

COURS 4

Programmation impérative

Les Fonctions

- Syntaxe
- Passage de paramètres
 - par valeur
 - par adresse
- Récursivité

SOMMAIRE

- Informations pratiques
- Introduction
- Éléments de base
 - Programmer en Langage C – Compilation
 - Structure d'un programme / Règles d'écritures
 - Types de base
 - Constantes/Variables
 - Opérateurs
 - Instructions de contrôle
 - Pointeurs
 - Tableaux
- Fonctions
- Chaînes de caractères
- Pointeurs- Tableaux-Fonctions
- Types Construits
- Entrées – Sorties sur Fichiers
- Compilation séparée
- Implémentation de Types Abstraits de Données

Fonctions

- Généralités

- fonction permet de partitionner les gros traitements en tâches plus petites
- fonction permet de construire les programmes à partir de briques déjà existantes
- le découpage en fonction permet de gérer explicitement la communication entre ces modules



code modulaire : plus souple, plus fiable et mieux réutilisable

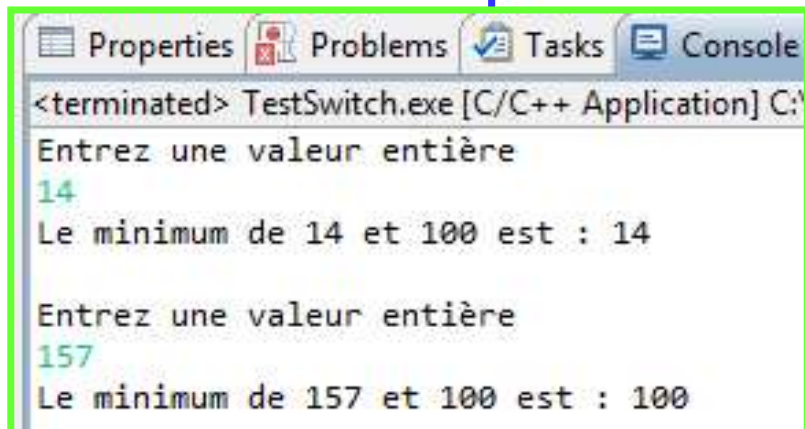
Fonctions

- Illustration

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

int main(){
    int x, y, minimum;
    const int nb=100;

    printf("Entrez une valeur entière\n");
    fflush(stdout);
    scanf("%d", &x);
    minimum = x<nb ? x : nb;
    printf (" Le minimum de %d et %d est : %d\n",x,nb, minimum);
    printf("Entrez une valeur entière\n");
    fflush(stdout);
    scanf("%d", &y);
    minimum = y<nb ? y : nb;
    printf (" Le minimum de %d et %d est : %d\n",y,nb, minimum);
    return EXIT_SUCCESS;
}
```

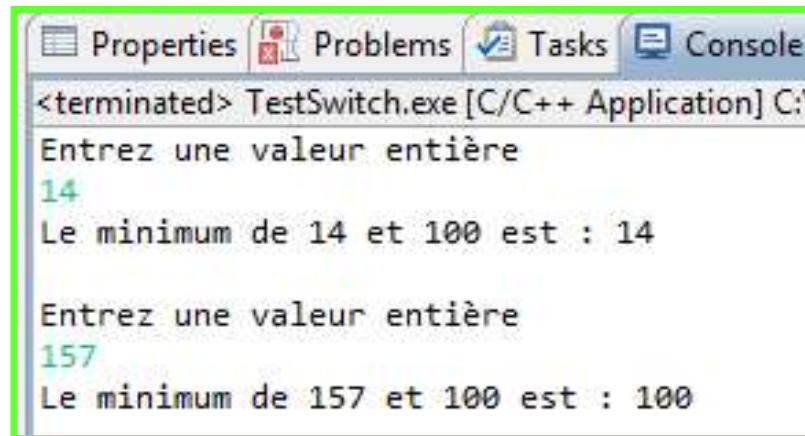


```
<terminated> TestSwitch.exe [C/C++ Application] C:\
Entrez une valeur entière
14
Le minimum de 14 et 100 est : 14

Entrez une valeur entière
157
Le minimum de 157 et 100 est : 100
```

Fonctions

- Illustration



```
<terminated> TestSwitch.exe [C/C++ Application] C:\
Entrez une valeur entière
14
Le minimum de 14 et 100 est : 14

