ULCO-L1-Algo2-CM : Initiation à l'algorithmique 2

Table des matières

1 Tableaux 2D **2**
- 1.1 Notion de tableau 2D . . . 2
- 1.2 Déclarer un tableau 2D . . . 2
- 1.3 Accéder à un tableau 2D . . . 2
- 1.4 Parcourir un tableau 2D . . . 2
- 1.5 Exemple : notes de TP d'une classe . . . 3
- 1.6 Exercices . . . 3

2 Fonctions **5**
- 2.1 Notion de fonction . . . 5
- 2.2 Définir une fonction . . . 5
- 2.3 Appeler une fonction . . . 5
- 2.4 Paramètres formels et paramètres effectifs . . . 6
- 2.5 Passage de paramètres par valeur ou par variable . . . 6
- 2.6 Exemple : calcul de la valeur absolue . . . 6
- 2.7 Exercices . . . 6

3 Enregistrements **8**
- 3.1 Notion d'enregistrement . . . 8
- 3.2 Déclarer un enregistrement . . . 8
- 3.3 Déclarer une variable de type enregistrement . . . 8
- 3.4 Utiliser une variable de type enregistrement . . . 8
- 3.5 Exemple : gestion de produits . . . 8
- 3.6 Exercices . . . 9

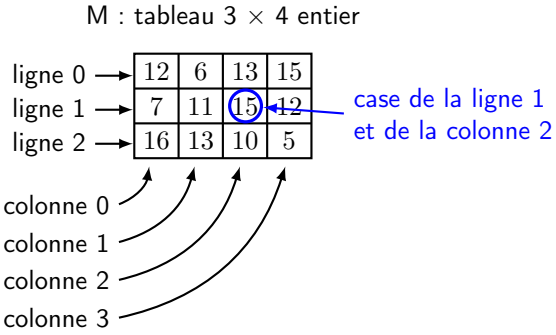
4 Récursivité **11**
- 4.1 Exemple : fonction factorielle . . . 11
- 4.2 Notion de fonction récursive . . . 11
- 4.3 Définir d'une fonction récursive . . . 11
- 4.4 Trace d'une fonction récursive . . . 12
- 4.5 Terminaison d'une fonction récursive . . . 12
- 4.6 Exercices . . . 13

5 Synthèse **14**
- 5.1 Notions d'algorithmique . . . 14
- 5.2 Méthode pour écrire un algorithme . . . 14
- 5.3 Exercices . . . 14

1 Tableaux 2D

1.1 Notion de tableau 2D

Structure de données composée de données de même type, contiguës selon deux dimensions.



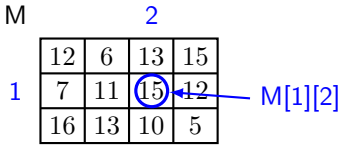
1.2 Déclarer un tableau 2D

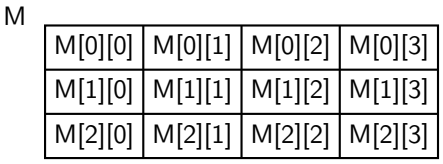
Identique à la déclaration d'un tableau 1D mais avec une taille 2D (l'exemple classique est la matrice).
Par exemple, pour déclarer un tableau M de 3 lignes et 4 colonnes d'entiers :

```
int M[3][4];
```

1.3 Accéder à un tableau 2D

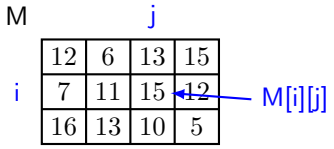
On accède à une case d'un tableau 2D en spécifiant le nom du tableau suivi de l'indice de ligne entre crochets puis de l'indice de colonne entre crochets. Les indices commencent à 0.





