



Université Chouaïb Doukkali
Faculté des Sciences El Jadida



LES FONCTIONS

Langage C

SMI – S4
FS – EL JADIDA

Pr. BELAQZIZ Salwa
Salwa.belaqziz@gmail.com

2017-2018

Programmation en langage C

Plan

1

Rappels (Types de bases, variables, opérateurs, structures de contrôles,..)

2

Les fonctions

3

Les pointeurs et l'allocation dynamique

4

Les chaines de caractères

5

Les types composés (structures, unions, synonymes)

6

Les fichiers

Fonctions et sous-programmes

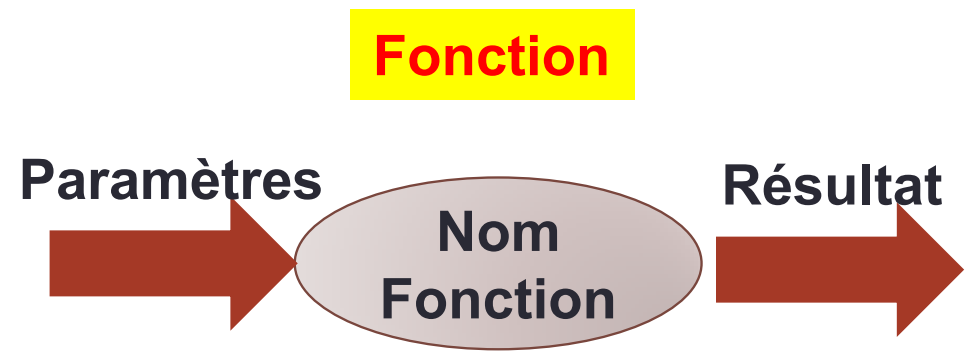
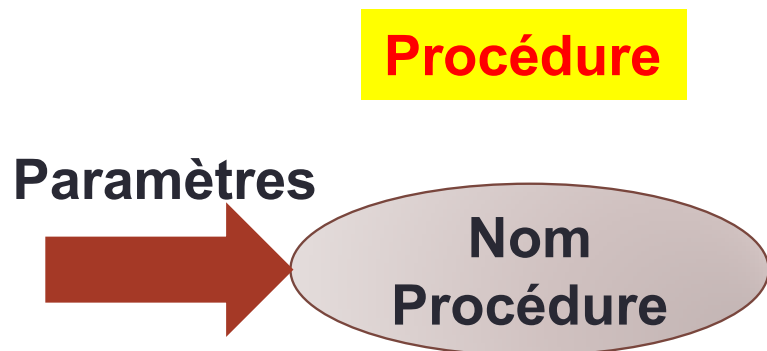
- Ecrire un programme qui résout un problème revient toujours à écrire des sous-programmes qui résolvent des sous parties du problème initial.
- Il existe deux types de sous-programmes :
 - **Les fonctions** : réalisent des traitements en se servant des valeurs de certaines variables et renvoient un résultat. Elles se comportent comme des fonctions mathématiques : $y=f(x, y, \dots)$
 - **Les procédures** : réalisent seulement des traitements mais ne renvoies aucun résultat

Fonctions et Procédures

- Les fonctions et les procédures sont des groupes d'instructions indépendants désignés par un nom (identificateur).
- Elles ont plusieurs intérêts :
 - permettent de "factoriser" les programmes, c-à-d de mettre en commun les parties qui se répètent
 - permettent une structuration et une meilleure lisibilité des programmes
 - facilitent la maintenance du code (il suffit de modifier une seule fois)
 - ces procédures et fonctions peuvent éventuellement être réutilisées dans d'autres programmes

Fonctions et Procédures

- Une **procédure** effectue un traitement **sans renvoyer aucune valeur**.
- Une **fonction** effectue le traitement et **renvoie une valeur** (en générale le résultat du traitement).
- En programmation C, il n'y a pas de différence entre fonctions et procédures dans la déclaration mais plutôt dans la façon d'utilisation



Déclaration & Syntaxe

Déclaration

- Déclarée en dehors de la fonction principale « main »
- Peut être appelée de n'importe où

Syntaxe

```
<type> <nom_fonction> ([type1 arg1[,type2 arg2, ...]])  
{  
    //Corps de la fonction  
    [return (expression)]  
}
```