Entrez une valeur entière
157
Le minimum de 157 et 100 est : 100
```

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

const int nb=100;

int SaisieEntier(){
    int x;
    printf("Entrez une valeur entière\n");
    fflush(stdout);
    scanf("%d", &x);
    return x;
}

int calculMin(int x){
    return(x<nb?x:nb);
}

int main(){
    int nb1, nb2, minimum;

    nb1=SaisieEntier();
    minimum = calculMin(nb1);
    printf ("Le minimum de %d et %d est : %d\n\n",nb1,nb, minimum);

    nb2=SaisieEntier();
    minimum = calculMin(nb2);
    printf ("Le minimum de %d et %d est : %d\n",nb2,nb, minimum);
    return EXIT_SUCCESS;
}
```



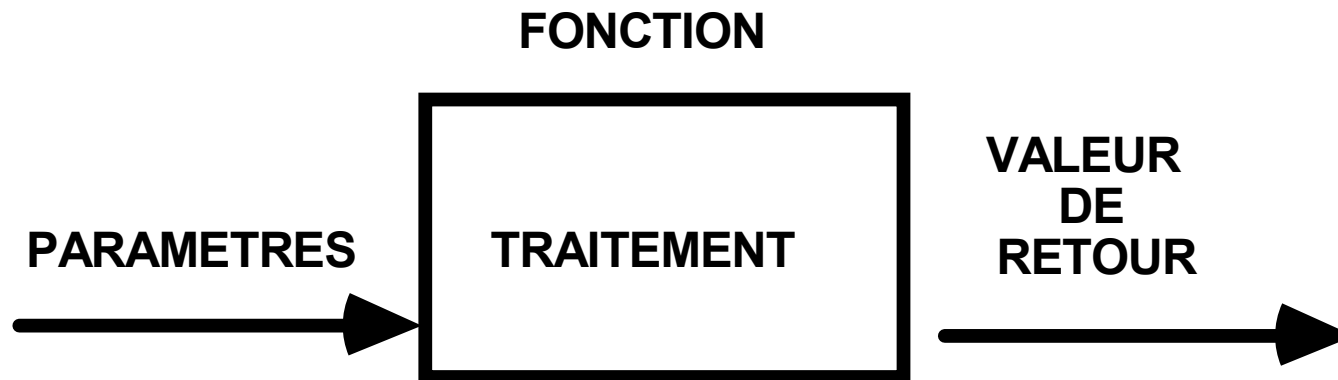
Listing du programme plus long
Mais
Programme principal plus court et mieux structuré

Fonctions

- Parties de code source qui permettent
 - de réaliser le même de type de traitement plusieurs fois, sur des variables différentes
- Une fonction communique avec le reste du programme par le biais d'une interface
 - Spécifiée à la compilation

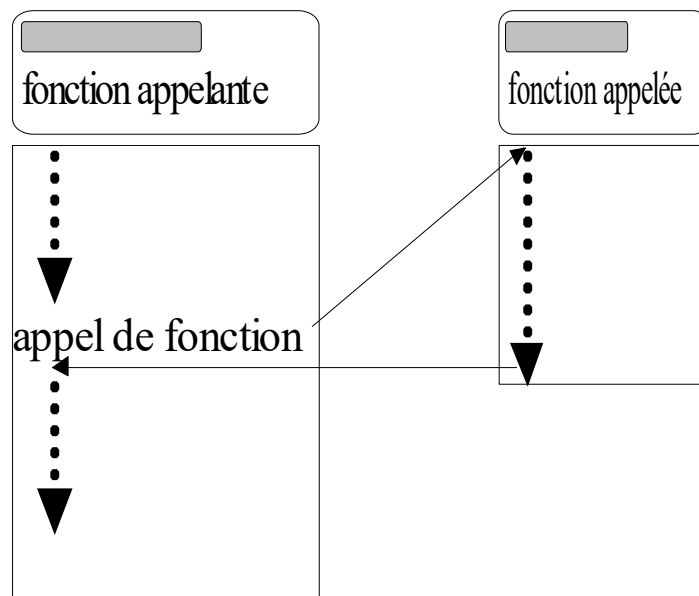
Fonctions

- L'appel de la fonction correspond
 - À un échange de données à travers cette interface
 - Au traitement de ces données dans le corps de la fonction
 - À un retour de résultat via cette interface



Fonctions

- Schéma d'un appel de fonction



5 étapes

1. Mise en pile des paramètres
Fonction appelante empile les copies des paramètres
2. Saut à la fonction appelée
 $f_{\text{appelante}}$ provoque un saut à l'adresse de déb de la fonction appelée et empile l'adresse de retour
3. Prologue dans $f_{\text{appelée}}$
fonction prépare son environnement en positionnant un pointeur de contexte dans la pile et sauvegarde l'ancien pointeur de contexte
⇔ rangement d'un côté des paramètres de la fonction et de l'autre côté les variables locales à la fonction
4. Exécution de $f_{\text{appelée}}$ jusqu'au `return` ou `}`
5. Epilogue dans $f_{\text{appelée}}$
restitution du contexte de la fonction appelante

