Fonctions, procédures et conditionnelle

A. Lambert/P. Courtieu

DSP - USAL 34

Construction d'un programme

Comment construire un programme ?

- Programmer = définir des procédures et fonctions
- Programme = pyramide de fonctions avec la fonction `main` au sommet

Quand une procédure/fonction est définie, elle est :

- utilisable comme les autres,
- aucune différence entre vos fonctions/procédures et celles des bibliothèques (`lireInt()`, etc).

Définition de procédures

Définir une procédure

Syntaxe :

- Mot clé `void` : déclare une procédure (fonction sans résultat)
- Nom de la procédure
- Paramètres entre parenthèses (même si aucun)
- Corps de la fonction :
  instructions à réaliser à chaque appel de la procédure

```
void proc(paramètres) {

  instructions

}
```

Définir une procédure – Exemple

Pas d'argument

```
void proc() {

   ...

}
```

Deux arguments

```
void proc(int x, int y) {

   ... x ... y ...

}
```

Arguments formels (decl.)

Argument formel (util.)

Définir une procédure – Exemple

```
void ecrireAdd(int x, int y) {
  ecrireInt(x);
  ecrireString(" + ");
  ecrireInt(y);
  ecrireString(" = ");
  ecrireInt(x+y);
  ecrireSautDeLigne();
}
```

P.S. Sans oublier le `#include"inout.h"`.

Exemple de procédure avec tests (1/2)

```
void ecrireAdd (int x, int y) {
  ecrireInt(x);
  ecrireString(" + ");
  ecrireInt(y);
  ecrireString(" = ");
  ecrireInt(x+y);
  ecrireSautDeLigne();
}

void main() {
  ecrireString("Début\n");
  ecrireAdd(8,6);
  ecrireAdd(7,13);
  int a = lireInt();
  int b = lireInt();
  ecrireAdd(a,b);
  ecrireString("Fin\n");
}
```

paramètres formels

procédure externe

procédure définie ci-dessus

paramètres réels

procédure externe

Exemple de procédure avec tests (2/2)

```
void ecrireAdd (int x, int y) {
  ecrireInt(x);
  ecrireString(" + ");
  ecrireInt(y);
  ecrireString(" = ");
  ecrireInt(x+y);
  ecrireSautDeLigne();
}

void testecrireAdd() {
  ecrireString("Début test ecrireAdd\n");
  ecrireAdd(8,6);
  ecrireAdd(7,13);
  int a = lireInt();
  int b = lireInt();
  ecrireAdd(a,b);
  ecrireString("Fin test ecrireAdd\n");
}

void main() {
  testecrireAdd();
}
```

Portée des variables (1/3)

```
void ecrireAdd(int x, int y) {
  ecrireInt(x);
  ecrireString(" + ");
  ecrireInt(y);
  ecrireString(" = ");
  ecrireInt(x+y);
  ecrireSautDeLigne();
}

void testecrireAdd() {
  ecrireString("Début test ecrireAdd\n");
  ecrireAdd(8,6);
  ecrireAdd(7,13);
  int a = lireInt();
  int b = lireInt();
  ecrireAdd(a,b);
  ecrireString("Fin test ecrireAdd\n");
}
```

paramètres **locaux**

~~x~~ ~~y~~ non visibles

~~X~~ ~~Y~~ non visibles

Déclarations **locale**

Portée des variables (2/3)

```
void ecrireAdd(int x, int y) {
  ecrireInt(x);
  ecrireString(" + ");
  ecrireInt(y);
  ecrireString(" = ");
  ecrireInt(x+y);
  ecrireSautDeLigne();
}

void testecrireAdd() {
  ecrireString("Début test ecrireAdd\n");
  int x = 5;
  int y = 2;
  ecrireAdd(x,y);
  ecrireAdd(7,13);
}
```

Rien à voir!

Portée des variables (3/3)

Règle générale très importante

Seules informations transmises à une procédure/fonction lors de l'appel : **valeurs (~~noms~~)** des paramètres

P.S. Sauf ~~variables globales~~ mais c'est mal (et interdit ici)

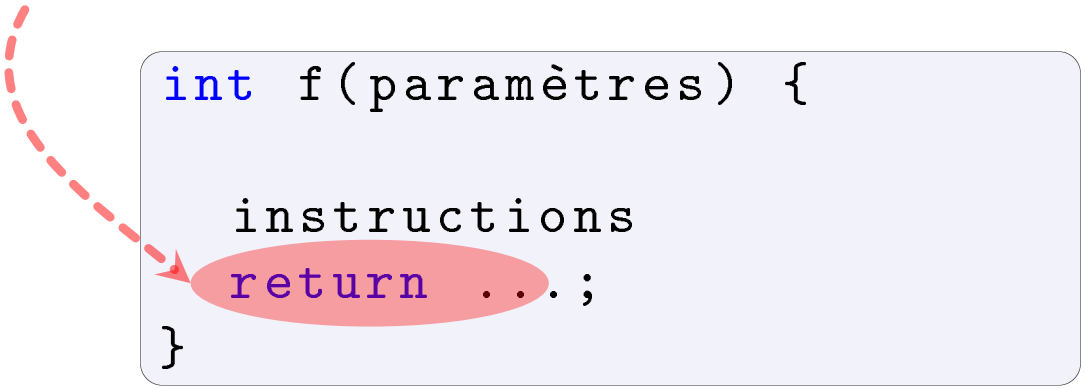
Parce-que

Définition de fonction

Définir une fonction

Syntaxe :

