

Introduction au C

Fonctions et décomposition d'un programme

University of Geneva
www.miralab.ch

Le langage C est *modulaire*

Regroupement de fonctionnalités par module

Simplification par découpage en sous problèmes

Séparation des tâches à accomplir

Programme intégrant ces différents modules

Comment ?

En utilisant des appels de sous routines

Chaque routine contient une liste d'instruction

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

2 types de sous routine

6

Fonction

Retourne une valeur au programme appelant
Par l'instruction *return()*

Procédure

Identique aux fonctions mais ne retourne pas de valeur
Utilise le type *void* lors de la déclaration

Déclaration, définition, références

7

Déclaration

= indication du **prototype** de la fonction (définition du type et des paramètres de la fonction)

1 seule par module

Souvent placé dans un fichier **header** (.h)

Définition

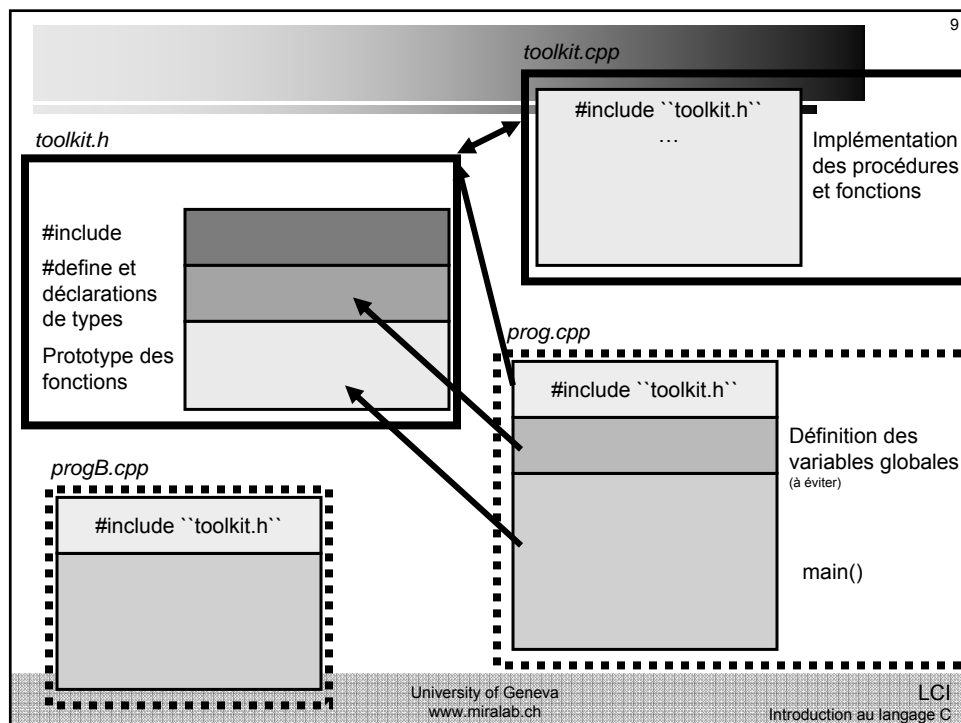
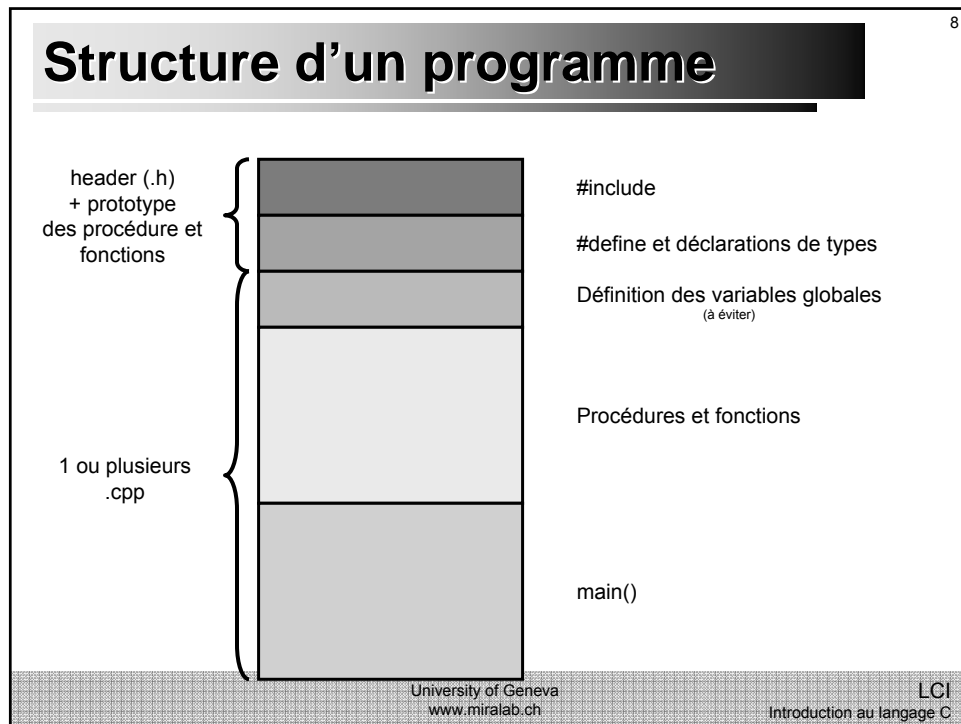
= indique l'**implémentation** de la fonction

