

Licence STS

Université Claude Bernard Lyon I

# LIFAP1 : ALGORITHMIQUE ET PROGRAMMATION IMPÉRATIVE, INITIATION

1

COURS 3 : Anatomie d'un programme C

# OBJECTIFS DE LA SÉANCE

- Apprendre à écrire des programmes plus complexes
  - En utilisant des sous-programmes
- Compléter les bases d'algorithmique
  - Les fonctions et les procédures en algorithmique
- Apprendre à rendre plus lisible un programme C :
  - L'indentation du code
  - Découpage en fonctions et procédures (abstraction)

# PLAN

- Anatomie d'un programme en C
- Les sous-programmes en C / en algorithmique
  - Fonctions
  - Procédures
- L'indentation d'un programme C

# ANATOMIE D'UN PROGRAMME

- Directives de pré-compilation
  - Inclusion des déclarations des fonctions et procédures contenues dans des bibliothèques
- Définition de l'ensemble des **sous-programmes**
  - **Fonctions** et **procédures** définies par le programmeur
    - pour simplifier l'algorithme
    - ou isoler des traitements complexes et réutilisables
  - Ex : Procédure de tri, calcul d'une fonction mathématique...

# ANATOMIE D'UN PROGRAMME

- Une fonction particulière : main (le programme principal)
  - Obligatoire et **unique**
  - Exécutée (ou invoquée ou appelée) au démarrage du programme
  - Renvoie un code permettant de savoir si l'exécution du programme s'est terminée correctement

# ANATOMIE D'UN PROGRAMME

- Les bibliothèques utilisées fréquemment dans les programmes :
  - `iostream` : gestion des entrées (`in`) / sorties (`out`) sur le clavier, console...
    - `cout`, `cin`
  - `math.h` : fonctions mathématiques
    - `sin`, `cos`, `log`, `pow` ...
  - `stdlib.h` : librairie standard
    - `exit`, `rand`, `srand`, `system`
  - `string.h` : outils de manipulation des chaînes de caractères

# ANATOMIE D'UN PROGRAMME

- `#include <...>`
- Ensemble de fonctions et de procédures
- Programme principal (fonction main)
- Opérations d'entrée / sortie standards
  - Assurent le “dialogue” : machine / utilisateur
  - Afficher (écrire)  
`cout << "hello";`
  - Lire (saisir)  
`cin >> valeur;`

## CODEBLOCKS OU DEV-CPP (OU VISUAL C++)

```
#include <iostream>
using namespace std;
...

int main(void)
{
    cout << "hello" << endl;
    return 0;
}
```



## DEV-CPP (OU VISUAL C++)

Si la fenêtre se ferme tout de suite à la fin de l'exécution (possible Dev-Cpp) :

```
#include <stdlib.h>
#include <iostream>
using namespace std;
...
```

```
int main(void)
{
    cout << "hello" << endl;
```

```
    system("PAUSE");
    return 0;
```

```
}
```

Instruction permettant de conserver la fenêtre ouverte à la fin de l'exécution → appel système

# PLAN

- Anatomie d'un programme en C
- Les sous-programmes en C / en algorithmique
  - Fonctions
  - Procédures
- L'indentation d'un programme C

## LES SOUS-PROGRAMMES

- Un **sous-programme** est un sous-ensemble du **programme**
- Un programme C est constitué
  - d'un ensemble de sous-programmes
  - et d'un programme principal (main)
- Il existe deux types de sous-programmes
  - Les fonctions
  - Les procédures

## ANATOMIE D'UNE FONCTION (EN C)

- Syntaxe : **TypeRetour Nomfct (type paramètre , ...)**
  - TypeRetour = type simple (caractère, entier, réel)
  - Ensemble de paramètres formels
    - avec leur type, séparés par des “,”
- Déclaration de variables locales
  - Résultats intermédiaires
  - Compteurs de boucles
- Corps de la fonction constitué d'instructions
  - Affectations
  - Tests
  - Boucles
  - Appels à d'autres fonctions ou procédures

## EXEMPLE DE FONCTION

```
int factorielle (int n)
{
    int i;
    int f;

    i= 1;
    f= 1;

    while (i <= n)
    {
        f= i * f;
        i= i + 1;
    }

    return f;
}
```

- La fonction retourne un entier (valeur de f)
- Elle prend un paramètre n (entier dont on veut calculer la factorielle)
- Elle nécessite la déclaration de deux variables locales
  - i compteur de boucle
  - f permettant de stocker le résultat

## APPEL DE FONCTION

- Fournir des valeurs à la fonction
  - = lui attribuer des paramètres effectifs
- Récupérer les résultats après exécution
  - En affectant le résultat à une variable de même type :  
**resultat = factorielle (5);**
  - En affichant directement le résultat obtenu  
**cout << factorielle (5);**