- Type de la valeur retournée (~~void~~)
- Nom de la fonction
- Paramètres entre parenthèses (même si aucun)
- Corps de la fonction :
  - instructions à réaliser à chaque appel de la fonction
  - + **Valeur à retourner**



Définir une fonction

```
int f(paramètres) {

   instructions
   return ...;
}
```

doit être un int

Définir une fonction – Exemples (1/2)

```
int oppose (int x) {
  return -x;
}
```

```
int x2plus2xyplusy2 (int x,int y) {
  return x*x + 2*x*y + y*y;
}
```

```
int max (int x, int y) {
  if (x>y) { return x; }
  else { return y; }
}
```

Définir une fonction – Exemples (2/2)

```
int somme3 (int x,int y,int z) {
  int res = x;
  res = res + y;
  res = res + z;
  return res;
}

void testsomme3 () {
  ecrireInt(somme3(2,7,11));
  int x = somme3(5,6,7);
  ecrireInt(x);
}
void main() {
  testsomme3();
}
```

La structure de contrôle conditionnelle

La conditionnelle

Rôle :

- spécifier différents cas possibles à l'exécution (« conditions »)
- spécifier un comportement différent pour chaque cas

Fonctionnement :

1. Tester une condition
2. Exécuter la séquence d'instructions relative au résultat du test

```
if (cond) {
   ... // instructions si test vrai
}
else {
   ... // instructions si test faux
}
```

« tester » une condition $\neq$ « tester » une fonction

La conditionnelle en C

```
if (cond) {
   ... // instructions si test vrai
}
else {
   ... // instructions si test faux
}
```

- On peut omettre le `else` si la séquence d'instructions associée est vide.
- Si il n'y a pas d'accolades, la suite d'instructions est composée d'une seule ligne
- Si il y a plus d'1 instruction, mettre des accolades
- Conseil : Mettez toujours les accolades

La conditionnelle en C - Exemple

```
int x;
x = lireInt();
if (x >= 0) {
  ecrireString("nombre positif ou nul.");
}
else {
  ecrireString("nombre strictement négatif.");
}
ecrireString("\n");
```

1. évaluation de la condition (`x >= 0`).
   - si condition *vraie* on exécute la première séquence d'instructions
   - sinon on exécute la deuxième séquence d'instructions
2. enfin, on exécute de toute façon `ecrireString("\n")`

Les conditions

```
x <= 0
```

Une condition :

- est une expression ($\neq$ instruction)
- retourne une valeur à l'exécution

  valeurs possibles ? « VRAI » ou « FAUX »

Le type `boolean` n'existe pas en C ($\neq$ Java, C++, etc) !

$\Rightarrow$ on retourne un `int`

- valeur 0 : FAUX
- valeur $\neq 0$: VRAI

Le typage des conditions

A ne pas faire !

```
int x = 0;
if (x = 0) {
  ecrireString("Oui");
} else {
  ecrireString("Non");
}
```

- `x = 0` ~~test~~ affectation !
- En `C` : l'affectation retourne la valeur de droite
- la condition retourne donc 0 reconnu comme FALSE
- C'est alors la branche du `else` qui est exécutée

Le typage des consitions

A ne pas faire !

```
int x = 0;
if (x = 0) {
  ecrireString("Oui");
} else {
  ecrireString("Non");
}
```

Correction

```
int x = 0;
if (x == 0) {
  ecrireString("Oui");
} else {
  ecrireString("Non");
}
```

- `x = 0` ~~test~~ affectation !
- En `C` : l'affectation retourne la valeur de droite
- la condition retourne donc 0 reconnu comme FALSE
- C'est alors la branche du `else` qui est exécutée

Les opérateurs de conditions

- `e1 == e2` et `e1 != e2`
- `e1 <= e2` et `e1 >= e2`
- `e1 < e2` et `e1 > e2`
- `!test`
- `test1 && test2` et `test1 || test2`

Tester le résultat d'une fonction : exemple

```
int add3(int x, int y, int z){
  return x + y + z;
}

void testerAdd3 () {
  int res = add3(2,3,4);
  if (res != 9) {
    ecrireString("Erreur sur add3!");
    ecrireString("Attendu: 9");
    ecrireString("Obtenu: ");
    ecrireInt(res);
    ecrireSautDeLigne();
  }
}

void main() {
  testerAdd3();
}
```