Algorithmique et Structures de données

Mouhamadou GAYE

Enseignant-Chercheur
Département d'Informatique
Licence 2 en Ingénierie Informatique

23 mai 2022

Objectifs du cours

- Comprendre les principes de la programmation en langage C
- Décrire et manipuler les structures de données les plus classiques
- Utiliser des structures de haut niveau de manière optimale pour développer des programmes.

Contenu du module

- Chapitre 1 : Eléments de base du Langage C
- Chapitre 2 : Structures linéaires
- Chapitre 3 : Structures arborescentes
- Chapitre 4 : Graphes

Évaluation

- Controle continu : **20%**
- Projet : **20%**
- Examen final : **60%**

Chapitre 1 : Eléments de base du Langage C

Historique du langage C

- Le langage **CPL** (Combined Programming Language) conçu au début des années **1960** par les universités de Cambridge et de Londres
- Le langage **BCPL** (Basic CPL) conçu à Cambridge en **1966** par Martin Richards, une version fortement simplifiée de CPL
- Le langage **B** a été créé par Ken Thompson vers **1970** dans les laboratoires Bell d'AT&T, une simplification de BCPL
- Le langage **C** conçu en **1972** par Dennis Richie et Ken Thompson pour développer un système d'exploitation UNIX

Compilation d'un programme C

Le programme est écrit dans un fichier texte et compilé en quatre phases successives :

1. **Traitement par le préprocesseur** : analyse du fichier source
2. **Compilation** : traduction du fichier source en assembleur
3. **Assemblage** : transformation du code assembleur en code binaire
4. **Edition de liens** : liaison des différents fichiers objets et production du fichier exécutable

NB : 2 et 3 se font généralement dans la foulée.

Structure d'un programme C

- Un programme C est structuré comme suit :

-fichier.c

```
directives au préprocesseur
déclarations de variables globales (optionnel)
définitions de fonctions secondaires (optionnel)
int main(){
    corps du programme principal
    return 0 ;
}
```

Directives au préprocesseur

- **Directive #include**

  Elle permet d'inclure un fichier (en général un fichier d'entête de librairie) dans le fichier source.

  Syntaxe :

  ```
  #include <nom-de-fichier>
  #include "nom-de-fichier"
  ```

- **Directive #define**

  Elle permet de définir des constantes symboliques ou des macros

  Syntaxe :

  ```
  #define NOMCONSTANTE valeur
  #define NOMMACRO(parametres) corpsMacro
  ```

Types de base

- **int**, **short**, **long** : pour les entiers
- **float**, **double** : pour les réels
- **char** : pour les caractères
  L'attribut de représentation **signed** ou **unsigned** peut être utilisé devant un entier pour dire qu'il est signé ou non signé.

  Exemple : unsigned int désigne l'intervalle $[0\,;\,2^{32}[$

Opérateurs

- **Opérateurs arithmétiques** : +, -, *, /, % (modulo)
- **Opérateurs relationnels** : >, >=, <, <=, == (égal), != (différent)
- **pérateurs logiques** : && (et), || (ou), ! (négation)
- **Opérateur d'affectation** : =
- **Opérateur d'incrémentation** : ++
- **Opérateur de décrémentation** : - -
- **Opérateur d'adressage** : &
- **Opérateur de conversion de type ou cast** : (type) objet

Opérateurs

Table – Priorité des opérateurs

Opérateurs	Associativité
()	gauche à droite
++ - - (type) &	droite à gauche
* / %	gauche à droite
+ -	gauche à droite
> >= < <=	gauche à droite
== !=	gauche à droite
&&	gauche à droite
||	gauche à droite
= += -= *= /= %=	droite à gauche

Variable

- **Déclaration de variable**

  Syntaxe :

  type identifiant ;

  NB : Un identifiant est une suite de caractères alphanumériques (lettres non accentuées ou chiffres) et de tiret de soulignement éventuellement qui ne commence pas par un chiffre. Il ne peut pas aussi être un mot du langage (mot réservé).