- La première ligne c'est l'en-tête de la fonction
- **type** est le type du résultat retourné. Dans le cas où la fonction ne retourne pas de résultat, elle est de type void. Et on n'écrit pas l'instruction return (expression)
- **nom_fonction** : le nom ou l'identificateur de la fonction
- entre **parenthèses**, on spécifie les arguments (paramètres) de la fonction et leurs types.
- **return (expression)**: le mot clé return suivi de la valeur à renvoyer (qui doit être du même type que la fonction)

Déclaration & Syntaxe

Remarques

- Dans le cas où une fonction n'a pas d'argument en entrée, on peut déclarer la liste des paramètres comme (**void**) ou simplement laisser vide entre parenthèses ()
- Si une fonction doit retourner une valeur, on fait appel au mot clé « **return** ».
 - Le type de retour doit être le même que le type de la fonction dans l'entête
 - Peut être utilisé n'importe où dans le code (là où on veut quitter la fonction en retournant une valeur)
 - Peut exister plusieurs fois dans une fonction
- Pour les fonctions ne retournant rien, soit on utilise l'instruction `return` sans argument, soit on met rien.
- Les paramètres (arguments) des fonctions sont des variables **locales** au bloc d'instructions de la fonction.
 - Cependant, on peut définir d'autres variables dans le bloc
- Une fonction peut avoir des arguments de différents types

Exemples

1. **SommeCarre** : calcule la somme des carrées de deux réels x et y

```
float SommeCarre (float x, float y )
{
    float z;
    z  = x*x+y*y;
    return z; // on retourne le résultat à l'aide de l'instruction return
}
```

2. **Pair** : détermine si un nombre est pair

```
int Pair(int x)
{
    if (x%2 == 0)
        return 1; // vue qu'il n'y a pas de type booléen en C, on retourne 1
    else //pour dire que c'est vrai et 0 pour faux
        return 0;
}
```


Exemples

3. **Affiche**: affiche le message Bonjour

```
void Affiche ( )  
{  
    printf("Bonjour \n");  
    /* L'affichage à l'écran n'est pas considéré comme un résultat en C, donc  
       pas besoin de l'instruction return */  
}
```

4. **ValeurAbsolue**: affiche la valeur absolue d'un entier passé en paramètre

```
int ValeurAbsolue ( int X )  
{  
    int valeur;  
    if (X >= 0 )  
        valeur = X;  
    else  
        valeur = - X;  
    return valeur;  
    /* remarquer que le type de la variable valeur est le même que celui du  
       retour de la fonction ValeurAbsolue  
    */  
}
```

Prototype

- Pour certain compilateur, il est obligatoire de donner la définition d'une fonction. Si une fonction est définie après son premier appel, elle doit être déclarée auparavant.
- La déclaration d'une fonction se fait en haut après les inclusions « include » par son **prototype** et qui indique le type de la fonction ainsi que les types des arguments:

Type NomFonction (type1 [param1],..., typeN [paramN]);

```
#include <stdio.h>  
float multiplication (float, float);
```

Ou bien

```
#include <stdio.h>  
float multiplication (float a, float b);
```

Fonction « main »

- **int main ()** est une fonction particulière qui retourne un entier et dont la liste des paramètres est vide. Elle est appelée la fonction principale.
- **Tout programme doit contenir obligatoirement la fonction principale, qui est exécutée lorsque le programme est lancé.**

```
int main ( )  
{  
    ...  
    return 0;  
}
```

Appel d'une fonction

- On appelle une fonction en utilisant son nom, avec la valeur des arguments entre parenthèses, dans l'ordre de sa définition.

NomFonction (para1,..., paraN)

Remarques:

- Il faut respecter l'ordre et les types des arguments
- Dans le cas d'une fonction ne retournant rien, l'appel de la fonction sera direct : Affiche ();
- Dans le cas d'une fonction qui retourne une valeur, la fonction doit être utilisée dans une instruction d'affectation :
 - x=Pair(5);
 - printf("%d", Pair(6));
- Lors de l'appel d'une fonction, les paramètres sont appelés paramètres effectifs (ou arguments) : ils contiennent les valeurs pour effectuer le traitement.
- Lors de la définition, les paramètres sont appelés paramètres formels