1 seule par fonction

Référence(s)

= appel de la fonction depuis un bloc de contrôle

Autant que nécessaire



#include `` ou < ?

10

Les #include peuvent utiliser 2 types de notations

``fileheader.h`` : définit explicitement le chemin où trouver le fichier *fileheader.h*
où ``C:\lib\tookit.h``

<fileheader.h> : recherche le fichier dans les répertoires systèmes (ou par défaut suivant l'environnement de programmation)

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Déclaration de sous-routines

11

(fonctions ou procédures)

Similaire au cas des variables

Doit se faire avant la référence de la sous-routine

En utilisant des types existants *int*, *float*, *void*, ... ou personnel

type nomProcedure ();

Exemple

Déclaration d'une procédure

void ecrire_message();

Déclaration d'une fonction retournant un booléen

bool ecrire_message();

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

12

Décomposition d'un programme

Exemple résolution d'une série d'équations
du 2nd degré

$$ax^2 + bx + c = 0$$

On doit lire les coefficients a, b et c

Résoudre l'équation et afficher la solution

La condition d'arrêt : a == 0

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

13

Lire les 3 coefficients

Tant qu'il y a une équation à traiter (a ≠ 0)

 Calculer la ou les solutions

 calculer le discriminant $\Delta = b^2 - 4.a.c$

 en déduire s'il y a 0, 1 ou 2 solutions réelles

 si $\Delta < 0$: pas de solution

 si $\Delta = 0$: 1 solution

$$x_1 = -\frac{b}{2.a}$$

 sinon 2 solutions

$$x_1 = \frac{-b - \sqrt{\Delta}}{2.a} \quad x_2 = \frac{-b + \sqrt{\Delta}}{2.a} \quad x_1 = -\frac{b}{2.a}$$

 Lire les 3 coefficients

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Fonction main()

14

```
void main() {
    lire_coefs();
    while (y_a_equation()) {
        calculer_sol();
        lire_coefs();
    }
}
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

lire_coefs()	→	cin>>a>>b>>c
y_a_equation()	→	a != 0
calculer_sol()	→	<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> re-décomposition </div> <div> { delta = calculer_delta(); if (delta < 0) ecrire_message(); else if (delta == 0) solution_unique(); else solution_double(); } </div> </div>

15

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

16

```

calculer_delta()    → = b*b - 4*a*c;
ecrire_message()   → cout << ``Pas de
                    solutions``<<endl;
solution_unique()   → x = (-1*b) / (2.*a)
                    cout<<``la solution
                    est `` << x << endl;
solution_double()   → x1 = (-b - sqrt....)

```

17

