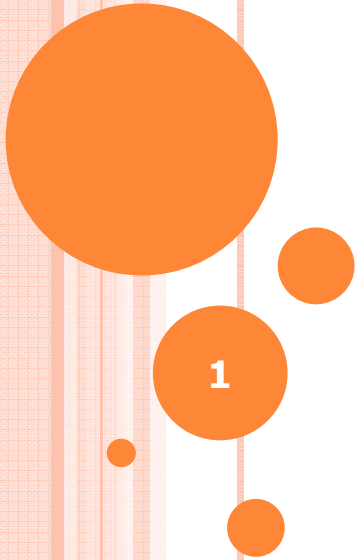
Licence STS

Université Claude Bernard Lyon I

LIF1 : Algorithmique et Programmation Impérative, Initiation



1

COURS 7 : Structures et Fichiers

Objectifs de la séance

- Deux notions abordées dans ce cours
  - Celle de structure
  - Celle de fichier
- Comprendre l'intérêt des structure et apprendre comment les manipuler
- Savoir stocker le résultat d'une exécution dans un fichier et la récupérer ultérieurement

Plan

- Les structures
  - Définition
  - Intérêt
  - Syntaxe
  - Manipulation
- Les fichiers
  - Stockage et arborescence
  - L'utilisation des fichiers en C

Structure : Définition et vocabulaire

- Agrégat d'informations associées à une entité
- **Type complexe** construit à l'aide de type simples ou d'autres types complexes
- Chacune des informations contenue dans une structure s'appelle un **champ**
- Une variable de type structure est aussi appelée un **enregistrement**
  - Analogie avec les bases de données

DÉCLARATION

- En Algorithmique

```
Structure Nom_Structure
    champ1 : type
    champ2 : type
    …
Fin structure
```

- En C

```
Struct Nom_Structure
{
    type champ1;
    type champ2;
    …
};
```

EXEMPLE : EN ALGORITHMIQUE

Structure IdentiteEtudiant

prenom : tableau[64] de caractères

nom : tableau[64] de caractères

Fin structure

Structure Etudiant

identite : IdentiteEtudiant

note : tableau[10] de réels

numero : entier

Fin Structure

- identite, note et numéro sont les champs de la structure Etudiant.
- Chacun des champs est
  - soit de type simple
    - numero
  - soit de type complexe
    - identite

EXEMPLE : EN C

```
struct IdentiteEtudiant
{
    char prenom[64];
    char nom[64];
};

struct Etudiant
{
    struct IdentiteEtudiant identite ;
    float note[10];
    int numero;
};
```

- Mot clé : **Struct**
- En C on termine la définition de la structure par un ";" après l'accolade
- Tous les champs se terminent par un ";"

UTILISATION DE CONSTANTES EN C

Possibilité de définir des constantes et de fixer leurs valeurs

```
const int longueurNom = 64 ;
const int nombreDeNotes = 10 ;

struct etudiant
{
  char nom[longueurNom] ;
  float note[nombreDeNotes] ;
} ;
```

Déclaration d'une variable de type structure

- Nécessaire avant d'utiliser la structure
- De même qu'on écrit "int i" avant d'utiliser "i", on déclare une variable de type structure Nom_Structure avant de l'utiliser
- En algorithmique :
  - Etu : etudiant
- En C :
  - struct etudiant toto ;

ACCÈS À UN CHAMP

- Pour remplir une variable de type structure, il faut procéder champ par champ (pas de remplissage global) car les types des champs sont différents
- Exemple en C:

```
struct etudiant e;          déclaration de e, variable de type etudiant
cin>> e.numero;             lit le numero de l'étudiant e
cout << e.note[i];          affiche la ieme note de l'étudiant e
cin>>e.identite.nom;        avec un champ de type structure
```

UTILISATION DES STRUCTURES

- Une fonction peut retourner une structure
- Une structure peut faire l'objet d'une affectation (avec une variable de même type !)
  - Etudiant e1,e2;
  - e2=e1;
- Les tableaux de structures sont possibles

```
struct etudiant classe[20] ;          /*tableau de 20 etudiants*/
struct etudiant y ;
y = creerEtudiant() ;
classe[0] = y ;
```

UTILISATION

- Exemple de création d'une fiche étudiant :

```
struct etudiant creerEtudiant(void)
{
  struct etudiant e ;
  int i ;

  cout << endl << "entrer le nom :" << endl ;
  cin >> e.identite.nom ;
  cout << endl << "entrer le prénom :" << endl ;
  cin >> e.identite.prenom ;
  cout << endl << "entrer le numero de l etudiant :" << endl ;
  cin >> e.numero ;
  for (i=0; i < nombreDeNotes; i++) {
    cout << endl << "entrer la " << i << "ème note :" << endl;
    cin >> e.note[i] ;
  }
  return e ;
}
```

