

Entrées sorties et variables en C

Séance 1

de l'ECUE « introduction à la programmation »

Bruno Bouzy

bruno.bouzy@parisdescartes.fr

Premier programme C

```
// premierProg.c
#include <stdio.h>
int main() {
    printf("Bonjour.\n");
    return (0);
}
```

- Le « source »
- Ecrit en langage C avec un éditeur de texte

Premier programme C

```
// premierProg.c  
#include <stdio.h>  
int main() {  
    printf("Bonjour.\n");  
    return (0);  
}
```

- Titre du fichier source
- **//** indique un commentaire sur une ligne

Premier programme C

```
// premierProg.c
#include <stdio.h>
int main() {
    printf("Bonjour.\n");
    return (0);
}
```

- Inclusion de l'en-tête pour les entrées sorties
- **stdio.h**: en-tête pour les entrées sorties
- **#include**: pour inclure un fichier en-tête

Premier programme C

```
// premierProg.c
#include <stdio.h>
int main() {
    printf("Bonjour.\n");
    return (0);
}
```

- Le « programme principal »
- `main()` est une fonction unique et obligatoire
- `int` type du retour

Premier programme C

```
// premierProg.c
#include <stdio.h>
int main() {
    printf("Bonjour. \n");
    return (0);
}
```

- Imprimer « Bonjour. » sur l'écran
- `printf()` : fonction d'impression usuelle
- `Bonjour. \n` : ce que l'on veut imprimer
- `\n`: saut de ligne

Premier programme C

```
// premierProg.c
#include <stdio.h>
int main() {
    printf("Bonjour.\n");
    return (0);
}
```

- Fin de l'exécution d'une fonction
- `return` : retour de la fonction
- `0` : la valeur de retour vaut 0

Premier programme C

```
// premierProg.c
#include <stdio.h>
int main() {
    printf("Bonjour.\n");
    return (0);
}
```

- Une instruction se termine par un ;
- Un programme est une suite d'instructions.

Premier programme C

```
// premierProg.c
#include <stdio.h>
int main() {
    printf("Bonjour.\n");
    return (0);
}
```

- Les { } marquent le début et la fin d'un « bloc ».
- Un bloc regroupent des instructions.

Premier programme C

- La compilation avec **gcc**

```
ProgC > gcc premierProg.c
```

```
ProgC > ls
```

```
a.out      premierProg.c
```

```
ProgC >
```

- **gcc** : **GNU C Compiler**
- gcc s'exécute en ligne de commandes Linux
- **a.out**: l'exécutable

Premier programme C

- L'exécution

```
ProgC > ./a.out
```

```
Bonjour.
```

```
ProgC >
```

- `a.out`: l'exécutable
- L'exécution de `a.out` imprime `Bonjour.` sur l'écran.

Premier programme C

- Résumé
 - On édite
 - avec un éditeur de texte (emacs, textedit, gedit, vi)
 - On compile
 - avec gcc
 - On exécute
 - Sous la forme d'une commande Linux

Premier programme C

- Spécifier le nom de l'exécutable avec l'option `-o`

```
ProgC > gcc -o toto premierProg.c
```

```
ProgC > ls
```

```
toto    premierProg.c
```

