Algorithmique & Langage C
IUT GEII S1

Les fonctions

Notes de cours

cours_algo_lgc_fonctions.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'invoquant 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.

Fonction sans paramètre ni valeur retournée

Elle permet de regrouper des lignes de codes.

Exemple :

```
void calcul(void) ;

int main(void) {

  printf("begin\n");

  calcul() ;

  printf("end\n");
  return 0 ;

}

void calcul(void){
      int i=5, j;
      printf("i = %d \n",i);
      j=i+4;
      printf("j = %d \n",j);

}
```

prototype (déclaration de la fonction)

Un **appel** de la fonction

variables **locales** à la fonction calcul()

définition de la fonction calcul()

Fonction sans paramètre avec une valeur retournée

Cette fonction retourne une valeur au programme appelant mais ne prend aucun paramètre. Par exemple une fonction qui retourne un nombre aléatoire, une fonction qui lit un capteur, une fonction qui retourne l'heure,....

```
int rand(void) ;

int main(void) {

  int x ;

  x = rand() ;

  printf("x=%d\n",x);

  return 0 ;

}

int rand(void){
      int v;
      ...
      ...
      return v ;
}
```

La variable **x** est **locale** à la fonction **main**().

appel de la fonction rand(), x prend la valeur retournée par rand().

Type de la valeur retournée

La variable **v** est **locale** à la fonction **rand**().
La valeur de v est **retournée** par la fonction rand().

Nb : même si on réutilisait le nom x dans rand, les variables seraient différentes : une variable est locale au bloc {} dans lequel elle est déclarée.

Type `void` (vide)

Ce type est utilisé pour indiquer l'absence de valeur, comme paramètre ou comme valeur retournée.

```
int rand(void) ;
void calcul(void) ;

int main(void) {

  int x ;

  x = rand() ;

  printf("x=%d\n",x);

  calcul() ;

  return 0 ;

}
```

rien !

rien !

En paramètre :
Les fonctions rand() et main() ne prennent aucune paramètre. On ne met donc rien entre les parenthèses lors de l'appel de la fonction

En type de retour :
La fonction calcul() ne retourne aucune valeur. Pas de = lors de l'appel.

```
void calcul(void){
      int i=5, j;
      printf("i = %d \n",i);
      j=i+4;
      printf("j = %d \n",j);
}
```

Fonction avec paramètre(s) et sans valeur retournée

Cette fonction est paramétrable mais ne retourne aucune valeur au programme appelant : une fonction qui démarre un moteur (parmi n), une fonction qui affiche un certain nombre de ligne,....

```
void start_motor( int motor_number) ;

int main(void) {

  int n=4 ;

  start_motor(n) ;

  start_motor(2) ;

  start_motor(n-1) ;

  return 0 ;
}
```

appel avec une variable

appel avec une constante

appel avec une expression

```
void start_motor( int motor_number){
     if(motor_number == 1 ) {
      ...
     else if (motor_number == 2 ) {
      ...
     }
     ...
}
```


Fonction avec paramètre(s) et sans valeur retournée

Par exemple une fonction qui affiche un certain nombre de ligne :

```
void affiche( int lignes ) ;

int main(void) {

  int n=4 ;

  affiche(n) ;

  affiche(2) ;

  affiche(n-1) ;

  return 0 ;
}
```

type `void` (vide) car la fonction ne retourne aucune valeur

```
void affiche( int lignes){
    int i=0 ;
    while ( i < ligne ) {
      printf("voici la ligne %d\n",i+1);
      i = i + 1 ;
     }
}
```

Fonction avec paramètre(s) et sans valeur retournée

Avec deux paramètres :

```
void affiche( int lignes , int valeur ) ;

int main(void) {

  int n=4 , x=5 ;

  affiche(n,x) ;

  affiche(2,2*x) ;

  affiche(n-1,4) ;

  return 0 ;

}

void affiche( int lignes, int valeur){
    int i=0 ;
    while ( i < ligne ) {
      printf( "%d\n",valeur);
      i = i + 1 ;
     }
}
```

prototype

Liste ordonnée des paramètres

Appels en respectant l'ordre
(et le type (*)) des paramètres

Lorsque la fonction s'exécute,
les paramètres prennent les
valeurs passés lors de l'appel

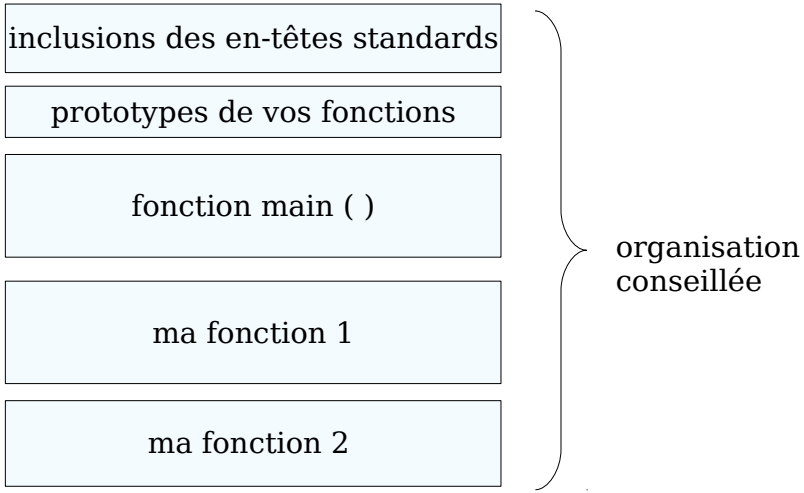
(*) : si lors de l'appel le type n'est pas identique, la valeur est convertie dans le type définit en paramètre. On obtient éventuellement un *warning* lorsque la conversion n'est pas "naturelle".