TRANSFORMATION

- Il est possible de transformer la fonction précédente en procédure
- L'entête devient alors :
  - void creerEtudiant(struct etudiant & e)
- Un structure peut être passée en donnée – résultat
- Une structure peut être retournée par une fonction

Résolution d'un polynôme

- Informations à connaître ou à évaluer
  - Les coefficients du polynôme : a, b, c donnés par l'utilisateur
  - Le discriminant delta calculé en fonction de a, b, et c
  - Le nombre de racine (en fonction de delta 0 1 ou 2 racines)
  - Les racines réelles dans la mesure où elles existent
- Soit on utilise 7 variables différentes
- Soit on met toutes ces informations dans une structure

La structure "polynôme"

- En algo

```
Structure polynome
    a,b,c : réels
    delta : réel
    nb_racines : entiers
    rac1,rac2 : réels
Fin Structure
```

- En C

```
Struct polynome
{
    float a,b,c;
    float delta;
    int nb_racines;
    double rac1,rac2;
};
```

Les fonctions associées

- Plusieurs fonctions à écrire
  - Saisie des coefficients
  - Calcul de delta
  - Calcul du résultat
  - Affichage du résultat
- 1 paramètre unique à passer : une variable de type "polynome"
- Certains champs seront remplis / calculés / affichés
- Structure passée en donnée / résultat ou retournée en résultat

LA FONCTION DE SAISIE

- On demande à l'utilisateur de donner les 3 coefficients a, b et c

```
struct polynome saisie_coefficients(void)
{
        struct polynome p;
        cout << "donnez a, b et c" ;
        cin>>p.a>>p.b>>p.c;
        return p;
}
```

- On retourne la structure car elle est vide au départ

LA FONCTION DE CALCUL DE DELTA

- On calcule delta en fonction de a, b et c

```
void calcul_delta(struct polynome & p)
{
        p.delta = (p.b*p.b) - 4*p.a*p.c;
}
```

- p est passé en donnée/resultat car on va utiliser 3 champs pour en remplir un.

LA FONCTION DE CALCUL DES RACINES

- On calcule les racines en fonction de delta, a, b et c

```
void calcul_racines(struct polynome & p)
{
        if (p.delta == 0)
        {
                p.rac1=-p.b / (2*p.a);
                p.rac2 =-p.b / (2*p.a);
                p.nb_racines=1;
        }
        else if (p.delta > 0)
                {
                        p.rac1=(-p.b + sqrt(p.delta))/ (2*p.a);
                        p.rac2 =(-p.b – sqrt(p.delta))/ (2*p.a);
                        p.nb_racines=2;
                }
                else p.nb_racines=0
}
```

Les Fichiers

- Les informations lues ou calculées par un programme C disparaissent à fin de son exécution (lorsqu'on ferme la fenêtre)
- Il est souvent utile et nécessaire de conserver des informations pour un usage ultérieur : exemple : remplissage d'un tableau assez fastidieux
- Les fichiers sont là pour çà !

Les fichiers : où sont ils ?

- Les fichiers sont conservés sur divers supports :
  - 1 seule écriture : Mémoire morte (read-only memory) :
    - ensuite que des lectures possibles (CD-ROM, DVD-ROM)
  - Ecriture difficile, lecture facile :
    - CD-RW, EPROM,...
  - Lecture/Ecriture : disques durs, disquettes, clés USB, ...
- Sur la machine ou sur le réseau (Locaux ou Distants)

COMMENT SONT-ILS RANGÉS ?

- Cela dépend du système d'exploitation
- Sous windows :
  - Le lecteur identifié par une lettre suivie de ":"
    - A: C: W:
  - Les dossiers :
    - Qui contiennent d'autres dossiers ou des fichiers
    - Qui servent à organiser le rangement selon une hierarchie = l'arborescence
  - Dossier spécial qui contient tous les autres = la racine du lecteur

COMMENT SONT-ILS NOMMÉS ?

- Le nom est le chemin d'accès : c'est le nom correspondant au chemin absolu (depuis la racine)
  - W:\LIF1\Cours\cm1.pdf
  - W:\LIF1\TP\TP1\hello.cpp
- Le nom court est la dernière partie du chemin
  - cm1.pdf
- Utilisable si on est déjà dans le répertoire (dossier courant ou de travail) = chemin relatif

COMMENT LES MANIPULER ?

- Par les outils fournis par le système
  - Copie /déplacement
  - Effacement / destruction
  - Ouverture
  - Etc..
- Par les applications disponibles
- Voir le cours de PCII
- Par programme codeblocks : c'est la suite...