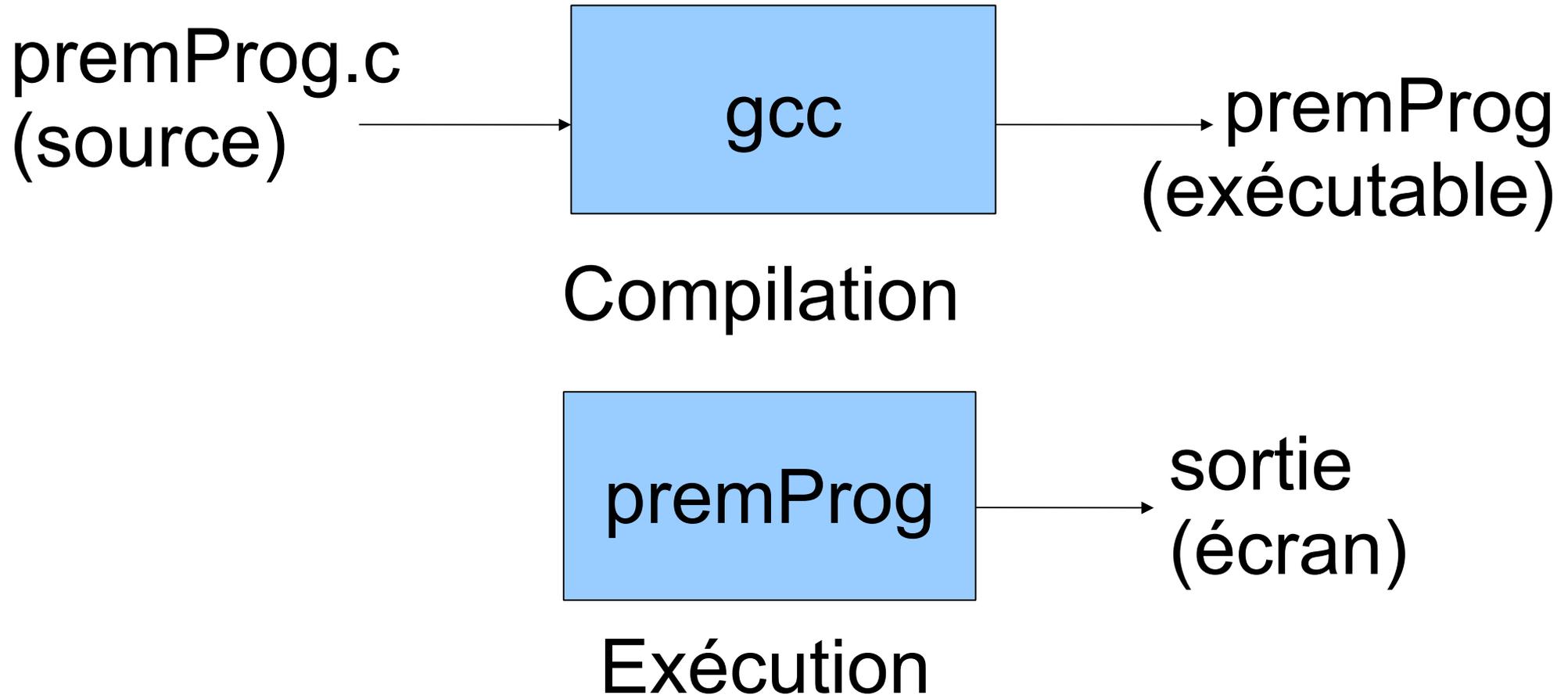
```
ProgC > ./toto
```

```
Bonjour.
```

```
ProgC >
```

- **toto**: l'exécutable
- L'exécutable est le produit de la compilation.

Premier programme C



Première variable C

```
// var.c
#include <stdio.h>

int main() {
    printf("Bonjour.\n");
    int a;
    a=3;
    printf("a = %d\n", a);
    printf("Au revoir.\n");
    return (0);
}
```

Première variable C

- `int a;`
- **Déclaration** de la variable `a`
- Crée un espace mémoire pour la variable `a`.
- Notation graphique:



Première variable C

- `a = 3;`
- Met 3 dans l'emplacement mémoire de `a`.
- **Affectation** d'une valeur à la variable `a`.
- Notation graphique:



Première variable C

```
printf("a = %d\n", a);
```

- `%d` est le format correspondant au type `int`.
- Affiche `a = 3` à l'écran.

```
ProgC > gcc -o var var.c
```

```
ProgC > ./var
```

```
a = 3
```

```
ProgC >
```

Première entrée

```
// varScanf.c
#include <stdio.h>
int main() {
    printf("Bonjour.\n");
    int x;
    printf("Tapez une valeur : ");
    scanf("%d", &x);
    printf("Vous avez tape <%d>.\n", x);
    printf("Au revoir.\n");
    return (0);
}
```

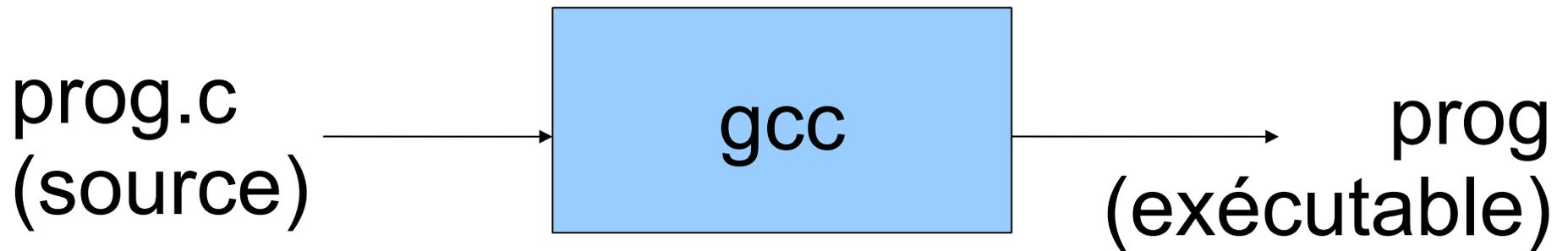
Première entrée

- Compilation exécution:

```
ProgC > gcc varScanf.c
ProgC > ./a.out
Bonjour.
Tapez une valeur: 7
Vous avez tape <7>.
Au revoir.
ProgC >
```