1.4 Parcourir un tableau 2D

On peut parcourir un tableau 2D avec deux boucles "pour" imbriquées.
Par exemple, on parcourt chaque ligne i et, à chaque ligne, on parcourt chaque colonne j.




```
for( i=0 ; i<2 ; i++)
{ // traitement de la ligne i
  for( j=0 ; i<3 ; i++)
  {  // traitement de la colonne j de la ligne i
    M[i][j] = 0;
  }
}
```

1.5 Exemple : notes de TP d'une classe

Problème :

Une classe de 20 étudiants suit un module comportant 6 TP notés. On veut écrire un programme pour gérer ces notes (les saisir, calculer la moyenne et l'écart-type...)

Exemple d'exécution :

```
Etudiant 0, note 0 ?
Etudiant 0, note 1 ?
...
Etudiant 0, note 5 ?
Etudiant 1, note 0 ?
...
Etudiant 19, note 5 ?
Moyenne = ...
Ecart-type = ...
```

Méthode :

Stocker les notes dans un tableau de taille 20×6. Ainsi, chaque ligne du tableau correspond à un étudiant et chaque colonne à un TP.

Algorithme :

```
#include <iostream>
using names std;
void main()
{
    const int NBE = 20, // nombre d'etudiants
              NBN = 6;  // nombre de notes par etudiant
    int i,
        j;
    double notes[NBE][NBN]; // tableau de notes

    for( i=0 ; i<NBE ; i++)
    { // traitement du i-eme etudiant
      for( j=0 ; j<NBN ; j++)
      {  // traitement de la j-eme note du i-eme etudiant
        cout << "Etudiant " << i << ", note " << j << " ?";
        cin >> notes[i][j];
      }
    }
}
```

1.6 Exercices

1.6.1 Moyenne générale des notes de colles

Complétez le programme de l'exemple précédent pour calculer la moyenne et l'écart-type.

1.6.2 Moyenne d'un étudiant et moyenne d'un TP

Écrivez un programme qui saisit les notes de TP, le numéro d'un étudiant et le numéro d'un TP, puis affiche la moyenne de l'étudiant saisi et la moyenne du TP saisi.

1.6.3 Nombre d'entiers positifs dans un tableau 2D

Écrivez un programme qui saisit un tableau de 3 lignes et de 2 colonnes d'entiers puis affiche le nombre d'entiers positifs contenus dans le tableau.

Faites la trace pour l'entrée (-3, -2, -1, 0, 1, 2).

2 Fonctions

2.1 Notion de fonction

Algorithme qui prend des données en entrée et retourne des données en sortie.

Deux étapes :
- définir la fonction : décrit ce qu'il faut faire avec les paramètres "formels"
- appeler la fonction : demande d'effectuer les traitements sur des paramètres "effectifs"

2.2 Définir une fonction

L'entête de fonction contient :
- les paramètres d'entrée (noms et types) et les types de retour
- le nom de la fonction
- les variables locales

Le corps de fonction contient :
- les traitements à réaliser
- une action "retourner" qui termine la fonction en spécifiant les données à sortir

```
typeRetour nomFonction(parametres)
{
    // variables locales
    // traitement
    return valeurRetour;
}
```

Remarques :
- une fonction peut ne pas avoir de paramètre d'entrée
- une fonction peut ne rien retourner

2.3 Appeler une fonction

Pour appeler une fonction on écrit le nom de la fonction suivi des paramètres effectifs entre parenthèses, **dans l'ordre des paramètres formels correspondant** :

```
nomFonction(valeurParam1, valeurParam2, ...);
```

Les valeurs de retour peut être utilisées pour :
- une affectation
- une condition
- un autre appel de fonction

```
// recuperation de la valeur de retour
variableRetour = nomFonction(valeurParam1, valeurParam2, ...);
// utilisation de la valeur dans un test (ici retour booleen)
if (nomFonction(valeurParam1, valeurParam2, ...)) ...
```

Déroulement d'un appel de fonction :
- l'ordinateur calcule tous les paramètres
- l'ordinateur réalise les traitements de la fonction
- l'ordinateur remplace l'appel de fonction par les valeurs de retour produites

2.4 Paramètres formels et paramètres effectifs

Dans une définition de fonction, les paramètres sont "formels" : ce sont juste des noms utilisés dans le corps de la fonction pour décrire les traitements à réaliser.

