•Algorithmique & Langage C
•IUT GEII S1
•
•Les fonctions
•
•Notes de cours
• (sixième partie)

cours_algo_lgc6.10.odp

Licence

Définition

Une fonction est *une suite d'instructions* que l'on regroupe en lui donnant *un nom*.

Cette suite pourra être ensuite utilisée une ou plusieurs fois dans un programme en l'invocant simplement par son nom (et ainsi sans devoir répéter la suite elle même).

Les fonctions peuvent utiliser d'autres fonctions.
On peut ainsi créer des programmes puissants en quelques lignes.

Nous avons déjà utilisé des fonctions :
printf , scanf, rand, sqrt, pow, sin, etc....

Aujourd'hui nous allons apprendre à créer nos propres fonctions.

Un exemple simple

Considérons la suite d'instructions :

```
int i = 0 ;
printf("#");
while ( i < 20 ) {
        printf("*");
        i = i + 1 ; }
printf("#\n");
```

qui permet d'afficher #********************#

(pour mettre en valeur un résultat par exemple)

Un exemple simple

Si nous souhaitons présenter nos résultats comme cela :

```
#********************#
x = 37.2344
#********************#
y = -14.2235
#********************#
z = -56.7890
#********************#
```

Il faudra donc répéter quatre fois la séquence d'instructions précédente entre les affichages des valeurs x, y et z.

Un exemple simple

Il est possible de regrouper cette séquence dans une fonction (en respectant la syntaxe particulière du c) pour pouvoir la réutiliser sans la réécrire entièrement. On va donner un nom à cette séquence par exemple decore.

```
void decore ( void ) {
     int i = 0 ;
    printf("#");
    while ( i < 20 ) {
       printf("*");
       i = i + 1 ; }
    printf("#\n");
}
```

Nous avons créé la fonction decore. Les instructions de cette fonctions sont comprises entre **{** et **}**.

decore est le nom de la fonction.

Un exemple simple

Maintenant, il possible de produire l'affichage souhaité par :

```
float x,y,z ;
....;
....;

decore( );
printf("x = %f\n",x);
decore( );
printf("y = %f\n",y);
decore( );
printf("z = %f\n",z);
decore( );
```

decore(); permet d'exécuter la séquence d'instructions.
On dit qu'on *appelle la fonction* decore.

Voir le programme complet : decoration1.c

Un exemple simple

signifie que l'on ne récupère aucun résultat de cette fonction

signifie que la fonction s'exécute sans qu'on ne lui fournisse aucune valeur

```
void decore ( void ) {
    int i = 0 ;
    printf("#");
    while ( i < 20 ) {
      printf("*");
      i = i + 1 ; }
    printf("#\n");
}
```

La variable i est **locale** à la fonction decore. Elle n'a de sens qu'à l'intérieur du bloc où elle est déclarée.

Un exemple simple

Comme pour les variables, tous les noms choisis
par le programmeur doivent être *déclarés* au compilateur.
il faut donc déclarer le nom "decore" en indiquant qu'il
s'agit d'une fonction à laquelle on ne fournit aucune valeur
et dont on ne récupère aucun résultat.

void decore (void) ;

Cette ligne de déclaration de fonction, doit être placée
avant l'utilisation de la fonction (avant l'appel de la fonction).

Une déclaration de fonction s'appelle **un prototype**.

Allons plus loin ...

La fonction decore précédente n'est pas paramétrable.
On aimerait par exemple pouvoir changer le nombre d'étoiles
pour pouvoir varier la présentation.
Il suffit pour cela de remplacer la valeur 20 par une variable.
La valeur de cette variable sera fixée avant l'exécution de la
séquence :

```
int i = 0 ;
printf("#");
while ( i < n ) {
        printf("*");
        i = i + 1 ; }
 printf("#\n");
```

fonction decore paramétrable

La nouvelle définition de la fonction decore2 sera alors :

```
void decore2 ( int n ) {
        int i = 0 ;
        printf("#");
        while ( i < n ) {
                printf("*");
                i = i + 1 ; }
         printf("#\n");
}
```

La valeur de n sera fournie lors de l'appel de la fonction

et son prototype :

void decore2 (int) ;

decore2 est une fonction qui ne retourne aucune valeur et qui nécessite une valeur de type int pour s'exécuter.