```

#include <iostream>
#include <math.h>
using namespace std;

int a,b,c;

void lire_coefs() {
    cin >> a >> b >> c;
}

bool y_a_equation() {
    return (a != 0);
}

int calculer_delta() {
    return (b*b - 4*a*c);
}

void ecrire_message() {
    cout << "Pas de solution !" << endl;
}

void solution_unique() {
    float x = (-1 * b) / (2.0 * a);
    cout << "La solution est " << x << endl;
}

void solution_double() {
    float x1 = (-1*b - sqrt((double)calculer_delta())) / (2.0*a);
    float x2 = (-1*b + sqrt((double)calculer_delta())) / (2.0*a);
    cout << "Les solutions sont x1 = " << x1 << " et x2 = " << x2
    << endl;
}

void calculer_sol() {
    int delta = calculer_delta();
    if (delta < 0) {
        ecrire_message();
    }
    else {
        if (delta == 0) {
            solution_unique();
        }
        else {
            solution_double();
        }
    }
}

void main() {
    lire_coefs();
    while (y_a_equation()) {
        calculer_sol();
        lire_coefs();
    }
}

```

18

Référence ou appel de routine

Se fait à l'intérieur d'un bloc de contrôle

Valide pour une fonction déclarée ou définie précédemment

Peut être récursif

Appel de la fonction dans le corps de celle-ci

Le résultat d'une fonction peut être utilisé

Pour l'affectation d'une variable

```
int a = calcul();
```

Dans une expression

```
int a = 3 * calcul();
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

19

Appels de sous-routines

```
void calculer_sol() {
    int delta = calculer_delta();           // doit avoir été déclaré
    if (delta < 0) {
        cout << "Pas de solutions" << endl;
    }
    else {
        if (delta == 0) {
            solution_unique();           // doit avoir été déclaré
        }
        else {
            solution_double();          // doit avoir été déclaré
        }
    }
}
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

20

Définition d'une routine

Rappel

Fournit l'implémentation de la routine

Consiste en

La répétition de la déclaration (identique)

Suivie d'un bloc d'instructions

= corps de la fonction

Contenant les instructions à exécuter

Peut faire office de déclaration

(définition = déclaration + définition)

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

21

Exemple avec déclaration

[...]

// déclaration

```
void lire_coefs();
bool y_a_equation();
int calculer_delta();
void ecrire_message();
void solution_unique();
void solution_double();
void calculer_sol();
```

// définition

```
void lire_coefs() {...}

bool y_a_equation() {...}

int calculer_delta() {...}

void ecrire_message() {...}

void solution_unique() {...}

void solution_double() {...}
```

```
void calculer_sol() {
    int delta = calculer_delta();
    [...]
    if (delta == 0) {
        solution_unique();
    }
    else {
        solution_double();
    }
}
```

// programme principal

```
void main() {
    lire_coefs();
    while (y_a_equation()) {
        calculer_sol();
        lire_coefs();
    }
}
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Exemple avec déclaration + définition

22

```
[...]

// déclaration et définition
void lire_coefs() {[...]}

bool y_a_equation() {[...]}

int calculer_delta() {[...]}

void ecrire_message() {[...]}

void solution_unique() {[...]}

void solution_double() {[...]}

void calculer_sol() {
    int delta = calculer_delta();
    [...]
    if (delta == 0) {
        solution_unique();
    }
    else {
        solution_double();
    }
}

// programme principal
void main() {
    lire_coefs();
    while (y_a_equation()) {
        calculer_sol();
        lire_coefs();
    }
}
```

Exemple avec déclaration + définition ERROR !