Fonctions

Définition d'une fonction	
<pre>typeRetour nomFonction (type1 param1, type2 param2) { instruction_1; instruction_2; instruction_n; return valeur; }</pre>	<p>interface ⇔ Signature de la fonction</p> <p>bloc d'instructions ⇔ corps de la fonction</p> <p>valeur doit être de type typeRetour</p> <p>Si pas de typeRetour précisé ⇔ par défaut int</p>
<pre>int Somme (int a, int b){ int sum ; sum = a + b ; return sum; }</pre>	<p>Déf de la fonction Somme</p> <ul style="list-style-type: none">• prend en paramètres 2 variables de type int• renvoie une valeur de type int

Fonctions

Règles	
Pas de déclaration de fonction à l'intérieur d'une autre fonction	<pre>typeRet fonction(type param){ typeRet2 fonction2(type2 param2){ return valeur; } return val; }</pre> 
Une fonction doit être définie avant d'être appelée ou elle doit être prototypée en début de fichier	<pre>typeRet2 fonction2(type param2){ return valeur; } typeRet fonction(type param){ typeRet2 x; x= fonction2(param); return x; }</pre>

Fonctions

- Prototypage de la fonction

- Déclaration de la signature de la fonction en début de fichier
 - Permet au compilateur de vérifier les paramètres d'entrée et de sortie, ainsi que leur type lors de l'appel de la fonction

Sans Prototypage

```
typeRet2 fonction2(type param2){
    ...
    return valeur;
}

typeRet2 fonction( type param){
    ...
    typeRet2 x;
    x=fonction2(param);
    return x;
}
```

Avec Prototypage

```
typeRet2 fonction2(type );

typeRet2 fonction( type param){
    ...
    typeRet2 x;
    x=fonction2(param);
    return x;
}

typeRet2 fonction2(type param2){
    ...
    return valeur;
}
```

Fonctions

Retour de Fonction	
Valeur de retour	<ul style="list-style-type: none">• Si fonction ne renvoie pas de valeur<ul style="list-style-type: none">⇔ type de retour est void⇔ mot clé return pas nécessaire⇔ fin de l'exécution à l'accolade }• 1 et 1 seule valeur de retour qui doit être de type<ul style="list-style-type: none">• scalaire (short, int, char, float, double)• pointeur• structure• Mais pas de type tableau <p>Génération de la valeur retour est provoquée par l'appel de l'instruction return</p>
return ; return expression;	<ul style="list-style-type: none">• Provoque la sortie immédiate de la fonction• Provoque la sortie immédiate de la fonction et renvoie la valeur de l'expression évaluée

Fonctions

Appel de Fonction

`nomDeFonction (Liste des paramètres) ;`

Appel de fonction sans type de retour

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

#include <stdio.h>

void echange (int a, int b){
    int aux;
    aux = a;
    a=b;
    b=aux;
    printf("à la fin d'échange: %d et %d \n",a,b);
}

int main(){
    int x=5, y=8;

    printf("avant appel d'échange: %d et %d \n",x,y);
    echange(x , y);
    printf("après appel d'échange: %d et %d \n",x,y);
    return EXIT_SUCCESS;
}
```

Fonctions

Appel de Fonction

`val=nomDeFonction (Liste des paramètres) ;`

avec

`val` de même type que le type de retour de la fonction

Appel de fonction qui retourne une valeur

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

const int nb=100;

int SaisieEntier(){
    int x;
    printf("Entrez une valeur entière\n");
    fflush(stdout);
    scanf("%d", &x);
    return x;
}

int calculMin(int x){
    return(x<nb?x:nb);
}

int main(){
    int nb1, nb2, minimum;

    nb1=SaisieEntier();
    minimum = calculMin(nb1);
    printf ("Le minimum de %d et %d est : %d\n\n",nb1,nb, minimum);

    nb2=SaisieEntier();
    minimum = calculMin(nb2);
    printf ("Le minimum de %d et %d est : %d\n",nb2,nb, minimum);
    return EXIT_SUCCESS;
}
```

Fonctions

Appel de Fonction	
formel	<p>apparaît dans la définition de la fonction</p> <p>⇔ nom sous lequel un argument d'une fonction est connu à l'intérieur de celle-ci</p> <p>Au niveau de la visibilité</p> <p>⇔ même visibilité qu'une variable locale à la fonction</p>
effectif	<p>objet substitué au paramètre formel lors de l'exécution de la fonction</p> <p>⇔ argument fourni lors de l'appel de la fonction</p> <p>⇔ copie de la valeur du paramètre effectif dans l'espace mémoire géré par la pile (prologue $f_{\text{appelée}}$) qui correspond au paramètre formel</p>



la correspondance entre paramètres formels et effectifs est établie selon l'ordre d'apparition dans l'en-tête de la fonction

