REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de M'sila
Faculté des Mathématiques et de l'Informatique
Département d'informatique

جامعة المسيلة
كلية الرياضيات والإعلام الآلي
قسم الإعلام الآلي

Chapitre 1:
Les sous-programmes: Fonctions et Procédures

Algorithmique et structure de données 2

Présenté par: Dr. Benazi Makhlouf
Année universitaire: 2022/2023

Contenu du chapitre 01:

1. Introduction
2. Définitions
3. Les variables locales et les variables globales
4. Le passage des paramètres
5. La récursivité

Introduction

Un programme est un ensemble d'instructions séquentielles pour résoudre un problème spécifique. Afin de trouver la méthode de résolution (algorithme), le problème doit être divisé en différents sous-problèmes dont la solution est moins compliquée. Les problèmes partiels peuvent être résolus à l'aide de sous programmes.

Sous-programmes :

- Est un ensemble d'instructions indépendantes qui ont un nom et qui est appelé pour l'exécution.
- L’appelant est soit le programme principal, soit un autre sous-programme.
- Lorsque le programme, pendant son exécution, atteint l'instruction qui appelle la procédure, le contexte d'exécution devient le contenu du sous-programme, et une fois qu'il a fini l'exécution du sous-programme, il revient à l'exécution de l'instruction qui suit immédiatement l'invocation.
- Les sous-programmes sont appelés : procédures, fonctions, méthodes, routines, subroutines, macro.

Procédure

C'est un sous-programme qui ne renvoie aucune valeur dans son nom, mais il peut renvoyer des résultats via des arguments. Le nom de la procédure peut être utilisé comme instruction complète. par exemple:

- L’affichage sur l'écran des nombres inférieur à une certaine limite
- Afficher les valeurs du tableau à l'écran
- Résoudre une équation quadratique

algorithme	**C**
`SomeProc`	`SomeProc();`
`OtherProc(x)`	`OtherProc(x);`

Fonction

Il s'agit d'un sous-programme qui renvoie obligatoirement un résultat dans son nom, car son nom est considéré comme une variable qui porte une certaine valeur. Par conséquent, l'appel de la fonction peut être utilisé comme variable dans les opérations d'affectation et d'autres expressions. Par exemple :

- Carré d'un nombre
- L'aire d'un rectangle
- La résolution d'une équation du premier ordre
- La somme d'un tableau
- savoir si le nombre est premier ou non

Algorithme	**C**
`Y← SomeFunction(X)*5`	`Y= SomeFunction(X)*5 ;`

Avantages de l'utilisation de sous-programmes

- **Lisibilité** : l'utilisation de sous-programmes organise le programme et le simplifie, ce qui aide à comprendre le code du programme
- **Rapidité** dans la programmation : ne pas répéter plusieurs fois la même séquence d'instructions au sein du programme.
- **Réduire** la taille du programme
- **Facilite** le processus de **maintenance**
- **Réutilisation** : il peut être stocké dans des bibliothèques pour être réutilisé dans d'autres programmes.

Déclarations

Procédure

- **Algorithme**

```
procedure nom_proc (liste paramètres)
    variables locales
Debut
    instructions
Fin.
```

- **En C**

```
void nom_proc (liste paramètres){
    variables locales ;
    instructions ;
}
```

- nom_proc: des identificateurs valides.
- Liste des paramètres (facultatif) : appelés « paramètres formels » un ensemble de variables par lesquelles les données sont transmis et les résultats sont récupérés,
    - séparés par une virgule «,»
    - sont entre deux parenthèses
    - sont de la forme nomParam : type en algorithme et type nomParam en C
    - tel que: (a:entier, b:réel) en **algorithme** et (int a, float b) en **C**
    - Les parenthèses () sont obligatoires même si elles ne contiennent aucun argument en **C** tel que: proc() .
- Déclarations locales (facultatif) : Une liste de variables locales de la forme :
    - var varLoc : type
- Instructions : un ensemble d'instructions de tout type, qui seront exécutées lorsque le sous-programme est appelé.
- toutes les variables déclarées dans la liste des paramètres ou dans la déclaration locale, qui sont appelées variables locales, et les variables déclarées dans le programme principal, appelées variables globales

Déclarations

Fonction

- **Algorithme**

```
fonction nom_fonc (liste paramètres) : type
   variables locales
Debut
   instructions
Fin.
```

- **En C**

```
type nom_fonc (liste paramètres){
   variables locales ;
   instructions ;
}
```

- Result_Type : Lorsque le programme est une fonction, le type de valeur que la fonction retournera au programme qui l'a appelée doit être spécifié, et une valeur doit être attribuée au nom de la fonction. Il s'agit généralement de la dernière instruction de la fonction.
  - En algorithme elle est de la forme nom_fonction← expression où le nom de la fonction agit comme une variable spéciale qui contient la valeur de retour par la fonction.
  - En C, pour indique que la fonction ne renvoi rien cad une procédure soit en laisse Type vide soit en utilise le mot **void** selon la version.
  - le mot **return** est utilisé pour attribuer une valeur au nom de la fonction.
  - l'instruction **return** fait quitter le sous-programme et reprendre le programme qui l'a appelé à l'instruction suivant immédiatement l'invocation. Il peut renvoyer une valeur au programme qui a appelé le sous-programme s'il s'agissait d'une fonction.
  - **Format** : **`return`** `[ <expression> ] ;`
  - **Exemple :**
  - `return 5*x ;`
  - `return ;`

