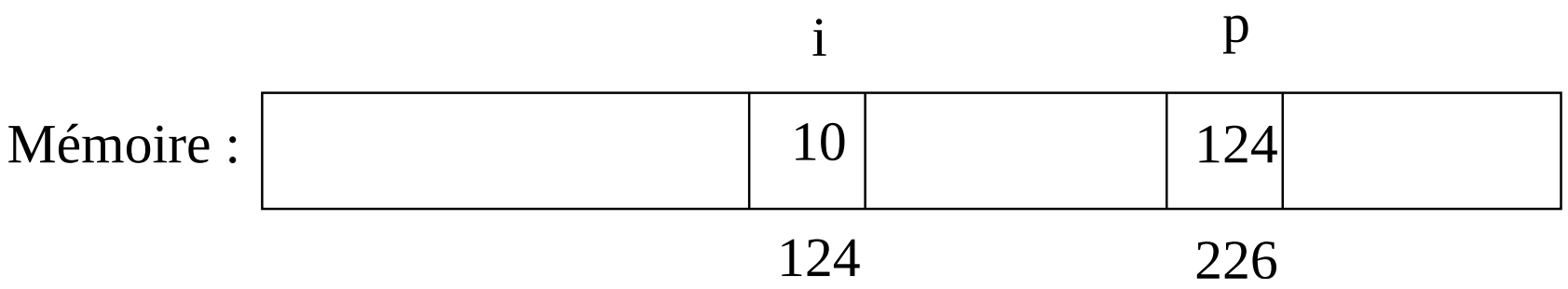
Passage de paramètre par adresse...

```
void remiseAZero(int* p) {
   *p=0;
}

void exemple2() {
   int i=10;

   remiseAZero(&i);
   printf("%d",i);
}
```

	exemple	carre
Avant appel	i = 3	
Appel de remiseAZero	i = 3 &i	p = 124
	copie	
Après appel de remiseAZero	i = 0	



Traduire `écrire`...

- L'instruction `printf` (du module `stdio.h`) permet d'afficher des informations à l'écran
- Syntaxe :

```
printf("chaîne de caractères"[,variables])
```

  - Si des variables suivent la chaîne de caractères, cette dernière doit spécifier comment présenter ces variables :
    - %d pour les entiers (int, short, long)
    - %f pour les réels (float, double)
    - %s pour les chaînes de caractères
    - %c pour les caractères
  - La chaîne de caractères peut contenir des caractères spéciaux :
    - \n pour le retour chariot
    - \t pour les tabulations

Traduire `écrire`...

- Par exemple :

```
int i=1;
float x=2.0;
printf("Bonjour\n");
printf("i = %d\n",i);
printf("i = %d, x = %f\n",i,x);
```

- ...affiche :

```
Bonjour
i = 1
i = 1, x=2.0
```

Traduire `lire`...

- L'instruction `scanf` (du module `stdio.h`) permet à l'utilisateur de saisir des informations au clavier
- Syntaxe :

```
scanf("chaîne de formatage",pointeur var1 , ...)
```

  - La chaîne de formatage spécifie le type des données attendues, par exemple :
    - %d pour les entiers (int, short, long)
    - %f pour les réels (fbat, double)
    - %s pour les chaînes de caractères
    - %c pour les caractères

Traduire `lire`...

- Par exemple :

```
int i;
float x;
scanf("%d%f",&i,&x);
```

Traduire les programmes...

- Le programme principal en C est une fonction (comme une autre) dont le nom est `main` et son type de retour est `int`

Programme *nom du programme*
Définition des constantes
Définition des types
Déclaration des variables globales
Définition des sous-programmes
début
instructions du programme principal
fin

$\Rightarrow$

```
// inclure les librairies (#include)
// definir les constantes (#define)
// definir les types
// declarer les variables globales
// definir les fonctions
int main () {
  // instructions
}
```

Un exemple...

- Nous allons traduire le programme suivant :

```
Programme NombrePremier
    Déclaration n : Entier; c : Caratère
    fonction estPremier (a : Entier) : Booléen
début
    répéter
        écrire(Entrez un entier :)
        lire(n)
        si estPremier(n) alors
            écrire(n," est premier")
        sinon
            écrire(n," n'est pas premier")
        finsi
        répéter
            écrire("Voulez vous tester un autre nombre (O/N) :")
            lire(()  c);
        jusqu'à ce que c='O' ou c='o' ou c='n' ou c='N'
    jusqu'à ce que c='n' ou c='N'
fin
```

Un exemple...

- avec :

fonction estPremier **(**a : **Entier) : Booléen**

Déclaration i : **Entier**

début

i ← 2

tant que i<a div 2 et a mod i≠0 **faire**

i ← i+1

fintantque

si a mod i=0 **alors**

retourner Faux

sinon

retourner Vrai

finsi

fin

Un exemple...

- Le programme C correspondant :

```
#include<stdio.h>

int estPremier(int a) {
  int i=2;
  while ((i<(a/2)) && ((a%i)!=0))
    i++;
  if ((a%i)==0)
    return 0;
  else
    return 1;
}

int main() {
  int n;
  char c;
  do {
    printf("Entrez un nombre entier");
    scanf("%d",&n);
    if (estPremier(n))
      printf("%d est premier\n",n);
    else
```

Un exemple...

```
      printf("%d n'est pas premier\n",n);
    do {
      printf("Voulez vous tester un autre nombre (O/N) :");
      scanf("%c",&c);
    }while (c!='o' && c!='O' && c!='n' && c!='N');
  }while(c=='o' || c=='O');
}
```