Fonctions

- Paramètres formels/effectifs

```
/* définition de la fonction Min */
int Min (int a, int b){ /*a b sont les paramètres formels*/
    return(a<b ? a : b);
}

int main(){
    int x=5,y=8,u,v,mini;
    mini=Min(x , y);

    printf("minimum de %d et %d est %d\n",x,y,mini);

    u=8; v=9;
    mini=Min(u , v);

    printf("minimum de %d et %d est %d\n",u,v,mini);
    return 1;
}
```

appel de la fonction Min avec les **paramètres effectifs**

x et **y**

⇔ **x** va devenir synonyme de **a**, et

y synonyme de **b**

copie de la valeur de **x** dans **a** et de **y** dans **b**

/* appel de la fonction Min avec les **paramètres effectifs u et v**

⇔ **u** va devenir synonyme de **a**, et

v synonyme de **b**

copie de la valeur de **u** dans **a** et de **v** dans **b**

Fonctions

Appel de Fonction	
par valeur	<p>création d'une copie de l'objet paramètre effectif qui reçoit le nom du paramètre formel</p> <p>seules les valeurs des copies peuvent être changées par la fonction appelée et non les valeurs des paramètres effectifs.</p>
par adresse	<p>l'adresse de la variable est passée en paramètre et non la valeur</p>

Fonctions

- Passage par valeur

```
#include <stdio.h>

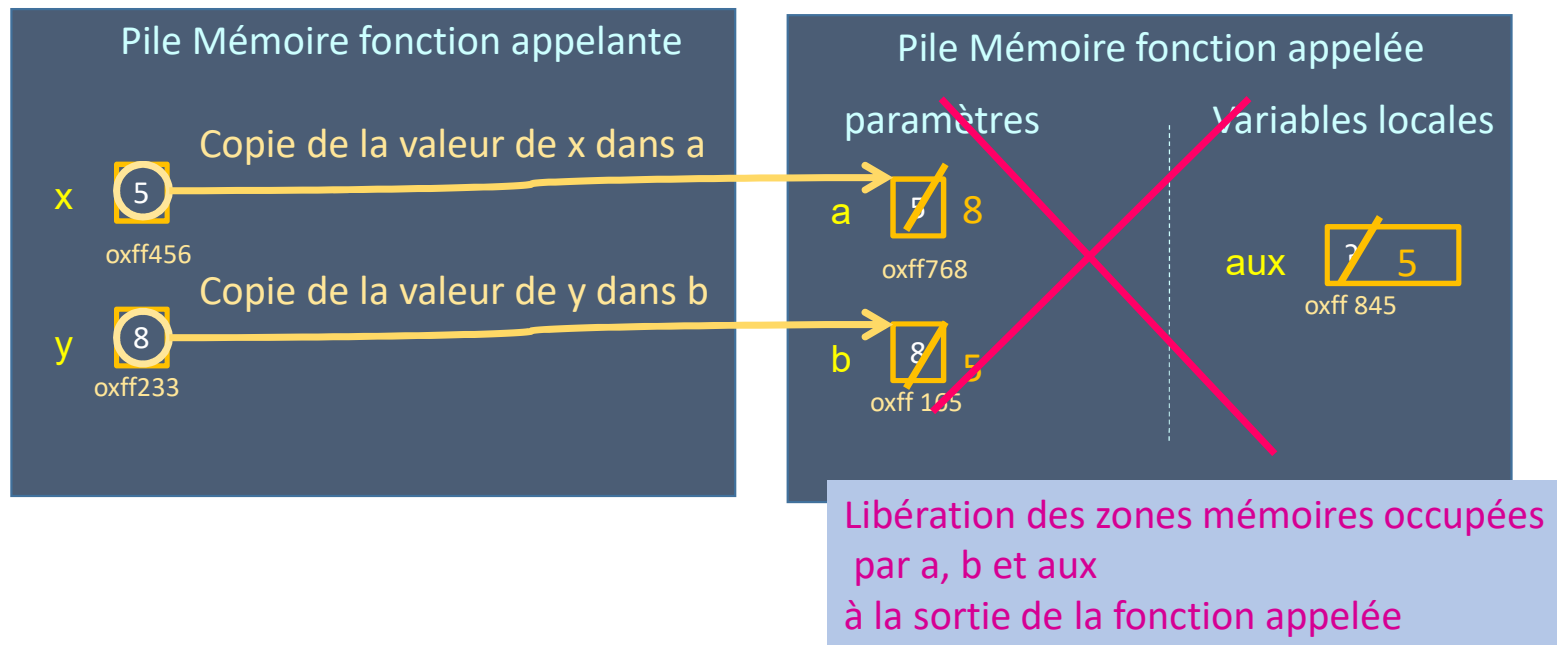
void echange (int a, int b){/* définition de la fonct. echange */
    int aux;
    printf("au debut d'echange: %d et %d \n",a,b);
    aux = a;
    a=b;
    b=aux;
    printf("à la fin d'echange: %d et %d \n",a,b);
}

int main(){
    int x=5, y=8;
    printf("avant appel d'echange: %d et %d \n",x,y);
    echange(x , y);
    printf("après appel d'echange: %d et %d \n",x,y);
    return 1;
}
```

```
avant appel d'echange: 5 et 8
au debut d'echange: 5 et 8
à la fin d'echange: 8 et 5
après appel d'echange: 5 et 8
```