2 TYPES DE STOCKAGES

- Les fichiers binaires:
  - Humainement illisibles
  - Son, musique
  - Format spécifique à l'application
- Les fichiers textes :
  - Lisible par des éditeurs de texte
  - Les données sont sous formes de codes de caractères (ASCII, UTF,...)
  - Ce sont eux que nous présenterons

Les fichiers en C

- Désignés par une variable typée
- Selon l'usage fait par le programme :
  - Lecture : ifstream
  - Ecriture : ofstream
  - Lecture / Ecriture : fstream
- Deux variables prédéfinies :
  - cin est du type ifstream
  - cout est du type ofstream

Manipulation d'un fichier

- Avant de pouvoir lire ou écrire, il faut ouvrir le fichier au préalable
- Les variables de type ifstream et ofstream sont empruntés au C++, ce qui explique le formalisme particulier
- Toujours les passer en tant que donnée/résultat (&) dans les fonctions ou procédures qui les utilisent

OUVERTURE D'UN FICHIER EN LECTURE

- Il faut lui donner un nom (passé en paramètre éventuellement)
- On l'ouvre en ifstream
- Et vérifier que tout s'est bien passé ➔ renvoyer un code d'erreur

```
bool ouvertureFichierLecture( ifstream& fichier,
                              char nom[1024] )
{
  fichier.open(nom) ;
  return fichier.good() ;
}
```

OUVERTURE D'UN FICHIER EN ÉCRITURE

- Toujours passer son nom en paramtre
- Ici on le déclare en ofstream puisqu'on veut écrire dedans
- on vérifie que tout s'est bien passé

```
bool ouvertureFichierEcriture( ofstream& fichier, char nom[1024] )
 {
  fichier.open(nom) ;
  return fichier.good() ;
}
```

FERMETURE DE FICHIER

- Identique qu'il ait été ouvert en lecture ou en écriture
- Fichier ouvert en lecture

```
void fermetureFichierLecture( ifstream& fichier )
{
  fichier.close() ;
}
```

- Fichier ouvert en écriture

```
void fermetureFichierEcriture( ofstream& fichier )
{
  fichier.close() ;
}
```

Des opérations spéciales

- open(char nomDuFichier[1024]) :
  - Ouvre le fichier dont le nom est donnée en paramètre
  - Si le fichier est de type ofstream et qu'il n'existe pas, le fichier est crée (autrement, c'est une erreur)
  - Préalable à toutes autres opérations
  - Initialise la variable associée
- close(void)
  - Ferme le fichier, assure que toutes les données sont écrites

Opérations spéciales

- >> pour lire :
  - Convertit la chaine de caractères lue en se basant sur le type de la variable lue (si possible)
- << pour écrire :
  - Convertit la valeur donnée en chaine de caractères (si possible)
  - Conversions possibles pour tous les types de bases et les chaines de caractères

UN PETIT EXEMPLE

- Ecriture de la structure étudiant, en séparant chaque champ par un espace

```
void ecrireEtudiant(ofstream& fichier, struct etudiant &e)
{
  int i ;
  fichier << e.nom << ' ' ;
  for (i = 0 ; i < nombreDeNotes; i++ ) {
    fichier << e.note[i] << ' ' ;
  }
  fichier << endl ;
}
```

EXEMPLE

- Relire la ligne dans une structure, en supposant que les champs sont séparés par un espace (éliminé lors de la lecture)

```
bool lireEtudiant( ifstream& fichier, struct etudiant &e )
{
  int i ;
  fichier >> e.nom ;
  if (fichier.eof()) {
    return false ;
  }
  for (i = 0 ; i < nombreDeNotes; i++ ) {
    fichier >> e.note[i] ;
  }
  return true ;
}
```

À QUOI RESSEMBLE LE FICHIER

- Ce que l'on voit en ouvrant par un éditeur de texte

```
Pierre 1 2 3 4 5 6 7 8 9 10
Paul   11 12 13 14 15 16 17 18 19 20
Jacques 15 16 17 18 15 16 17 18 15 16
```

Les exemples complets

- Un programme de saisie :
  - exempleFichier0.cpp
- Un programme pour lire le fichier, remplir le tableau de structure et exploiter les données :
  - exempleFichier1.cpp

Conclusion

- Stuctures :
  - Permettent de ranger dans une même variable toutes les informations relatives à un objet : exemple étudiant
  - Moins de variables, informations mieux organisées
  - Possibilité de faire des tableaux de structures
- Fichiers
  - Permettent de stocker le résultat d'une exécution et de pouvoir la réutiliser
  - Deux types de fichiers en C : ouvert en lecture ou en écriture