Lors de l'appel de la fonction, ces paramètres correspondent à des "vraies" variables, de noms certainement différents.

2.5 Passage de paramètres par valeur ou par variable

Les paramètres de types simples (entier, réel, booléen, texte) sont passés par valeur : dans le corps de la fonction, on manipule une copie du paramètre effectif.

Les paramètres de types composés (tableaux, enregistrements) sont passés par variable (ou référence) : dans le corps de la fonction, on manipule vraiment la variable passée en paramètre (effet de bord).

2.6 Exemple : calcul de la valeur absolue

```
#include <iostream>
using names std;
void main()
{
  double a, // saisi
         b; // calcul
    cin >> a;
    b = calculerAbs(a);
    cout << b;
}
```

```
double calculerAbs(double x)
{
    if (x >= 0)
    {
        return x;
    }
    else
    {
        return -x;
    }
}
```

2.7 Exercices

2.7.1 Carré d'un réel

1. Écrivez une fonction `calculerCarre` qui retourne le carré d'un réel donné.
2. Écrivez un programme qui saisit un réel puis calcule et affiche son carré.
3. Faites la trace pour l'entrée (3.0).

2.7.2 Afficher un entête

1. Écrivez une fonction `afficherTirets` à un paramètre n, qui affiche une ligne de n tirets.
2. Écrivez un programme qui affiche une ligne de 10 tirets, le texte "Ulco" puis une ligne de 20 tirets.

2.7.3 Minimum de 4 entiers

1. Écrivez une fonction `mini` qui retourne le minimum de 2 réels donnés.
2. Écrivez un programme qui saisit 4 réels et affiche le plus petit.
3. Faites la trace pour l'entrée (3.7, 1.3, 4.2, 5.1).

2.7.4 Minimum et maximum de deux entiers

1. Écrivez une fonction `calculerMinMax` qui prend 2 entiers en paramètre et retourne le minimum et le maximum.
2. Écrivez un programme qui saisit 2 entiers et affiche le minimum et le maximum.

2.7.5 Factorielle

1. Écrivez une fonction `calculerFactorielle` qui retourne la factorielle d'un entier donné. Rappel : $x! = 1 \times 2 \times ... \times x$ et $0! = 1$.
2. Écrivez un programme qui saisit un entier n, et affiche la factorielle de n.

2.7.6 Somme des éléments d'un tableau

1. Écrivez une fonction `calculerSommeTableau` à un paramètre V, de type tableau 20 entier, qui retourne la somme des éléments de V.
2. Écrivez un programme qui saisit un tableau T de 20 entiers, puis calcule et affiche la somme de ses éléments.

3 Enregistrements

3.1 Notion d'enregistrement

Structure de données permettant de contenir plusieurs données de types différents.

Trois étapes :

1. déclarer un enregistrement : décrit le nouveau type de données
2. déclarer une variable du nouveau type enregistrement
3. utiliser une variable du nouveau type enregistrement

3.2 Déclarer un enregistrement

Noeud avec :
— le nom de l'enregistrement suivi du mot-clé `enregistrement`
— la liste des champs (noms et types)

```
struct NomEnregistrement
{
    // declaration de ses membres
    type1 nomChamp1;
    type2 nomChamp2;
    type3 nomChamp3;
    ...
};
```

3.3 Déclarer une variable de type enregistrement

Un enregistrement définit un nouveau type. Créer une variable de ce type se fait comme pour tous les types :

```
NomEnregistrement nomVariable;
```

3.4 Utiliser une variable de type enregistrement

Dans une variable de type enregistrement, chaque champ est une variable classique. L'accès à un champ se fait avec l'opérateur point :

```
nomVariable.nomChamp1
```

3.5 Exemple : gestion de produits

Problème :
Pour un magasin, un produit est défini par une référence (entier), une désignation (texte) et un prix (réel). Comment gérer un ensemble de 50 produits ?

