Fonctions et Procédures de test [ss]
Résumé de cours

Université de Haute Alsace

Unisciel algoprog Version 15 mai 2018

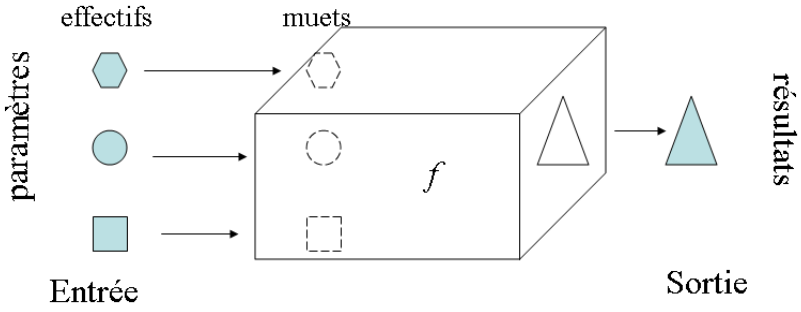
Table des matières

1 C - Résumé de cours

1.1 Algorithme d'un module

Module (procédure ou fonction)
Bloc d'instructions ayant un **début** et une **fin**, identifié par un **nom** (l'identifiant) et associé à la définition d'une interface par le biais d'une spécification de **paramètres**.

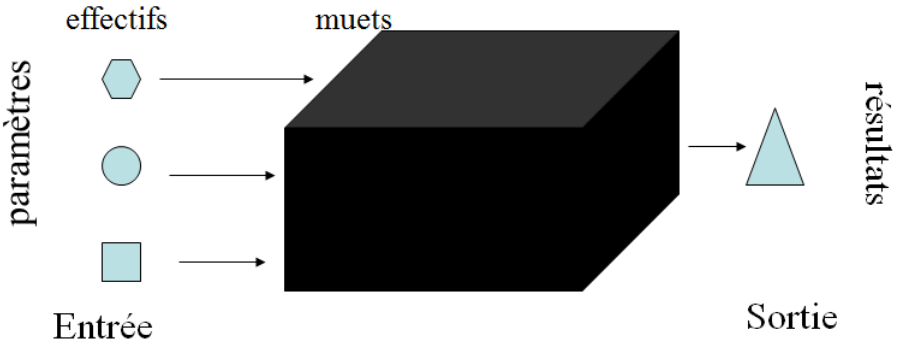
Algorithme d'un module
Description de ce qu'il y a à l'intérieur de la « boîte » qui permet la production du résultat à partir des paramètres.



Élaborer un algorithme
C'est construire la « boîte ».
Ceci peut se faire à partir d'autres modules déjà existants.

Variable locale
Toute variable est **locale** au module dans lequel elle apparaît, ce qui veut dire que son existence est ignorée en dehors de ce module.

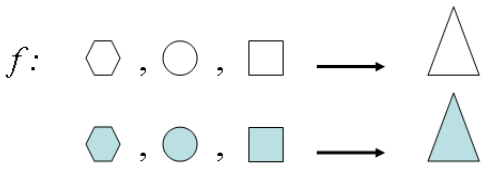
De façon imagée, vous ne voyez pas ce qu'il y a à l'intérieur de la « boite noire ».



1.2 Passage de paramètre

Passage de paramètre par position
Les paramètres effectifs sont associés **dans le même ordre** aux paramètres formels :

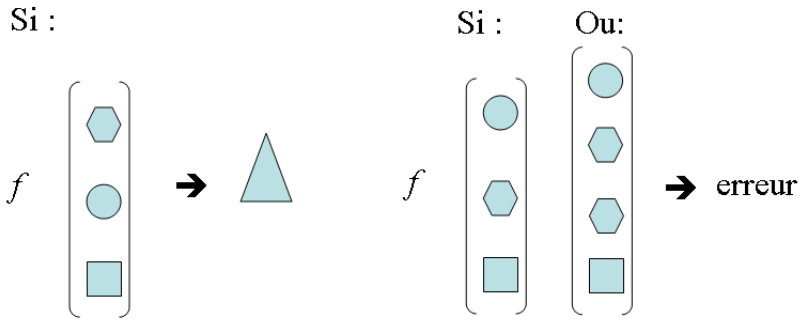
- Le premier argument (= paramètre effectif) au premier paramètre formel.
- Le deuxième effectif au deuxième formel.
- etc.



f : ensemble de départ → ensemble d'arrivée
f : paramètres formels → description du résultat

Règles à respecter
Deux règles :

- **Même nombre** d'arguments que de paramètres.
- **Types et arités** des arguments/paramètres doivent correspondre.



1.3 Schéma d'une fonction

C/C++ **Schéma d'une fonction**

```
TypeRes nomFcn(TypeParam1 param1,TypeParam2 param2,...)
{
  TypeRes resultat = valeurInitiale;
  calcul_du_resultat;
  return resultat;
}
```

C/C++ **Expression fonctionnelle**

```
TypeRes nomFcn(TypeParam1 param1,TypeParam2 param2,...)
{
  return expression;
}
```

C/C++ : Primitive return

Quelles que soient les situations (conditions), il doit **toujours** y avoir **une exécution** de la primitive `return` (sinon le résultat est aléatoire).

1.4 Appel d'un module

Appel d'un module

Il suffit de spécifier son nom (avec ses paramètres effectifs).

Mécanisme d'évaluation d'un module

Celui-ci s'effectue comme suit :

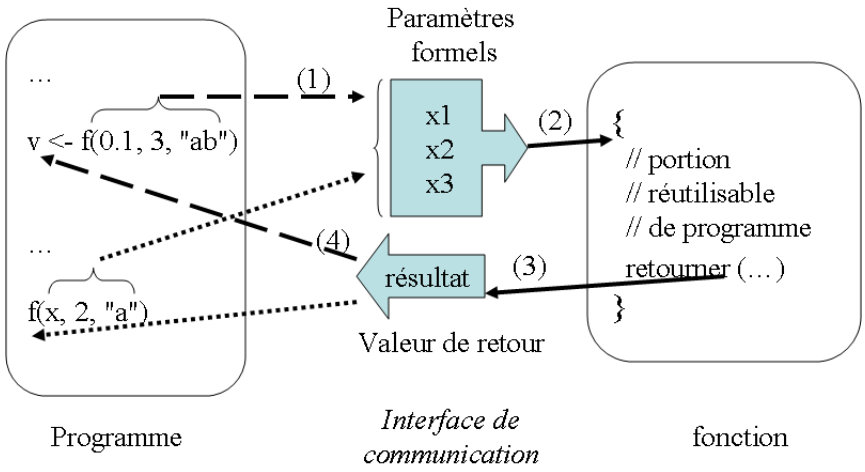
(0) Évaluation de l'identifiant `f` et des paramètres effectifs

(1) Copie ou liaison des paramètres formels aux paramètres effectifs

(2) Réalisation du calcul dans cet environnement

(3) Retour de la valeur (en cas de fonction)

(4) Continuation du programme après l'appel de `f`



1.5 Procédure de test

Motivation

L'encapsulation d'un programme dans une **procédure de test** `test_xxx` permet de :

1. Décomposer un problème (= l'algorithme) en sous-problèmes (= les modules).
2. Vérifier chacun des sous-problèmes de façon plus ou moins indépendante.
3. Valider le problème final en écrivant un unique algorithme (= le programme).

C/C++ **Procédure de test**

```c
void test_xxx()
{
  // écriture des instructions du mini-"main"
}
int main()
{
   test_xxx(); // exécution de la procédure de test
}
```