Appel d'une fonction

Exemple :

```
#include <stdio.h>
int ValeurAbsolue (int X);
int main()
{
    int val1, val2;

    printf("Saisir un entier");
    scanf ("%d", &val1);

    val2 = ValeurAbsolue(val1);
    printf("la valeur absolue de %d est : %d ", val1, val2);

    return 0;
}

int ValeurAbsolue (int X)
{
    int valeur;
    if (X >= 0 )
        valeur = X;
    else
        valeur = -X;

    return valeur;
}
```

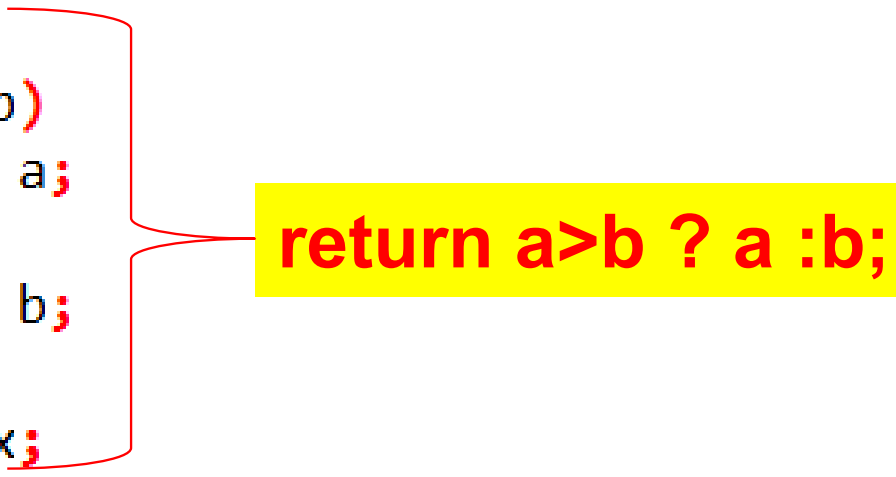
Exercice

- Ecrire une fonction **Maxi2** qui retourne le maximum de deux entiers donnés.
- Ajouter une autre fonction, nommée **Maxi3** qui retourne le maximum de 3 valeurs passés en argument, en utilisant la fonction **Maxi2**.
- Ecrire un programme qui demande à l'utilisateur de saisir trois réels et qui affiche leur max

Exercice

- Ecrire une fonction **Maxi2** qui retourne le maximum de deux réels donnés.

```
// Fonction qui calcule le max de 2 réels  
float max2(float a, float b )  
{  
    int max;  
    if (a >= b)  
        max = a;  
    else  
        max = b;  
    return max;  
}
```



return a>b ? a :b;

Exercice

- Ajouter une autre fonction, nommée **Maxi3** qui retourne le maximum de 3 valeurs passés en argument, en utilisant la fonction **Maxi2**.

```
/* Fonction qui calcule le max de 3 réels */  
float max3 (float a, float b, float c )  
{  
    float m, max;  
  
    m = max2(a,b);  
    max = max2(m,c);  
  
    return max;  
}
```

return c > Max2(a,b) ? c : Max2(a,b);

return Max2(Max2(a,b),c);

Exercice

- Ecrire un programme qui demande à l'utilisateur de saisir trois réels et qui affiche leur max

```
#include <stdio.h>

float max2(float a, float b) ;
float max3 (float a, float b, float c);

/* Fonction principale qui lance l'exécution */
int main()
{
    float x, y, z, max ;

    printf("Saisir trois réel : ");
    scanf ("%f%f%f",&x,&y,&z);

    max = max3(x,y,z);

    printf("La max est %f : ", max);

    return 0;
}
```

Variables locales & globales

- Chaque variable déclarée est accessible dans son contexte de déclaration
- Une variable définie à l'intérieur d'une fonction est une **variable locale à la fonction**, et elle n'est connue qu'à l'intérieur de cette fonction
 - Elle est créée à l'appel de la fonction et détruite à la fin de son exécution
- Une variable définie à l'extérieur des fonctions est **une variable globale**
 - Elle est définie durant toute l'application et peut être utilisée et modifiée par toutes les fonctions du programme
- Une variable locale cache la variable globale qui a le même nom
- Une variable déclarée dans un bloc d'instructions est uniquement visible à l'intérieur de ce bloc
 - C'est une variable locale à ce bloc, elle cache, localement, toutes les variables du même nom des blocs qui l'entourent