Notes importantes :

- Pour trouver les arguments, nous posons la question qu'est-ce que nous donnons au sous-programme en entrée et qu'est-ce qu'il renvoie en résultat.
- La liste des paramètres de la partie définition du sous-programme doit être identique en nombre et en type à celle utilisée dans l'invocation du sous-programme.
- La première ligne d'une déclaration de fonction ou de procédure c'est à dire : type de fonction, nom de la fonction, type, ordre et nombre d'arguments, sauf leurs noms, est appelée entête ou prototype.
- Les arguments ne sont pas regroupés s'ils sont du même type comme dans (x, y:entier), mais on met (x:entier, y:entier) (int x, int y)
- Tout type de retour autre que void indique que le programme est une fonction et non une procédure.
- void main() ou simplement main() est une procédure, tandis que int main() est une fonction, vous devez donc utiliser return.
- scanf() et printf() sont deux fonctions déclarées dans la bibliothèque stdio

emplacement

- Dans l'algorithme, il se trouve après la déclaration des variables et avant début du programme principal.
- Dans un programme C, elle est déclarée avant la fonction main().
- L'ordre des sous-programmes est important car chaque fonction doit être définie avant de pouvoir être utilisée
- si la fonction f1() appelle la fonction f2(), alors la fonction f2() doit être définie avant la fonction f1().

L’invocation

- Pour appeler et exécuter une procédure, on utilise son nom comme une instruction à part et on affecte des valeurs et/ou des variables aux arguments entre parenthèses, appelés paramètres effectifs. Les parenthèses peuvent être omis en l'absence de tout argument, mais en C, elles sont obligatoires.
- La même chose pour l'appel d'une fonction, où son nom est considéré comme une variable qui porte une certaine valeur, donc l'appel de la fonction peut être utilisé comme une variable dans les opérations d'affectation et d'autres expressions.
- Les paramètres effectifs doivent correspondre en nombres en type et en ordre avec les paramètres formels

Exemples

```
Algorithme test
   var z : réel
```

```
procedure afficheNbrs (n:entire)
   var i:entier
Début
   pour i←1 à n faire
      Ecrire(i)
   finpour
Fin procédure
```

```
fonction sommeNbrs (x:entier, y:entier) : entier
Début
   sommeNbrs ←x+y
Fin fonction
```

```
Début
   afficheNbrs (5)
   z←sommeNbrs (5, 3)
   Ecrire("la somme est ", z)
Fin.
```

Exemples

```
#include <stdio.h>
   float z ;
```

```
void afficheNbrs (int n)
{
   int i ;
   for (i=1 ; i<=n; i++)
      printf("%d\t",i);
}
```

```
int sommeNbrs (int x, int y)
{
   return x+y ;
}
```

```
int main(){
   afficheNbrs (5);
   Z=sommeNbrs (5, 3);
   printf("la somme est %d", z);
return 0 ;
}
```

Variables locales et variables globales

- Une **variable globale** (global variable) est une variable déclarée en dehors du corps de tout sous-programme, et donc utilisable n'importe où dans le programme.
  - il n'est pas nécessaire de la passer comme paramètre pour l'utiliser dans des sous-programmes.
  - elle est créée lors du chargement du programme en mémoire, et elle n'est supprimée qu'à la fin de l'exécution du programme.
- **Une variable locale** est une variable qui ne peut être utilisée que dans le sous-programme ou le bloc où elle est définie,
  - la variable est créée lorsque la fonction est appelée et supprimée lorsque son exécution est terminée.
- Il est recommandé d'utiliser des variables locales et des paramètres plutôt que des variables globales pour éviter les erreurs et garantir l'indépendance des fonctions.

Exemple Algorithme

```
Algorithme glob_loc
   Var glob, b : entier
```

```
Procedure tst
   Var b, loc : entier
Début
   glob←11
   b←22
   loc←33
   Ecrire("dans tst : glob=", glob, "b=", b, "loc=", loc)
End
```

```
Début
   glob←1
   b←2
   Ecrire("avant tst : glob=", glob, "b=", b)
   tst
   Ecrire("après tst : glob=", glob, "b=", b)
end
```

Exemple C

```
#include <stdio.h>
int glob, b ;
```

```
tst(){
    int b, loc ;
    glob=11;
    b=22;
    loc=33;
    printf("dans tst : glob=%d b=%d loc=%d", glob, b, loc);
}
```

```
int main(){
    glob=1;
    b=2;
    printf("avant tst : glob=%d b=%d", glob, b);
    tst();
    printf("après tst : glob=%d b=%d", glob, b);
    return 0 ;
}
```

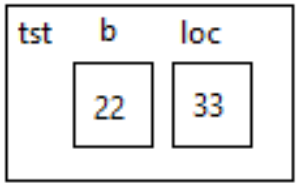
L'écran :

avant tst : glob=1 b=2

dans tst : glob=11 b=22 loc=33

après tst : glob=11 b=2

Explication :

avant d'appeler tst	Durant l'appel de tst	après avoir appelé tst
glob b 1 2	glob b 11 2 	glob b 11 2

4 Passage de paramètres

Les arguments sont les variables par lesquelles les informations peuvent être échangées entre les programmes

Il existe deux façons de passer des paramètres

Passage par valeur :

Dans ce mode, la valeur de la variable d'origine est copiée dans le paramètre (formel), et cette copie est utilisée (une variable locale), et la variable d'origine n'est pas modifiée.