Entrées/Sorties

- **printf** : une fonction prédéfinie d'affichage à l'écran
  Syntaxe :
  printf("texte et format des expressions", expr1, expr2, ..., exprn) ;
- **scanf** : une fonction prédéfinie de saisie de données et de leur stockage dans des variables
  Syntaxe :
  scanf("format des variables à lire", &var1, &var2, ..., &varn) ;

Table – Formats d'affichage et de lecture

Format	Type
%d	int
%ld	long
%u	unsigned int
%lu	unsigned long
%f	float
%lf	double
%c	caractère
%s	chaine de caractères

Entrées/Sorties

- **putchar** : fonction prédéfinie pour afficher un caractère à l'écran
- **getchar** : fonction prédéfinie pour lire un caractère ; elle retourne l'entier correspondant au caractère lu.

Exemple :

```
char c ;
c = getchar() ;
putchar(c) ;
```

Commentaires

- // Commentaire sur une ligne
- /* Commentaire
  sur
  plusieurs
  lignes */

Structures conditionnelles

- **Instructions conditionnelles simples**

```
if(condition){
    instructions_1
}
else{
    instructions_2
}
```

NB : La partie else est optionnelle.

Structures conditionnelles

- **Instructions conditionnelles imbriquées**

```
if(condition_1){
    instructions_1
}
else if(condition_2){
        instructions_2
    }
        ...
            else{
                instructions_n
            }
```

NB : Le dernier else est toujours optionnel.

Structures conditionnelles

- **Instructions conditionnelles multiples**

```
switch(variable){
    case valeur_1 : {instrcutions_1 break ;}
    case valeur_2 : {instrcutions_2 break ;}
    ...
    case valeur_n : {instrcutions_n break ;}
    default : {instrcutions_par_defaut}
}
```

NB : La variable doit être de type entier ou caractère.

Structures itératives

- **Boucle for**

```
for(initialisation ; test ; pas){
    instrcutions
}
```

Exemple :

```
for(i = 0 ; i < 10 ; i++){
    printf("L'excellence, ma référence\n") ;
}
```

Structures itératives

- **Boucle while**

```
while(condition){
    instrcutions
}
```

Exemple :

```
i = 0;
while(i < 10){
    printf("L'excellence, ma référence\n");
    i++;
}
```

Structures itératives

- **Boucle do while**

```
do{
    instrcutions
}while(condition);
```

Exemple :

```
i = 0;
do{
    printf("L'excellence, ma référence\n");
    i++;
}while(i < 10);
```

Structures

- Une structure est une suite finie d'éléments de types différents qui n'occupent pas nécessairement des zones contiguës en mémoire. Chaque élément de la structure, appelé membre ou champ, est désigné par un identificateur.
- **Déclaration d'une structure :**

```
struct modele{
    type_1 champ_1;
    type_2 champ_2;
    ...
    type_n champ_n;
};
```

- **Déclaration d'une variable de type structure :**

```
struct modele objet;
```

- Ainsi, objet.champ_i désigne le champ numéro i de la variable objet.

Pointeurs

- Un pointeur est un objet dont la valeur est l'adresse d'un autre objet. C'est une adresse typée.
- **Déclaration d'une variable de type pointeur :**

  type * nom_pointeur ;
- L'opérateur unaire d'indirection * permet d'accéder à la valeur de l'objet pointé.

  Exemple :

  ```
  int *p ;
  *p = 2 ;
  ```

Fonctions

- **Déclaration d'une fonction :**

  type nom_fonction (type_1, type_2, ..., type_n) ;

- **Définition d'une fonction :**

  type nom_fonction (type_1 argument_1, type_2 argument_2, ..., type_n arguemnt_n){

  [Déclaration de variables locales]

  instructions

  [return expression ;]

  }

- **Appel d'une fonction :**

  nom_fonction (parametre_1, parametre_2, ..., parametre_n)

Récursivité

- Une fonction est récursive si elle comporte au moins un appel d'elle-même dans son corps.
- Etapes de résolution d'un algorithme récursive
  1. Trouver la condition d'arrêt
  2. Faire une décomposition du problème