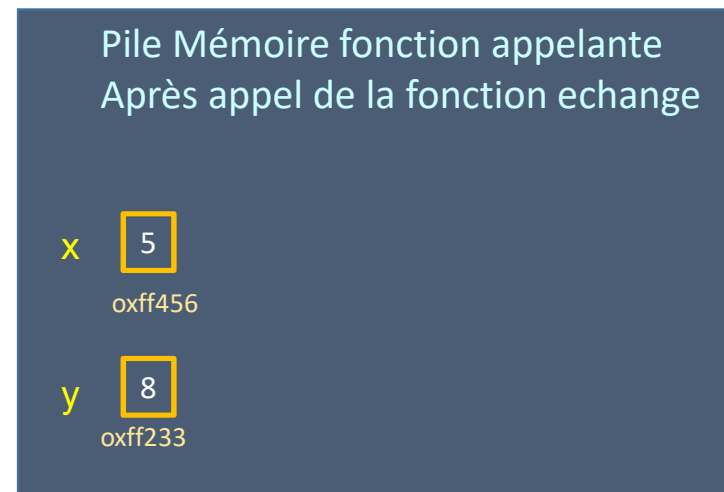
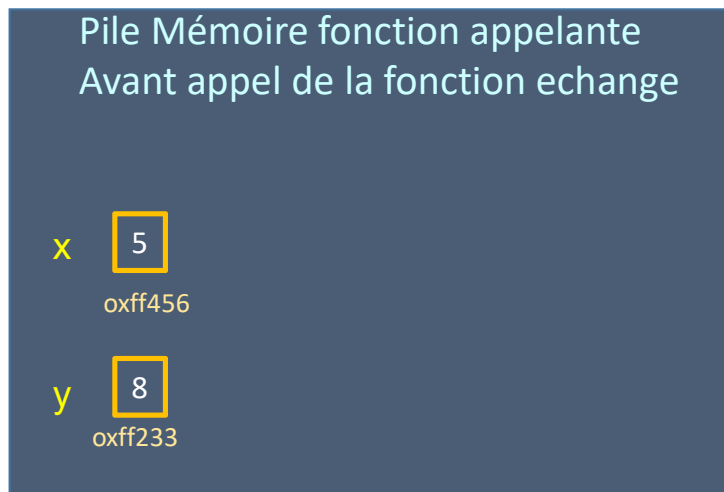
Fonctions

- Passage par valeur



Fonctions

- Passage par valeur
 - état de la mémoire avant et après l'appel de la fonction



Fonctions

- Passage par adresse

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

void echange (int *a, int *b){/* définition fonct. echange */
    int aux;
    printf("au debut d'echange: %d et %d \n",*a,*b);
    aux = *a;
    *a=*b;
    *b=aux;
    printf("à la fin d'echange: %d et %d \n",*a,*b);
}