- L'utilisateur a tapé 7 et 'entrée' au clavier.

compilation exécution entrées sorties



Compilation



Exécution

Première opération

```
// varAdd.c
#include <stdio.h>
int main() {
    int a=3, b=7, c;
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    c = a + b;
    printf("La somme est %d\n", c);
    return (0);
}
```

Première opération

- Exécution:

```
ProgC > ./a.out
```

```
a = 3
```

```
b = 7
```

```
La somme est 10
```

```
ProgC >
```

Première opération

- En mémoire:

```
int a=3, b=7, c;
```

```
c = a + b;
```

a	3
int	

b	7
int	

c	?
int	

a	3
int	

b	7
int	

c	10
int	

Première opération

- Pas de printf:

```
// varAdd.c
#include <stdio.h>
int main() {
    int a=3, b=7, c;
    c = a + b;
    return (0);
}
```

- Exécution:

```
ProgC > ./a.out
ProgC >
```

- Rien sur l'écran!

Première calculette

- Les opérandes sont entrées au clavier:

```
int main() {  
    int a, b, c;  
    printf("Valeur 1 ? ");  
    scanf("%d", &a);  
    printf("Valeur 2 ? ");  
    scanf("%d", &b);  
    c = a + b;  
    printf("Somme = %d\n", c);  
    return (0);  
}
```

Première calculette

- Exécution:

```
ProgC > ./a.out
```

```
Valeur 1 ? 2
```

```
Valeur 2 ? 5
```

```
Somme = 7
```

```
ProgC >
```

Premier nombre réel

```
int main() {  
    float y;  
    printf("Tapez un reel : ");  
    scanf("%f", &y);  
    printf("Le reel est %f.\n", y);  
    return (0);  
}
```

- `float`: type réel « flottant » (avec virgule)
- `%f` : format d'entrée sortie des `float`

Premier nombre réel

- Exécution:

```
ProgC > ./a.out  
Tapez un reel : 2.71  
Le reel est 2.710000.  
ProgC >
```

- nombre réel entré: 2 chiffres après la virgule
- sortie par défaut: 6 chiffres après la virgule

Précision

```
int main() {
    float x = 3.5;
    int y = 2;
    float z = x/y;
    printf("%2.2f / %d = %2.2f\n", x, y, z);
    printf("%2.4f / %d = %2.4f\n", x, y, z);
    printf("%10.2f / %d = %10.2f\n", x, y, z);
    printf("%10.4f / %d = %10.4f\n", x, y, z);
    return (0);
}
```

- **%m.nf (au lieu de %f) signifie:**
 - Largeur d'affichage minimal par défaut = m
 - Precision = n

Précision

- Exécution:

```
ProgC > ./a.out
```

```
3.50 / 2 = 1.75
```

```
3.5000 / 2 = 1.7500
```

```
    3.50 / 2 =          1.75
```

```
    3.5000 / 2 =        1.7500
```

```
ProgC >
```

- Nombre de caractères pour afficher = 4 ou 10
- Nombre de chiffres après la virgules = 2 ou 4

Entrée d'un caractère unique

- En ligne de commandes, un programme demande souvent un caractère à l'utilisateur.

```
ProgC > ./a.out  
caractere ? b  
caractere = b  
(code ascii=98)  
ProgC >
```

- L'utilisateur a tapé **b** puis 'entrée'.

Entrée d'un caractère unique

```
// scanfChar.c
#include <stdio.h>
int main() {
    char m;
    printf("caractere ? ");
    scanf("%c", &m);
    printf("caractere = %c\n", m);
    printf("(code ascii=%d)\n", m);
    return (0);
}
```

- **%c** : format d'entrée sortie pour les `char`
- **%d** : représentation `int` du `char`
- **code ascii d'un `char`** : nombre entier de 0 à 255

Entrée de 2 caractères

- Un programme peut demander 2 caractères:

```
ProgC > ./a.out  
caractere 1 ? q  
caractere 1 = q  
(code ascii=113)  
caractere 2 ? w  
caractere 2 = w  
(code ascii=119)  
ProgC >
```

Entrée de 2 caractères

```
int main() {
    char m, n;
    printf("caractere 1 ? ");
    scanf("%c", &m);
    printf("caractere 1 = %c\n", m);
    printf("(code ascii=%d)\n", m);
    printf("caractere 2 ? ");
    scanf("%c", &n);
    printf("caractere 2 = %c\n", n);
    printf("(code ascii=%d)\n", n);
    return (0);
}
```

- Ce programme ne marche pas. Pourquoi ?