**!** typeRetour et “resultat” doivent être de **même type**

# ANATOMIE D'UNE PROCEDURE (EN C)

- Syntaxe : **void** NomProc (type paramètre , ...)
  - Ne retourne rien : **void**
  - Ensemble de paramètres formels
    - avec leur type, séparés par des “,”
- Déclaration de variables locales
  - Résultats intermédiaires
  - Compteurs de boucles
- Corps de la procédure constitué d'instructions
  - Affectations
  - Tests
  - Boucles
  - Appels à d'autres fonctions ou procédures

## EXEMPLE DE PROCÉDURE

```
void affiche_mention(int note)
{
    if(note > 10)
    {
        cout << "admis" << endl;
    }
    else
    {
        cout << "recalé" << endl;
    }
}
```

- Ne renvoie rien (void)
- Prend un paramètre note qui sera utilisé dans le corps de la procédure
- Pas de variable locale pour mémoriser le résultat puisque pas de **résultat à calculer** ou à **renvoyer..**



# APPEL DE PROCÉDURE

- Fournir des valeurs aux paramètres de la procédure
  - = paramètres effectifs
- **Attention** : cette fois-ci pas d'affectation de résultat dans une variable car ne renvoie rien !!!
- Juste un appel à la procédure
  - Exemple : `affiche_mention(14)` : l'affichage du résultat se fera directement dans la procédure
  - Interdit d'écrire : `variable=affiche_mention(14)` !!!!

# ANATOMIE DE MAIN

- C'est une **fonction** « normale » avec un rôle particulier
  - Celle qui va s'exécuter en premier
  - Appelle les autres sous-programmes
  - Fonction ➔ retourne un résultat qui permet de savoir si l'exécution s'est terminée correctement
  - Plusieurs possibilités d'écriture de la valeur de retour

```
int  
main(void)  
{  
    ...  
    return 0;  
}
```

```
int main(void)  
{  
    ...  
    return  
        EXIT_SUCCESS;  
}
```

```
int main(void)  
{  
    ...  
    exit  
        (EXIT_SUCCESS) 18  
}
```

## PARAMÈTRE FORMEL / EFFECTIF

- **Paramètre formel** : variable utilisée dans le corps du sous-programme qui reçoit une valeur de l'extérieur
  - Ils font partie de la description de la fonction
- **Paramètre effectif** : la variable (ou valeur) fournie lors de l'appel du sous-programme
  - Valeurs fournies pour utiliser la fonction et valeurs renvoyées
- Copie de la valeur du paramètre effectif vers le paramètre formel correspondant lors de l'appel
- Paramètres formel et effectif ont des noms **différents**

## EXEMPLE AVEC DES FONCTIONS

```
bool est_multiple(int a, int b)
{ ... }
```

Fonction qui renvoie un booléen et prends deux paramètres de type entier (séparés par des ",")

```
bool est_premier(int v)
{
    int i;
    i= 2;
    while(i <= v-1)
    {
        if(est_multiple(v, i))
            return false;
        i=i+1;
    }
    return true;
}
```

On effectue directement le test  
⇔ if (est\_multiple(v,i))==true

On peut appeler un sous-programme à l'intérieur d'un autre sous programme

```
int main(void)
{
    cout<<est_premier(10);
}
```

Fonction est premier donc on affiche le résultat retourné

# Exemple avec des procédures

```
void affiche_mention(int note)
{
    if(note > 10)
    {
        cout << "admis" << endl;
    }
    else
    {
        cout << "essaye encore" << endl;
    }
}

int main (void)
{
    int n;

    cout<<"donnez votre note";
    cin>> n;
    mention (n);

    return (EXIT_SUCCESS);
}
```

# APPEL FONCTION / PROCÉDURE

- Une fonction renvoie une valeur que l'on peut utiliser :
  - afficher,
  - affecter dans une variable,
  - comparer à une autre valeur.
- Une procédure ne renvoie pas de valeur :
  - on ne peut ni afficher, ni affecter, ni comparer.

## EXEMPLES : LESQUELS FONCTIONNENT ?

- `cout << factorielle(7);`
- `if(factorielle(factorielle(3)) < 1000)`  
    `cout << "oui";`  
    `else`  
        `cout << "non";`
- `cout << mention(12);`
- `mention(factorielle(3));`
- `cout<< mention (factorielle (3));`

## RAPPELS : VARIABLES LOCALES

- Elles se déclarent au début du sous-programme
  - juste après “{”
- Elles conservent les résultats intermédiaires nécessaires à l'exécution du sous-programme, les compteurs de boucles...
- Elles n'existent pas en dehors de la fonction
  - On parle de **portée des variables**
  - La variable “a” de la fonction `toto` n'est pas la même que la variable “a” de la fonction `titi`
- Les variables locales sont **DETRUITES** à la sortie du sous-programme