Variables locales & globales

Exemple :

```
#include <stdio.h>
void affecte(int a);

int a=150; // variable globale

int main()
{
    int a=10; // variable locale à main cache a globale
    affecte(a);
    printf("dans main a=%d\n",a);

    return 0;
}

void affecte(int a)
{
    // a est une variable locale à affecte cache a globale
    printf("dans affecte avant affectation a=%d\n",a);
    a=5;
    printf("dans affecte après affectation a=%d\n",a);
}
```

dans affecte avant affectation a=10
dans affecte après affectation a=5
dans main a=10

Paramètres formels & effectifs

- Les paramètres servent à échanger des données entre la fonction appelante et la fonction appelée

Vocabulaire

- Un paramètre formel : appelé paramètre
- Un paramètre effectif : appelé argument

Signification

- Les paramètres placés dans la déclaration d'une fonction sont des **paramètres formels**. Ils peuvent prendre toutes les valeurs possibles dans le type déclaré mais ils sont abstraits (n'existent pas réellement)
- Les paramètres placés dans l'appel d'une fonction sont des **paramètres effectifs**. Ils contiennent les valeurs pour effectuer le traitement

Paramètres formels & effectifs

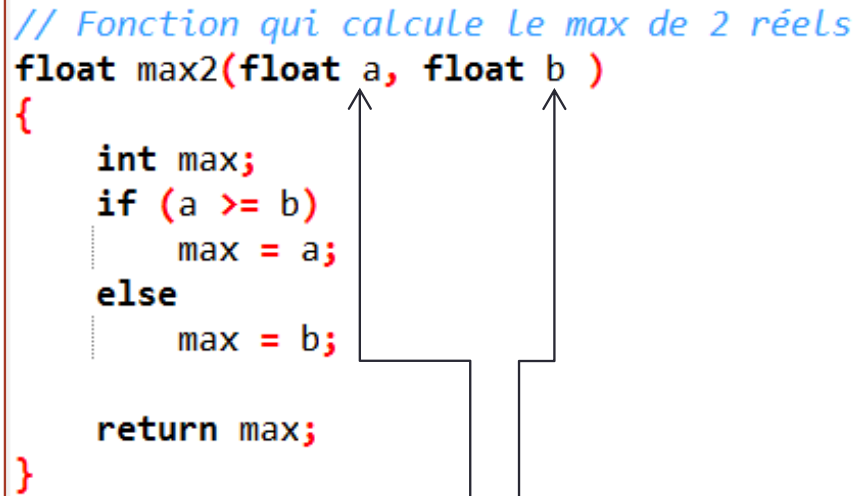
- **Un paramètre effectif est une variable ou constante (numérique ou définie par le programmeur)**
- **Le paramètre formel et le paramètre effectif sont associés lors de l'appel de la fonctions.**

Donc :

- **Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels.**
- **L'ordre et le type des paramètres doivent correspondre**

Paramètres formels & effectifs

```
// Fonction qui calcule le max de 2 réels
float max2(float a, float b)
{
    int max;
    if (a >= b)
        max = a;
    else
        max = b;
    return max;
}
```