23

```
[...]

// déclaration et définition
void lire_coefs() {[...]}

bool y_a_equation() {[...]}

int calculer_delta() {
    [...]
    ecrire_message(); // ERROR !!!
}

void ecrire_message() {[...]}
void solution_unique() {[...]}
void solution_double() {[...]}

void calculer_sol() {
    int delta = calculer_delta();
    [...]
    if (delta == 0) {
        solution_unique();
    }
    else {
        solution_double();
    }
}

// programme principal
void main() {
    lire_coefs();
    while (y_a_equation()) {
        calculer_sol();
        lire_coefs();
    }
}
```

Instruction *return()*

24

Rappel

Retourne la valeur de la fonction au programme appelant

```
return (expression);
```

C'est la **dernière** instruction exécutée dans la sous-routine !!

return; peut être utilisée dans une procédure pour stopper l'exécution de celle-ci

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Retour de *return()*

25

```
bool y_a_equation() {
    bool resultat = (a != 0);
    return resultat;
}
```

```
int calculer_delta() {
    int res = (-1.0*b) / (2.0*a);
    return res;
}
```

```
bool y_a_equation() {
    return (a != 0);
}
```

```
int calculer_delta() {
    return ((-1.0*b) / (2.0*a));
}
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Dernière instruction exécutée

26

```

bool y_a_equation {
    return (a != 0);
    a = 0;  ← NE SERA JAMAIS EXECUTEE !!
}

bool y_a_equation() {
    if (a != 0) {
        return (true); ← si vrai, l'exécution s'arrête ici
    }

    return (false); ← sinon ici
} ← les instructions ici ne seront jamais exécutées

```

27

```

void solutions()
float x1, x2;

if (calculer_delta() == 0) {
    float x1 = (-1 * b) / (2.0 * a);
    cout << "La solution est " << x1 << endl;
    return; // stoppe l'exécution
}

x1 = (-1*b - sqrt((double)calculer_delta())) / (2.0*a);
x2 = (-1*b + sqrt((double)calculer_delta())) / (2.0*a);
cout << "Les solutions sont x1 = " << x1 << " et x2 = " << x2 << endl;
}

void calculer_sol() {
    int delta = calculer_delta();
    if (delta < 0) {
        ecrire_message();
    }
    else {
        solutions();
    }
}

```

} solution_unique et
solution_double dans la même
routine

28

Valeur retournée par *return()*

Si le type variable/expression donnée à *return()* est \neq du type défini dans la déclaration de la fonction

Soit erreur de compilation

Soit un type-cast implicite

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

29

Déclaration des variables

Nous avons vu que les variables étaient « valide » à l'intérieur des blocs de contrôle où elles étaient définies

→ **variable locale**

Il est possible de définir des variables entre les corps des procédures et les fonctions

→ **variable globale**

Les variables sont accessibles dans tout le programme après la déclaration globale

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

30

```
#include <math.h>
using namespace std;

int a,b,c; ← variables globales

void lire_coefs(){
    cin >> a >> b >> c;
}

bool y_a_equation(){
    return (a != 0);
}

int calculer_delta(){
    return (b*b - 4*a*c);
}

void ecrire_message(){
    cout << "Pas de solution !" << endl;
}

void solution_unique(){
    float x = (-1*b) / (2.0*a);
    cout << "La solution est " << x << endl;
}

void solution_double(){
    float x1 = (-1*b - sqrt((double)calculer_delta()) / (2.0*a);
    float x2 = (-1*b + sqrt((double)calculer_delta()) / (2.0*a);
    cout << "Les solutions sont x1 = " << x1 << " et x2 = " << x2
    << endl;
}

void calculer_sol(){
    int delta = calculer_delta();
    if (delta < 0){
        ecrire_message();
    }
    else {
        if (delta == 0){
            solution_unique();
        }
        else {
            solution_double();
        }
    }
}

void main(){
    lire_coefs();
    while (y_a_equation()) {
        calculer_sol();
        lire_coefs();
    }
}
```

variable locale

variable locale

variables locales

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Retour de valeur par *main()*

Succès 0 ⇔ zéro erreur

Erreur : valeur != 0 ⇔ code d'erreur

Exemple :

```
int main() {
    return (lire_fichier());           // retourne 1 si erreur
    return (calculer_data());          // retourne 2 si erreur
    return 0;                          // le programme c'est bien terminé
}
```

Structure de données

University of Geneva
www.miralab.ch

struct

33

Les structures permettent de **regrouper** des objets (des variables) au sein d'une entité repérée par un seul nom de variable.

Lors de la déclaration de la structure, on indique les champs de la structure, c'est-à-dire le **type** et le **nom** des variables qui la composent:

```
struct Nom_structure {
    type_1    Nom_var1;
    type_2    Nom_var2;
```

```
    ...
};
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Exemple

34

```
struct MaStructure {
    int Age;
    char Sexe;
    char Nom[12];
    float MoyenneScolaire;
    struct AutreStructure StructBis;

    /* en considerant que la structure AutreStructure est definie */
};
```

Définition d'une variable structurée

35

La définition d'une variable structurée est une opération qui consiste à créer une variable ayant comme type celui d'une structure que l'on a précédemment déclaré, c'est-à-dire la nommer et lui réserver un emplacement en mémoire.

La définition d'une variable structurée se fait comme suit:

```
struct Nom_Structure Nom_Variable_Structuree;
```

Nom_Structure représente le nom d'une structure que l'on aura préalablement déclarée.

Nom_Variable_Structuree est le nom que l'on donne à la variable structurée.

Exemple

36

```
struct Personne{
    int Age;
    char Sexe;
};
```

On peut définir plusieurs variables structurées:

```
struct Personne Pierre, Paul, Jacques;
struct Personne toto = { 5, 'm'};
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Déclaration d'un typedef de struct

37

But : définir un nouveau type correspondant à une structure

```
typedef struct { ... } Nom_type;
```

Définition, exemple:

Nom_type a, b;

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Accès aux champs d'une variable structurée

38

Chaque variable de type structure possède des champs repérés avec des noms uniques. Toutefois le nom des champs ne suffit pas pour y accéder étant donné qu'ils n'ont de contexte qu'au sein de la variable structurée...

Pour accéder aux champs d'une structure on utilise l'opérateur de champ (un simple point) placé entre le nom de la variable structurée que l'on a défini et le nom du champ:

Nom_Variable.Nom_Champ;

Ainsi, pour affecter des valeurs à la variable *Pierre* (variable de type *struct Personne* définie précédemment), on pourra écrire:

```
Pierre.Age = 18;
Pierre.Sexe = 'M';
```

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Résumé

39

Structure d'un
programme

Procédures et fonctions

Structures de données

University of Geneva
www.miralab.ch

LCI
Introduction au langage C

Exercices

University of Geneva
www.miralab.ch

Exercice 3 : while

41

Faire un programme qui affiche une somme ajoutant 2 à chaque fois, jusqu'à un résultat maximum donnée par l'utilisateur (la somme sera calculée par additions successives)

Exemple :

pour 7 → affichage de 0, 2, 4, 6

pour 4 → affichage de 0, 2, 4



University of Geneva
www.miralab.ch

LCI
Introduction au langage C

42

Exercice 4 : struct

1. Définir un type de structure permettant de représenter un point de l'espace (float) et un index (int). + qqs exemples d'initialisation et d'accès.
2. Définir une structure et un type de structure représentant les informations suivantes :
 - Le nom (string)
 - Un identifiant (int)
 - L'âge (float)
 + qqs exemples d'initialisation et d'accès.

43

Exercice 5

Tout pixel d'une image est caractérisé par un couple de coordonnées (x,y). On peut donc calculer des distances entre pixels. Les distances les plus courantes sont (pour deux pixels P(xp,yp) et Q(xq,yq)):

- distance de Manathan : $d1(P,Q) = |xp - xq| + |yp - yq|$
- distance euclidienne : $d2(P,Q) = [(xp - xq)^2 + (yp - yq)^2]^{1/2}$
- distance de l'échiquier : $dinf(P,Q) = \text{Max}(|xp - xq|, |yp - yq|)$

Ces distances sont reliées par la propriété :

$$dinf(P,Q) \leq d2(P,Q) \leq d1(P,Q)$$

Écrire un programme définissant les structures nécessaires définir les points et les fonctions permettant de calculer les distances de **Manathan** et **euclidienne** (echiquier en option ☺) + un petit programme *main()* utilisant ces fonctions.

PS: <math.h> fournit 2 fonctions intéressantes : abs() et sqrt() ☺