int main(){
    int x=5, y=8;
    printf("avant appel d'echange: %d et %d \n",x,y);
    echange(&x , &y);
    printf("après appel d'echange: %d et %d \n",x,y);
    return EXIT_SUCCESS;
}
```

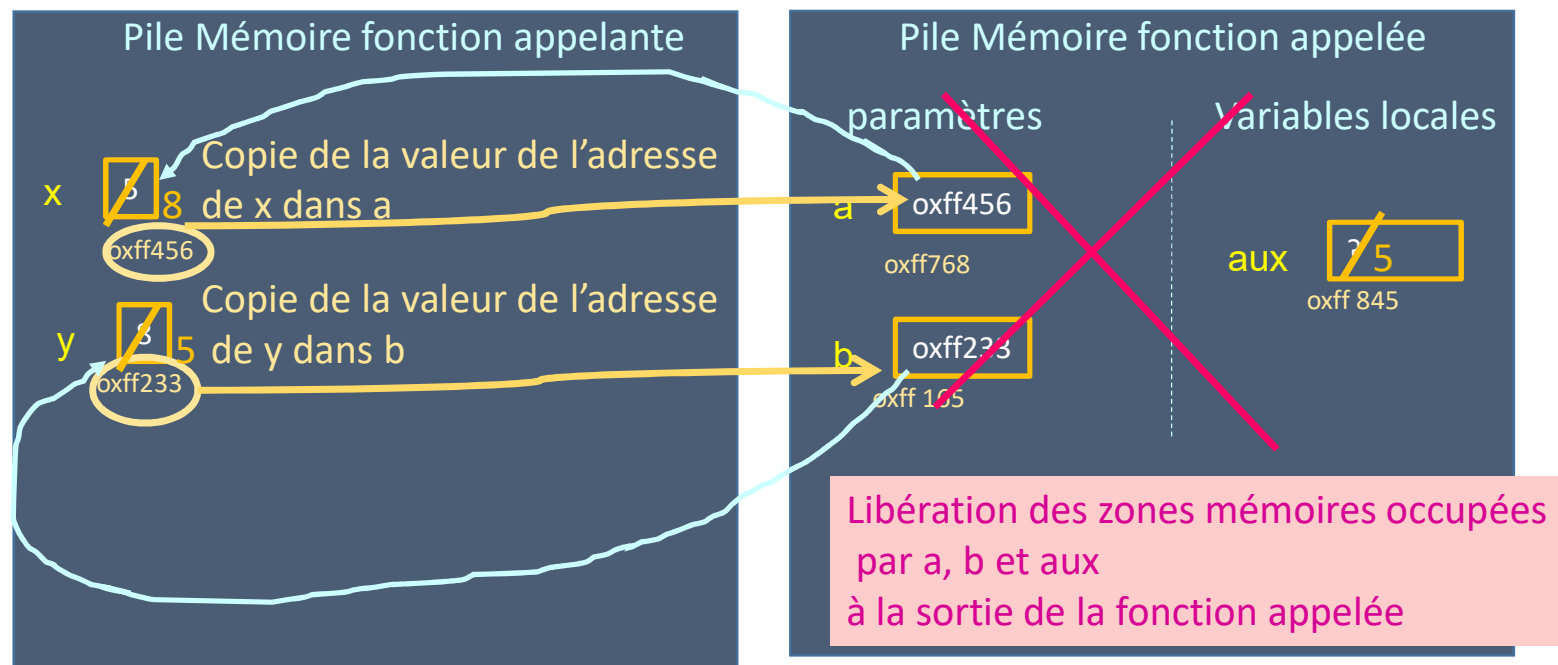
Utilisation de variables de type pointeur
pour stocker des adresses

Les adresses des paramètres effectifs
sont envoyées à la fonction echange

avant appel d'echange: 5 et 8
au debut d'echange: 5 et 8
à la fin d'echange: 8 et 5
après appel d'echange: 8 et 5

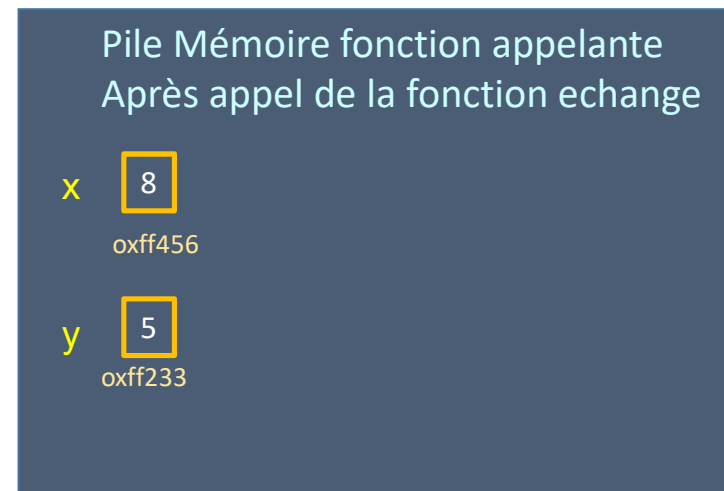
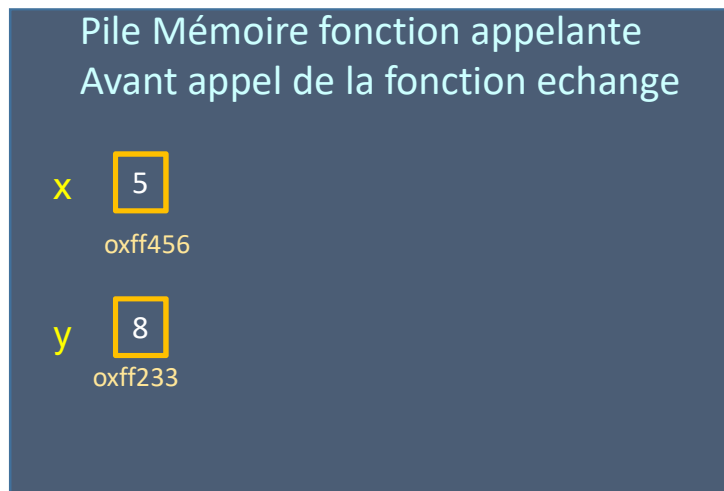
Fonctions

- Passage par adresse



Fonctions

- Passage par adresse
 - état de la mémoire avant et après l'appel de la fonction



Fonctions

- Passage de paramètres par valeur ou par adresse ?

Passage de Paramètres	Quand ?