## EN RÉSUMÉ : UNE FONCTION

- Doit avoir un nom clair et compréhensible, évocateur de ce qu'elle calcule
- Doit préciser comment l'utiliser (description des paramètres formels)
- Doit préciser la manière dont elle renvoie ses résultats :
  - avec renvoyer / return
  - avec un ou plusieurs paramètres

## EN RÉSUMÉ : UNE PROCÉDURE

- Doit avoir un nom clair et compréhensible toujours évocateur de ce qu'elle fait
- Doit préciser comment l'utiliser (description des paramètres formels)

- Doit être de la forme :

```
void Nom_Procedure(type paramètre1, type  
                    paramètre2, type paramètre3, ...)  
{  
    ...  
}
```

## ET EN ALGORITHMIQUE ?

- Comment écrire des fonctions et des procédures en algorithmique ?
- Informations supplémentaires
  - **Préconditions** : conditions d'utilisation du sous-programme vérifiées avant l'exécution
  - **Données** : variables ayant une valeur en entrant dans le sous-programme
  - **Résultats** : résultat que doit retourner le sous-programme
  - **Description** : donne une brève description de ce que doit faire le sous-programme

# LA FONCTION EN ALGORITHMIQUE

**Fonction** nom\_fonction (liste des paramètres) : type retourné

Préconditions :

Données :

Résultats :

Description :

Variables locales :

**Début**

instruction(s)

**retourner** valeur (ou **renvoyer**)

**Fin** nom\_fonction

# LA FONCTION EN ALGORITHMIQUE

**Fonction** minimum-fct( x : entier, y : entier) : entier

Précondition : aucune

Données : x et y

Résultat : minimum de x et y

Description : donne le minimum entre deux valeurs passées en paramètre

Variable locale :

m :entier

**Début**

si x < y alors

m ← x

sinon

m ← y

**FinSi**

renvoyer m

**Fin** minimum-fct

## LA FONCTION : APPEL

- Une fonction peut être utilisée
  - Dans le programme principal
  - Dans une autre fonction
  - À l'intérieure d'elle-même avec d'autres paramètres (récursivité)
    - notion abordée au semestre prochain en LIFAP2
- ➔ Programme appelant
- Paramètres réels ou effectifs
- Syntaxe : `variable <- nom_fonction (paramètres)`
- Exemple : `mini <- minimum-fct(4,8)`

# LA PROCÉDURE

- Groupe d'opérations ou suite d'instructions réalisant une certaine tâche
- Définie par un en-tête
  - Nom
  - Liste des paramètres ou arguments (0, 1 ou plusieurs) avec leur type (paramètres formels)
- Pas d'instruction **retourner**

# LA PROCÉDURE EN ALGORITHMIQUE

**Procédure** nom\_procédure (liste des paramètres)

Préconditions :

Données :

Description :

Variables locales :

**Début**

instruction(s)

**Fin** nom\_procédure



# LA PROCÉDURE EN ALGORITHMIQUE

**Définition** minimum-proc(x : entier, y : entier)

Préconditions :

Données : x, y

Description : affiche le minimum entre deux valeurs passées en paramètre

Variables locales : m : entier

**Début**

**si**  $x < y$  **alors**

$m \leftarrow x$

**sinon**

$m \leftarrow y$

**FinSi**

**afficher**(m)

**Fin** minimum-proc

# FONCTION / PROCÉDURE

- Fonction et procédure sont des **sous-programmes**
- Une fonction retourne une valeur mais ne modifie pas l'environnement
- Une procédure ne renvoie aucune valeur mais modifie l'environnement (ex : afficher)
- Une procédure est une fonction ne renvoyant rien

# PLAN

- Anatomie d'un programme en C
- Les sous-programmes en C / en algorithmique
  - Fonctions
  - Procédures
- L'indentation d'un programme C

# INDENTATION D'UN PROGRAMME

- Action qui permet d'ajouter des tabulations
- Après une "{" on décale les instructions sur la droite
- Rend le code source plus clair et plus lisible
- Fait quasi-automatiquement sous codeblocks /devcpp / ...

# INDENTATION D'UN PROGRAMME

```
int produit_iter(int x, int y)
{
    int prod,i;
    prod=0;
    for(i=1;i<=y;i++)
    {
        prod+=x;
    }
    return prod;
}
```

```
int produit_iter(int x, int y)
{
    int prod,i;
    prod=0;
    for(i=1;i<=y;i++)
    {
        prod+=x;
    }
    return prod;
}
```

# PLAN

- Anatomie d'un programme en C
- Les sous-programmes en C / en algorithmique
  - Fonctions
  - Procédures
- L'indentation d'un programme C