Fonction avec paramètre(s) et une valeur retournée

Par exemple une fonction qui retourne la surface d'un rectangle à partir de la longueur et de la largeur.

```
float surface( float largeur, float longueur ) ;

int main(void) {

    float lar = 5 , lon = 8 , surf ;

    surf = surface(lar,lon);

    printf("s=%fm2\n",surf);

    return 0 ;
}


float surface( float largeur, float longueur ) {

    float s ;

    s = largeur * longueur ;

    return s ;
}
```

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

Organisation du fichier source

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

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 a , float x ) {
  .....;}
```

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 tab[] )  ;
```

ou

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

Attention il n'y a pas d'indication 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 * ch ) ;
```

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 * s ) ;`

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 * ch , int n ) ;
```

Il n'est pas obligatoire de nommer les paramètres dans le prototype d'une fonction :

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

les noms des paramètres sont vus comme des commentaires par le compilateur. En général il vaut mieux nommer les paramètres pour rappeler leur signification à l'utilisateur de la fonction.

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 :

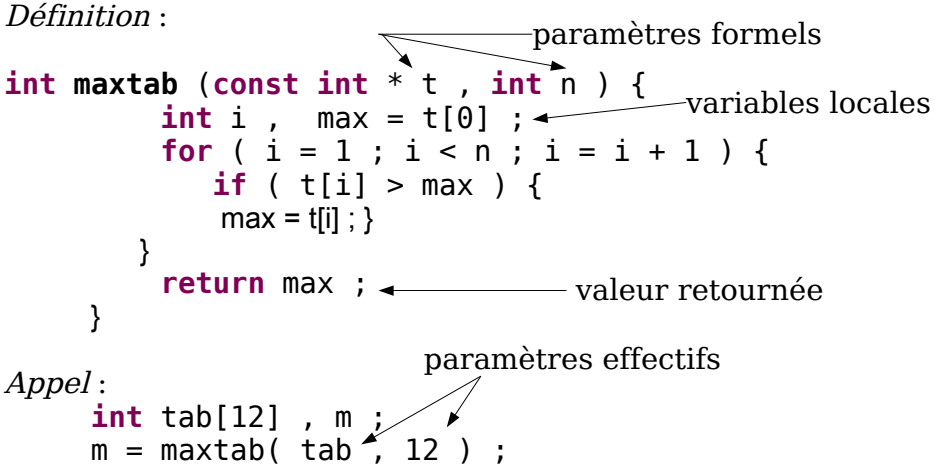
```
int strcmp ( const char * ch1 , const char * ch2 )
```

retourne 0 si ch1 et ch2 sont identiques.

Vocabulaire

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

Définition :



Appel :


```
int tab[12] , m ;
m = maxtab( tab , 12 ) ;
```

Visibilité et durée de vie des variables

Variable globale :
- déclarée hors de tout bloc
- visible (accessible) dans tout le fichier source (le nom n'est pas réutilisable)
- existe pendant tout la durée du programme (*statique*)

A éviter : l'utilisation des variables globales doit être limitée à des cas très particuliers.

```
int var ; // variable globale
int fct ( int x );

int main(void){
     var = 5 ;
     int a=fct(2); // a est locale à main
     // ici a et var ont été modifiées var:6 et a:12
     return 0 ;
}

int fct(int x ){
var++;
return var * x ;
}
```


Visibilité et durée de vie des variables

Variable locale :
- déclarée dans un bloc {}
- visible (accessible) dans tout le bloc ainsi que dans tous les blocs inclus dans le bloc de déclaration (le nom est réutilisable dans les blocs disjoints)
- existe pendant tout la durée d'exécution du bloc (*variable dynamique*)

C'est le type de variable qui est généralement utilisée car cela permet de réutiliser la mémoire et évite de réserver un nom simple pour l'ensemble du programme.

```
int fct ( int n );
int main(void){
int i,n; // i et n sont locals à main
for(i=0; i < 5 ; i++){
n = n + fct(i);
}
return 0 ;
 }
int fct(int n ){// n est ici un nom de paramatère
int i,r;// i et r sont locals à fct
for(i=0;i<n; i++){
r=r+2*i;
}
return r ;}
```

Dans le programme ci contre, il n'y a aucun problème à utiliser plusieurs fois les nom n et i. La communication se fait par le passage de paramètre et la valeur de retour. Des variables i et r de fct sont créées à chaque appel de fct et détruites après chaque return.

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