Dans ce mode, on peut utiliser une valeur constante, une expression ou une variable.

Ce mode est utilisé uniquement pour entrer des informations.

Passage par référence, adresse ou variable :

Non seulement la valeur est passée, mais la place de la variable d'origine (adresse) est passée

Variable formelle et variable effective deviennent une seule variable, et toute modification du paramètre dans le sous-programme entraîne la modification de la variable d'origine.

Dans ce mode, il n'est pas possible de passer une valeur constante ou une expression, mais il est nécessaire qu'il s'agisse d'une variable.

Ce mode est utilisé pour recevoir des résultats.

Il est également utilisé pour entrer des informations, surtout s'il s’agit de variable de grande taille, pour éviter de le copier, comme des tableaux et des matrices.

Passage par référence, adresse ou variable :

- Le mot var est utilisé avant de déclarer le nom de l'argument pour indiquer que le passage est un passe par variable ou passage par référence.
- Pour passer les arguments avec l'adresse en C, on utilise les pointeurs que l'on verra dans le troisième chapitre de ce cours, où le nom du paramètre formel est précédé de * lors de la déclaration et lors de l'utilisation, mais lors de l'appel de la fonction, la variable effective est précédée de &

En C++, on utilise le symbole $ (référence) lors de la déclaration uniquement

	Algorithme	C	C++
Déclaration	f(var x:entier)	int f(int *x)	int f(int $x)
Utilisation	x←5	*x=5;	x=5;
Appel	f(a)	f(&a);	f(a);

Exemples

```
Algorithme passage_valeur
   var a, c: réel
```

```
Procedure carre (x: réel, y: réel)
Debut
   y← x*x
fin
```

```
debut
   c←0
   a←3
   ecrire("avant carre c=", c)
   carre(a ,c) // on peut utiliser carre(3,c)
   ecrire("aprés carre c=", c)
fin
```

```
l'écran
avant carre c=0
aprés carre c=0
```

Exemples

```
Algorithme passage_variable
   var a, c: réel
```

```
Procedure carre (x: réel, var y: réel)
Debut
   y← x*x
fin
```

```
debut
   c←0
   a←3
   ecrire("avant carre c=", c)
   carre(a ,c) // on peut utiliser carre(3,c)
   ecrire("aprés carre c=", c)
fin
```

```
l'écran
avant carre c=0
aprés carre c=9
```

Exemples

```
#include <stdio.h>
void carre (float x, float y){
    y= x*x;
}
```

```
int main(){
    float a, c;
    c=0;
    a=3;
    printf ("avant carre c=%f ", c);
    carre(a ,c); // on peut utiliser carre(a,5)
    printf ("aprés carre c=%f ", c);
    return 0 ;
}
```

l'écran

```
avant carre c=0
aprés carre c=0
```

Exemples

```
#include <stdio.h>
void carre (float x, float *y){
  *y= x*x;
}
```

```
int main(){
   float a, c;
   c=0;
   a=3;
   printf ("avant carre c=%f ", c);
   carre(a ,&c); // on peut utiliser carre(a,5)
   printf ("aprés carre c=%f ", c);
   return 0 ;
}
```

l'écran

```
avant carre c=0
aprés carre c=9
```

Passer d'une procédure à une fonction :

Toute procédure qui renvoie un seul résultat peut être convertie en une fonction,

1. changer le mot Procédure en fonction
2. transformer le paramètre de résultat en une variable locale
3. définissons le type de la fonction comme étant le type de ce paramètre
4. A la fin de la fonction, attribuer la valeur de ce paramètre au nom de la fonction.

Exemple Algorithme

```
Procedure abs(x:réel,var y:réel)
Début
   si x<0 alors
      y← -x
   sinon
      y← x
   finsi
fin
Appel
abs(-5, z)
```

```
fonction abs (x: réel) : réel
   var y: réel
Début
   si x<0 alors
      y← -x
   sinon
      y← x
   finsi
   abs←y
fin
Appel
z←abs(-5)
```

Exemple C

```
void abs (float x, float *y){
   if (x<0)
      *y= -x;
   else
      *y= x;
}
appel
abs(-5, &z);
```

```
float abs (float x){
   float y;
   if (x<0)
      y= -x;
   else
      y= x;
   return y ;
}
appel
z=abs (-5);
```

La variable y peut être omise

Comme on peut omettre **else** qui vient après **return.**

```
float abs (float x){
   if (x<0)
      return -x;
   return x;
}
```

Fin partie 1 du Chapitre 01