Une solution serait d'utiliser 3 tableaux (un tableau de 50 entiers pour les références, un de 50 textes pour les désignations et un de 50 réels pour les prix). Cependant, cette méthode est fastidieuse car il faut utiliser les 3 tableaux pour gérer un produit.

Méthode :
Définir un nouveau type de données `Produit` et utiliser un tableau de 50 `Produit`.

```
struct Produit
{
    // declaration de ses membres
    entier reference;
    string definition;
    double prix;
};
```

```
void saisirProduit(Produit & p)
{
  cin >> p.reference
      >> p.designation
      >> p.prix;
}
```

Algorithme :

```
#include <iostream>
using names std;
void main()
{
  const int NBP = 50; // nombre de produits

  Produit T[NBP]; // tableau de produits
  int i;
  /* saisir les produits du tableau */
  for( i=0 ; i<NBP ; i++)
  {
     saisirProdui(T[i]);
  }
  /* afficher les produits du tableau */
  for( i=0 ; i<NBP ; i++)
  {
     cout << (T[i].designation) << endl;
  }
}
```

3.6 Exercices

3.6.1 Gestion de produits

1. Écrivez une fonction `saisirUnProduit` à un paramètre p, de type Produit, qui saisit les données d'un produit.
2. Écrivez une fonction `saisir50Produits` à un paramètre tabProduits, de type tableau 50 Produit, qui saisit 50 produits.
3. Écrivez une procédure `afficherUnProduit` à un paramètre p, de type Produit, qui affiche les données d'un produit.
4. Écrivez une fonction `afficher5Produits` à un paramètre tabProduits, de type tableau 50 Produit, qui affiche 50 produits.
5. Écrivez une fonction `TrouverPlusCher` à un paramètre V, de type tableau 50 Produit, qui retourne le produit le plus cher.
6. Écrivez un programme qui saisit 50 produits, les affiche puis affiche la désignation du produit le plus cher.

3.6.2 Nombres complexes

1. Définissez un type `Complexe` représentant un nombre complexe.
2. Écrivez des fonctions permettant de saisir un complexe, d'afficher un complexe, d'ajouter 2 complexes.

3. Écrivez un programme permettant de tester ces fonctions.
4. Faites une trace.

4 Récursivité

4.1 Exemple : fonction factorielle

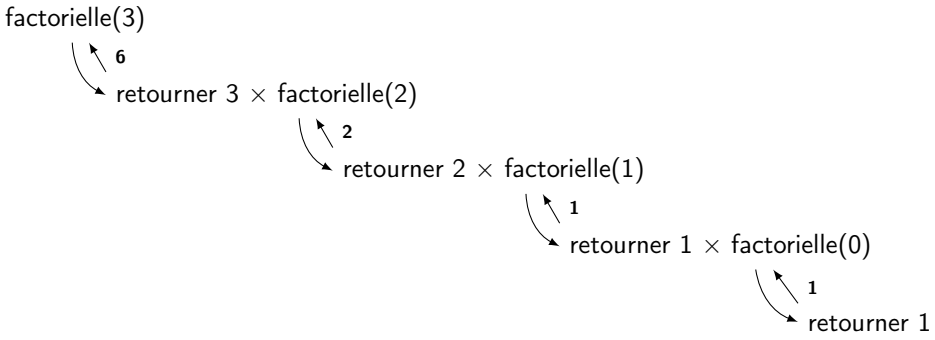
On peut définir la fonction factorielle par une relation de récurrence :

$$f(0) = 1$$
$$f(n) = n \times f(n-1) \qquad \forall n > 0$$

La traduction algorithmique directe est :

```
int factorielle(int n)
{
  if (n == 0)
  {
    return n;
  }
  else
  {
    return n * factorielle(n-1);
  }
}
```

À l'exécution, la fonction s'appelle elle-même, plusieurs fois :



4.2 Notion de fonction récursive

Une fonction récursive est une fonction qui s'appelle elle-même.