Utilisation de decore2

Il suffit de mettre une expression de type int dont la valeur représente le nombre d'étoiles souhaitées dans les parenthèses de l'instruction d'appel de la fonction :

```
float x,y,z ;
int netoiles = 15 ;
....;

decore2( netoiles );
printf("x = %f\n",x);
decore2(7);
printf("y = %f\n",y);
decore(2 * netoiles );
printf("z = %f\n",z);
decore( 0 );
```

```
#***************#
x = 37.2344
#*******#
y = -14.2235
#*******************************#
z = -56.7890
##
```

Voir le programme complet : decoration2.c

Utilisation de decore2

decore2 est une fonction qui prend une valeur entière en paramètre.
N'importe quelle expression entière peut convenir.
Ce peut être par exemple :
- une constante entière : decore2(7);
- la valeur d'une variable entière : int p ;
  decore2(p);
- une expression composée : decore2(p*p+2*p+3);

- une expression dont la conversion dans le type int à un sens :

```
decore2(7.3);     // 7.3 -> 7 -> 7 étoiles

float x=3.2 ;
decore2(x+2.5) ;  // x + 2.5 -> 5.7 -> 5 étoiles
```


Encore plus fort ...

La fonction decore2 permet d'afficher un nombre variable d'étoiles mais uniquement des étoiles. En rajoutant un paramètre, on va pouvoir faire varier le caractère affiché :

```
void decore3( int n , char c ) {
                    int i = 0 ;
                    printf("#");
                    while ( i < n ) {
                            printf("%c",c);
                            i = i + 1 ; }
                     printf("#\n");
  }
```

et son prototype :

```
void decore3 ( int , char  ) ;
```

Utilisation de decore3

Il y a maintenant deux valeurs séparées par une virgule lors de l'appel de la fonction decore3. Attention il faut respecter l'ordre choisit au moment de la définition de la fonction.

```
float x,y,z ;
int ncar = 10 ;
char car='+' ;
....;

decore3( ncar , car );
printf("x = %f\n", x );
decore3( 7 , 'X' );
printf("y = %f\n", y );
decore3( 2 * ncar , 0x41 );
printf("z = %f\n" , z );
decore3( 3 , '!'+1 );
```

```
#++++++++++#
x = 37.2344
#XXXXXXX#
y = -14.2235
#AAAAAAAAAAAAAAAAAAA#
z = -56.7890
#"""#
```

Voir le programme complet : decoration3.c

Toujours plus fort ...

En rajoutant encore un paramètre on peut varier le caractère initial et final :

```
void decore4( int n , char c , char c_encadre ) {
                int i = 0 ;
                printf("%c",c_encadre);
                while ( i < n ) {
                        printf("%c",c);
                        i = i + 1 ; }
                 printf("%c\n",c_encadre);
        }
```

son prototype : `void decore4 ( int , char , char ) ;`

appels possibles :
```
decore4( 11 , '+' , '{' );   // affiche {++++++++++{
decore4( 8 , '{' , '+' );   // affiche +{{{{{{{{+
```

Retourner une valeur

Comme le nombre de caractères affichés est variable, on peut vouloir connaître le nombre de caractères effectivement affichés par la fonction :

```
int decore5( int n , char c , char c_encadre ) {
                int i = 0 ;
                printf("%c",c_encadre);
                while ( i < n ) {
                        printf("%c",c);
                        i = i + 1 ; }
                 printf("%c\n",c_encadre);
                 return  n + 2 ;
        }
```

return met fin à l'exécution de la fonction et retourne la valeur de l'expression.