par valeur	<ol style="list-style-type: none">1. les paramètres effectifs ne doivent pas être modifiés2. une seule valeur à modifier, la valeur modifiée est renvoyée comme valeur de retour de la fonction
par adresse	<ol style="list-style-type: none">1. les paramètres effectifs doivent être modifiés2. plusieurs valeurs doivent être « renvoyées » par la fonction ⇔ On passe autant de paramètres par adresse qu'on a de résultats à « renvoyer »3. pour passer un tableau en paramètres

Fonctions

- Récupérer plusieurs résultats à la sortie d'une fonction ?

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

int addSub (int a, int b, int *resSub){
    *resSub = a-b;
    return (a+b);
}

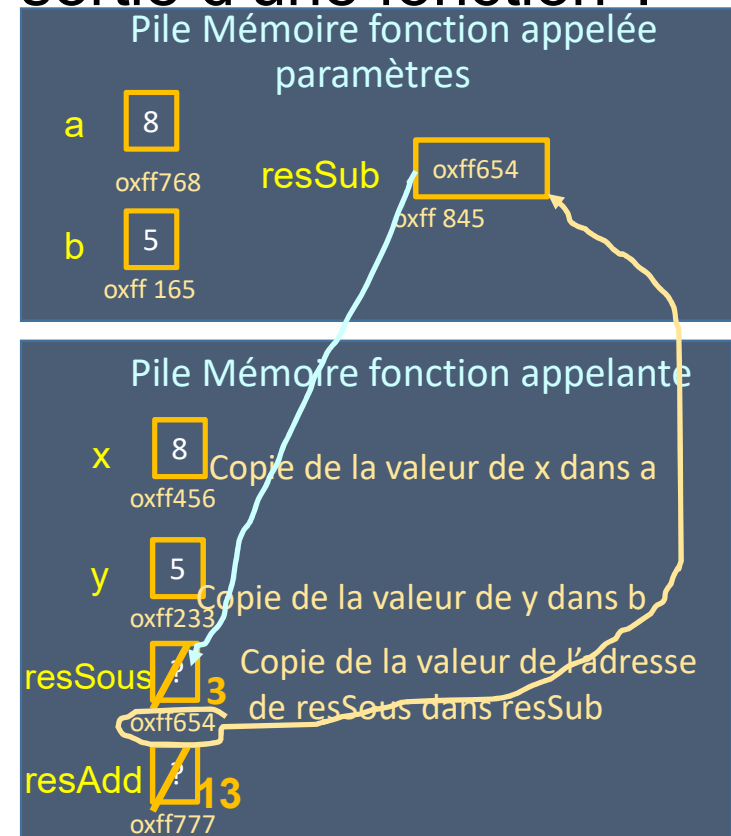
int main(){
    int x=8, y=5;
    int resAdd, resSous;
    resAdd= addSub(x,y,&resSous);

    printf("%d + %d = %d\n",x,y,resAdd );
    printf("%d - %d = %d\n",x,y,resSous );
    return EXIT_SUCCESS;
}
```