Entrée de 2 caractères

- Le `scanf` de `char` avec `%c` lit **tous** les caractères tapés au clavier.
- L'utilisateur doit taper 4 caractères:
 - Un premier caractère
 - 'entrée'
 - Un second caractère
 - 'entrée'
- 'entree' correspond au saut de ligne '`\n`' de code `ascii = 10`

Entrée de 2 caractères

```
int main() {  
    char m, n, bidon;  
    printf("caractere 1 ? ");  
    scanf("%c", &m);  
    scanf("%c", &bidon);  
    printf("caractere 1 = %c\n", m);  
    printf("caractere 2 ? ");  
    scanf("%c", &n);  
    scanf("%c", &bidon);  
    printf("caractere 2 = %c\n", n);  
    return (0);  
}
```

- Ce programme marche
- `bidon` reçoit les '`\n`' correspondant à 'entrée'

Entrée de 2 entiers

- Pourquoi la première calculette marchait ?
- Elle demandait 2 entiers sans variable `bidon` !
- Réponse:
 - `%d` est une entrée sortie **élaborée** qui extrait le `int` à partir des caractères tapés au clavier (les chiffres successifs et le '`\n`')
 - `%c` est une entrée sortie du niveau caractère
 - Les autres formats (`%f` et `%s` sont élaborés)

Entrée de 2 caractères

```
int main() {  
    char m[2], n[2];  
    printf("caractere 1 ? "); scanf("%s", m);  
    printf("caractere 1 = %c\n", m[0]);  
    printf("caractere 2 ? "); scanf("%s", n);  
    printf("caractere 2 = %c\n", n[0]);  
    return (0);  
}
```

- Ce programme marche
- **m** et **n** : tableaux de 2 caractères
- **%s** : format tableau de caractères
- **m[0]** et **n[0]** premières cases du tableau

Entrée de 2 caractères

- En mémoire:

```
char m[2], n[2];
```

m
char[2]

'q'	'\n'
-----	------

n
char[2]

'w'	'\n'
-----	------

Constante

```
// constante.c
#include <stdio.h>

#define CONS 10
int main() {
    int a, b;
    printf("Valeur ? "); scanf("%d", &a);
    b = a + CONS;
    printf("Valeur + constante %d = %d\n", CONS, b);
    return (0);
}
```

- `#define` définit une « constante » `CONS`
- `gcc` remplace `CONS` par `10` dans le source

fflush

- `fflush` : vide le contenu de `printf` sur la sortie
- `stdout`: sortie standard (l'écran, un fichier)

```
int main() {  
    int x;  
    printf("Tapez un entier: ");  
    fflush(stdout);  
    scanf("%d", &x);  
    printf("Entier = %d\n", x);  
    return (0);  
}
```

Les types pré-définis (1/2)

- les plus couramment utilisés...
 - `int`
 - Nombre entier sur 4 octets (32 bits)
 - `char`
 - Caractère ou nombre entier sur 1 octet (8 bits)
 - `float`
 - Nombre réel (flottant) simple (4 octets)
 - `double`
 - Nombre réel double précision (8 octets)

Les types pré-définis (2/2)

- suite...
 - `char *`
 - Pointeur sur caractère
 - `int *`
 - Pointeur sur nombre entier
 - `unsigned int`
 - Nombre entier positif ou nul (32 bits)
 - `unsigned char`
 - Nombre entier positif ou nul ≤ 255 (8 bits)

sizeof()

- `sizeof (t)` : taille du type `t` donnée en octets

```
printf("sizeof(char) = %d\n", sizeof(char));
printf("sizeof(short) = %d\n", sizeof(short));
printf("sizeof(int) = %d\n", sizeof(int));
printf("sizeof(long) = %d\n", sizeof(long));
printf("sizeof(long long) = %d\n", sizeof(long long));
printf("sizeof(float) = %d\n", sizeof(float));
printf("sizeof(double) = %d\n", sizeof(double));
printf("sizeof(char *) = %d\n", sizeof(char *));
printf("sizeof(int *) = %d\n", sizeof(int *));
printf("sizeof(unsigned char) = %d\n", sizeof(unsigned
char));
printf("sizeof(unsigned int) = %d\n", sizeof(unsigned int));
```

sizeof()

- Taille en octets des principaux types pré-définis

```
sizeof(char) = 1
sizeof(short) = 2
sizeof(int) = 4
sizeof(long) = 4
sizeof(long long) = 8
sizeof(float) = 4
sizeof(double) = 8
sizeof(char *) = 4
sizeof(int *) = 4
sizeof(unsigned char) = 1
sizeof(unsigned int) = 4
```

Résumé de la séance 1

- 1er programme C
- compilation (`gcc`), exécution, source, exécutable
- entrée, sortie, clavier, écran
- variable, mémoire, déclaration, affectation
- opérateur
- lecture d'entiers, réels, caractères
- constante, `fflush`, `sizeof`