Retourner une valeur

Appel :

naff va recevoir la valeur retournée par decore5

```
int  naff ;

naff =  decore5 ( 8 , '*', '#' );
printf("decore5 affiche %d caractere\n" , naff ) ;
```

Prototype :

```
int decore5 ( int , char , char ) ;
```

A la fin de son exécution, la fonction "retourne" une valeur de type int

Un exemple mathématique

Soit la fonction f qui à un réel x positif associe le réel f(x) selon :

$$f(x)=\sqrt{(x)}/(x+1)$$

Créons cette fonction en C :

Prototype : double f (double) ;

Définition :
```
double f ( double x ) {
    double t ;
      t = sqrt ( x ) ;
      return ( t  /  1 + x ) ;
   }
```


Un exemple mathématique

Utilisation de la fonction f :

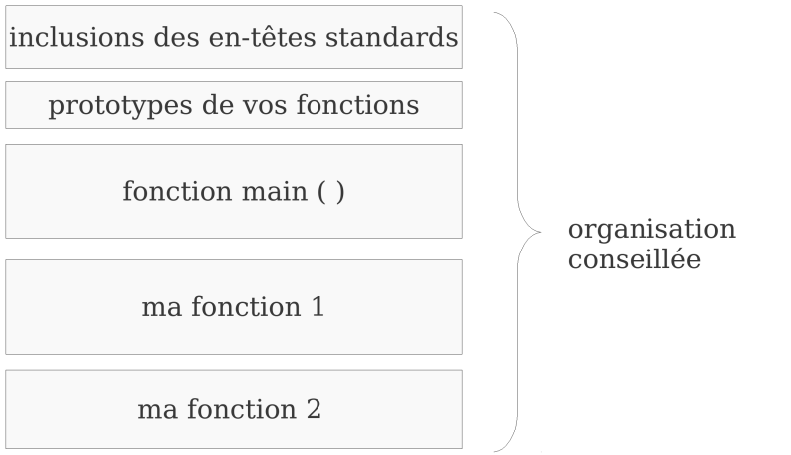
```
int main (void ) {
    double a , b ;
    printf(" Entrez une valeur : ") ;
    scanf("%lf",&a);
    b = f ( a ) ;   // appel de f
    printf("f(%f) = %f\n", a , b ) ;
    system("pause");
    return 0 ;
 }
```

Organisation du fichier source

```
#include <.....h>     // les fichiers .h contiennent les prototypes
#include<......h>    //  des fonctions de la librairie standard
double f ( double ) ;    // prototypes de vos fonctions
void mafonction ( int , int ) ;

int main (void ) {       // programme principal
    double x , y ;
    ....;
    y = f( x );       // appels des fonction
    ....;
    return 0 ;  }
                          // définitions de vos fonctions
double f ( double z ) {
   .....;  }
void mafonction ( int i , int j ) {
    .....; }
```


Organisation du fichier source



En fait, l'ordre des fonctions est quelconque. Il suffit seulement que le prototype apparaisse avant l'appel de la fonction

Fonctions ayant des tableaux en paramètres

Fonction retournant la moyenne des valeurs d'un tableau de 10 éléments de type float :

Prototype :

```
float moyenne_tab ( float [] )  ;
```

ou

```
float moyenne_tab ( float * )  ;
```

Attention il n'y a pas d'incation ni de contrôle de la taille du tableau. En fait c'est seulement l'adresse du premier élément qui est passée à la fonction.

Fonctions ayant des tableaux en paramètres

Définition de moyenne_tab :

```
float moyenne_tab ( float tab [ ]  ) {
   int i  ;
   float s = 0 ;
   for ( i = 0 ; i < 10 ; i = i + 1 ) {
     s = s + tab[i] ; }
   s = s / 10 ;
   return s ;
}
```

Appel :

```
float t[10] ;
float moy ;
moy =   moyenne_tab( t  ) ;
```

Fonctions ayant des tableaux en paramètres

La fonction précédente ne peut traiter que des tableaux de 10 éléments. En passant la taille en paramètre, on peut rendre la fonction plus générale :

```
float moyenne_tab ( float tab * , int taille  )  ;

float moyenne_tab ( float tab * , int taille  ) {
   int i  ;
   float s = 0 ;
   for ( i = 0 ; i < taille ; i = i + 1 ) {
     s = s + tab[i] ; }
   s = s / taille ;
   return s ;
}

moy =  moyenne_tab ( t , 10  ) ;
```