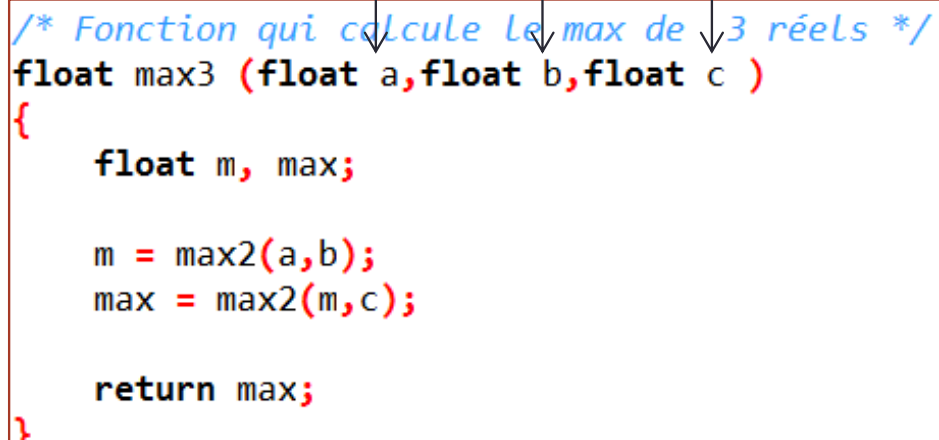


Paramètres formels

```
/* Fonction qui calcule le max de 3 réels */
float max3 (float a, float b, float c)
{
    float m, max;

    m = max2(a,b);
    max = max2(m,c);

    return max;
}
```



```
#include <stdio.h>

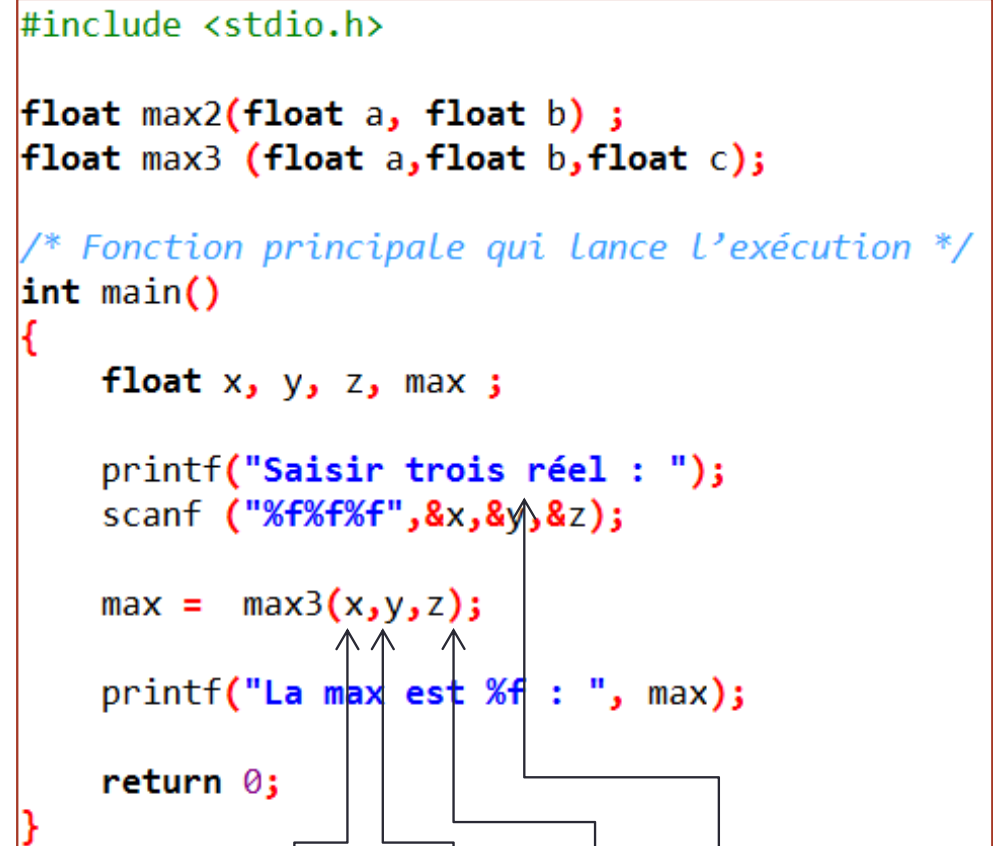
float max2(float a, float b) ;
float max3 (float a, float b, float c);

/* Fonction principale qui lance l'exécution */
int main()
{
    float x, y, z, max ;

    printf("Saisir trois réel : ");
    scanf ("%f%f%f",&x,&y,&z);

    max = max3(x,y,z);
    printf("La max est %f : ", max);

    return 0;
}
```



Paramètres effectifs

Transmission des paramètres

Réflexion: quel est le résultat d'exécution des deux programmes

```
#include <stdio.h>
void affecte(int a);

int main()
{
    int a=10;
    affecte(a);
    printf("a = %d\n",a);
    return 0;
}

void affecte(int a)
{
    a=5;
}
```

a = 10

```
#include <stdio.h>
void affecte(int *);

int main()
{
    int a=10;
    affecte(&a);
    printf("a=%d\n",a);
    return 0;
}

void affecte(int *a)
{
    *a=5;
}
```

a = 5

Transmission des paramètres

Passage par valeur

- La transmission des paramètres en C est par défaut par valeur.
- Ainsi quand une variable passée en argument à une fonction c'est sa valeur qui est passée