4.3 Définir d'une fonction récursive

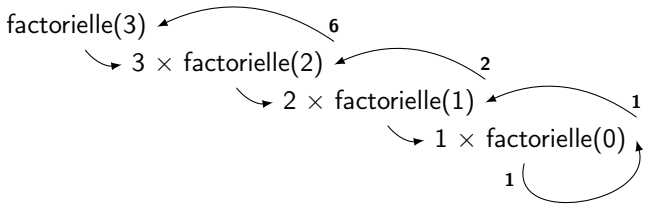
Structure d'alternative (si-alors-sinon) :
— critère d'arrêt ou cas terminal
— action terminale (termine la fonction)
— appel récursif (appelle la même fonction avec un paramètre plus proche du cas terminal)

```
typeRetour maFonctionRec(type param)
{
    if (critere arret)
    {
        // action termminale :
        // par ex. :  return valeur
    }
    else
    {
        // appel recursif
        // par ex. : return valeur * maFonctionRec(autre valeur)
    }
}
```

4.4 Trace d'une fonction récursive

Lorsqu'une fonction fait un appel récursif, elle attend que cet appel retourne avant de terminer. On a donc un empilement des appels récursifs jusqu'au critère d'arrêt puis un dépilement avec calcul du résultat.

Trace de factorielle(3) :



4.5 Terminaison d'une fonction récursive

Pour être correcte, une fonction récursive doit toujours atteindre son cas terminal. Au niveau de l'algorithme, il doit donc y avoir un cas terminal, auquel les appels récursifs doivent conduire.

```
int factorielleHS1(int n)
{
    /* ERREUR : pas de cas terminal */
        return n * factorielleHS1(n-1);
}
int factorielleHS2(int n)
{
    if (n == 0)
    {
        return n;
    }
    else
    { /* ERREUR : l'appel ne conduit pas au cas terminal... */
        return n * factorielleHS2(n+1);
    }
}
```

4.6 Exercices

4.6.1 Somme des n premiers entiers

1. Écrivez une fonction récursive `calculerSommeN` à un paramètre n, de type entier, qui retourne la somme des entiers de 1 à n.
2. Faites la trace de `calculerSommeN(4)`.
3. Écrivez un programme qui saisit un entier x puis calcule et affiche la somme des entiers de 1 à x.

4.6.2 Parité d'un entier

1. Écrivez une fonction récursive `estPair` à un paramètre n, de type entier, qui retourne `Vrai` si n est pair et `Faux` sinon (sans utiliser l'opérateur `mod`).
2. Faites la trace de `estPair(4)` et de `estPair(5)`.
3. Écrivez un programme qui saisit un entier x puis affiche si x est pair ou impair.

5 Synthèse

5.1 Notions d'algorithmique

5.1.1 Structures de données

— variable simple : une donnée, d'un type particulier (entier, réel...)
— tableau : plusieurs données de même type
— enregistrement : plusieurs données de types éventuellement différents

5.1.2 Structures de contrôle

— opérations élémentaires (affectation, opérations arithmétiques, opérations booléennes...)
— alternative : notion de choix
— boucle : notion de répétition
— fonction : notion de réutilisation

5.2 Méthode pour écrire un algorithme

— comprendre le problème à résoudre (données en entrée, résultats à produire)
— s'il y a une solution simple, l'écrire
— sinon, décomposer en sous-problèmes (fonctions, structures de données) et appliquer la même méthode pour chaque sous-problème
— vérifier que l'ensemble résout bien le problème

5.3 Exercices

5.3.1 CDthèque

Donnez un algorithme permettant de gérer une CDthèque via une interface utilisateur (ajouter des CDs, afficher la CDthèque...).

5.3.2 Tri par bulles

Écrivez un algorithme permettant de saisir 10 entiers et d'afficher ces entiers triés, en utilisant la méthode du tri par bulles.

Faites une trace pour l'entrée (8, 4, 20, 5, 4, 11, 2, 9, 10, 9).