Fonctions sur les chaînes de caractères

Une fonction qui retourne le nombre de caractères 'e' dans une chaîne :

Prototype :

```
int nbe ( char * ) ;
```

une chaîne est un tableau de char

Définition :

```
int nbe ( char * ch ) {
        int ne = 0  , i = 0 ;
        while ( ch[i]!='\0' ) {
                if ( ch[i] == 'e' ) {
                        ne = ne + 1 ; }
               i = i + 1 ;
               }
        return ne ; }
```

Appel :

```
int n ;
n = nbe("ecologiquement" );
```

La fonction peut modifier les éléments d'un tableau

Comme on passe l'adresse du premier élément, une fonction peut agir directement sur les éléments d'un tableau.
Fonction qui remplace les a par des b dans une chaîne :

Prototype : void Remplace_a_par_b (char *) ;

Définition :

```
void Remplace_a_par_b ( char * s ) {
        int i ;
        for ( i = 0 ; s[i] != '\0' ; i++ ) {
                if ( s[i] == 'a' ) {
                        s[i] = 'b' ;  }
        }
```

Appel :

```
char t[]="abracadabra!";
Remplace_a_par_b(t);
printf("%s",t) ;   // bbrbcbdbbrb!
```


Prototypes : const , noms de paramètres

On peut ajouter le mot clé const devant le type d'un paramètre tableau pour indiquer que la fonction ne modifie pas la valeur des éléments du tableau.

```
float moyenne_tab ( const char * , int  ) ;
```

On peut nommer les paramètres dans le prototype d'une fonction pour se rappeler leur signification :

```
float moyenne_tab ( const char * tableau , int taille ) ;
```

(les noms sont alors vus comme des commentaires par le compilateur)

Fonctions sur les chaînes de la librairie standard

La librairie standard du C fournie de nombreuses fonctions bien utiles pour traiter les chaînes de caractères.
Les prototypes se trouvent dans le fichier string.h.

Exemples :

Longueur d'une chaîne :

```
int strlen ( char * ch ) ;
```

Recopie de ch2 dans ch1 :

```
void strcpy ( char * ch1 ,  const char * ch2 ) ;
```

Ajout de ch2 dans ch1 après la fin de ch1 :

```
void strcat ( char * ch1 , const char * ch2 ) ;
```

Comparaison de chaînes :

```
int strcmp ( const char * ch1 , const char * ch2 ) ;
```

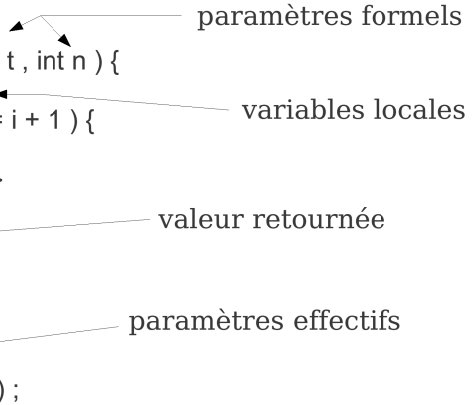
retourne 0 si ch1 et ch2 sont identiques.

voir le fichier d'exemple : fct_chaines.c

Vocabulaire

Prototype : `int maxtab (const int * t , int n ) ;`

Définition :



Appel :

Exercices

Pour chaque fonction, écrire le prototype, la définition et une fonction main() qui appelle une ou plusieurs fois la fonction pour montrer son utilisation.

Une fonction qui retourne la valeur absolue d'un réel.

Une fonction qui retourne le module d'un nombre complexe à partir de sa partie réelle et de sa partie imaginaire.

Une fonction qui retourne le maximum d'un tableau d'entiers

Une fonction qui passe en majuscule une lettre sur n d'une chaîne de caractères. La fonction retournera le nombre de caractères modifiés (et la valeur -1 si n <=0)

Une fonction qui "renverse" une chaîne : "salut" -> "tulas"

Une fonction qui mélange aléatoirement les caractères d'une chaîne : "bonjour" -> "njoourb"

Un fonction qui retourne la valeur entière d'une chaîne représentant un nombre écrit en binaire "100100" -> 36