Passage par référence (adresse)

- Pour effectuer une transmission par adresse en C, on déclare le paramètre formel de type **pointeur** et lors d'un appel de la fonction, on envoie **l'adresse** et non la valeur du paramètre effectif.

```
void Incrementation (int n, int *m)
{
    n=n+1;
    *m =*m+1;
}
```

```
int main( )
{
    int a = 1, b=2;
    Incrementation(a, &b);
    printf("a = %d et b=%d \n", a,b);
}
```

a=1 et b=3

Transmission des paramètres

Exemple : passage d'un tableau en paramètre

```
/* Passage par référence */
void remplirTab1 (int tab[],int taille)
{
    int i;
    for (i =0; i < taille; i++)
        tab[i] = i;
}

/* Passage par référence */
void remplirTab2 (int tab[],int taille)
{
    int i;
    for (i =0; i < taille; i++)
        tab[i] = 0;
}

void afficher(int tab[], int taille )
{
    int i =0;
    for (i =0; i < taille; i++)
        printf("La valeur de la case %d est %d \n", i, tab[i] );
}

int main ()
{
    int tab[3];

    remplirTab1(tab, 3);
    afficher(tab, 3);
    remplirTab2(tab, 3);
    afficher(tab, 3);

    return 0;
}
```

La valeur de la case 0 est 0
La valeur de la case 1 est 1
La valeur de la case 2 est 2
La valeur de la case 0 est 0
La valeur de la case 1 est 0
La valeur de la case 2 est 0

Transmission des paramètres

- **Types simples : int, float, char,**

Passage par valeur

- **Types : tableaux, pointeurs**

Passage par référence

Récurtivité

- Une fonction est réursive si elle peut s'appeler elle-même
- Pour éviter une récurtivité infinie, il faut fixer un cas limite (cas trivial) qui arrête la récurtivité

```
#include <stdio.h>
int fact(int N);

int main()
{
    int a=8, f;
    f=fact(a);
    printf("!=d=%d\n",a,f);

    return 0;
}

int fact(int N)
{
    if ( N==1 )
        return 1;
    else
        return ( N * fact (N-1) );
}
```

Exercice

- Ecrire une fonction **remplirTab** qui permet de remplir un tableau de réels. Les valeurs sont saisies par l'utilisateur.
- Ecrire une fonction **afficheTab** qui permet d'afficher un tableau de réels.
- Ecrire une fonction **incTab** qui permet d'incrémenter les valeurs des cases d'un tableau.
- Ecrire la fonction **main** qui
 - demande à l'utilisateur de donner le nombre de notes à saisir,
 - demande à l'utilisateur de saisir les notes
 - incrémente les valeurs saisies
 - affiche le résultat

Exercice

- Ecrire une fonction une fonction **remplirTab** qui permet de remplir un tableau de réels. Les valeurs sont saisies par l'utilisateur.

```
void remplirTab (float tab[],int taille)
{
    int i;
    for (i =0; i < taille; i++)
    {
        printf("Saisir la valeur de la case %d ", i);
        scanf("%f", &tab[i]);
    }
}
```

Exercice

- Ecrire une fonction **afficheTab** qui permet d'afficher un tableau de réels.

```
void afficheTab(float t[], int taille)
{
    int i =0;
    for (i =0; i<taille; i++)
        printf("La valeur de la case %d est %f \n", i, t[i]);
}
```

Exercice

- Ecrire une fonction **incrTab** qui permet d'incrémenter les valeurs des cases d'un tableau.

```
void incrTab(float t[], int taille)
{
    int i =0;
    for (i =0; i<taille; i++)
        t[i] = t[i]+1;
}
```

Exercice

- Ecrire la fonction **main**

```
#include <stdlib.h>
#include <stdio.h>

void remplirTab (float tab[],int taille);
void afficheTab(float t[], int taille);
void incrTab(float t[], int taille);

int main()
{
    int n;
    float tab[10] ;

    printf("Saisir le nombre des notes ( max est 10 )");
    scanf ("%d",&n);

    remplirTab(tab, n);

    incrTab(tab,n);

    afficheTab(tab,n);

    return EXIT_SUCCESS;
}
```