132

Properties TestSv

```
<terminated> TestSv
8 + 5 = 13
8 - 5 = 3
```



Fonctions

- Récupérer plusieurs résultats à la sortie d'une fonction ?

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

Pas de retour de valeur
void addSub (int a, int b, int *rAdd, int *rSub)
{
    *rSub = a-b;
    *rAdd = a+b;
}
```

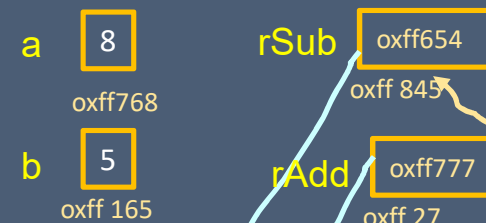
Renvoi des 2 résultats en utilisant le passage par adresse de 2 variables

```
int main() {
    Passage par valeur
    int x=8, y=5;
    int resAdd, resSous;
    addSub(x, y, &resAdd, &resSous);
    printf("%d + %d = %d\n", x, y, resAdd );
    printf("%d - %d = %d\n", x, y, resSous );
    return EXIT_SUCCESS;
}
```

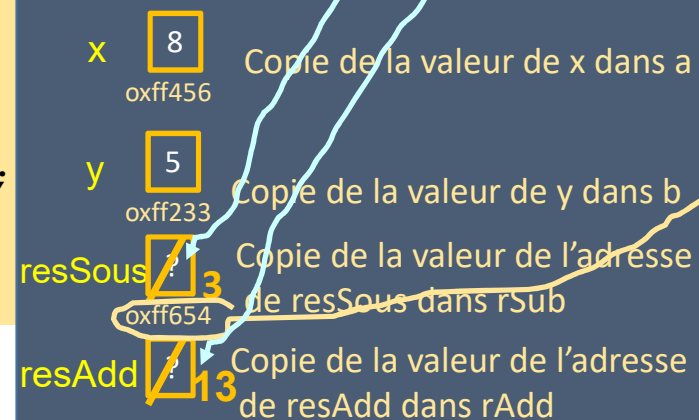
Passage par adresse

Properties P
<terminated> TestSv
8 + 5 = 13
8 - 5 = 3

Pile Mémoire fonction appelée paramètres



Pile Mémoire fonction appelante



Fonctions

- Passage de tableaux 1D en paramètres

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

#define NBMAX 100

void saisieTab(int t[], int nb){
    int i;
    for(i=0;i<nb;i++){
        printf("\ntab[%d]?", i);
        fflush(stdout);
        scanf("%d", &t[i]);
    }
}

void afficheTab(int t[], int nb){
    int i;
    printf("Affichage tableau: ");
    for(i=0;i<nb;i++){
        printf("%d\t", t[i]);
    }
    printf("\n");
}
```

Le nombre d'éléments de la dimension peut être omis dans la signature de la fonction

Nombre de cases effectivement remplies dans le tableau

Nombre de cases effectivement remplies dans le tableau

Fonctions

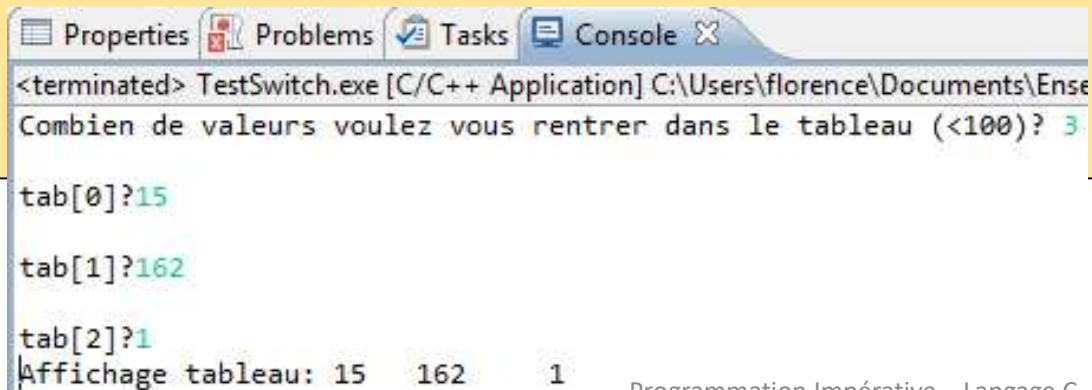
- Passage de tableaux 1D en paramètres

```
int main() {
    int tab[NBMAX], n;
    do{
        printf("Combien de valeurs dans le tableau(<=%d)? ", NBMAX);
        fflush(stdout);
        scanf("%d", &n);
    }while(n>NBMAX || n<1);
    saisieTab(tab, n);
    afficheTab(tab, n);
    return EXIT_SUCCESS;
}
```

tab ⇔ &tab[0]



Les tableaux sont toujours passés par adresse



```
<terminated> TestSwitch.exe [C/C++ Application] C:\Users\florence\Documents\Ense
Combien de valeurs voulez vous rentrer dans le tableau (<100)? 3
tab[0]?15
tab[1]?162
tab[2]?1
Affichage tableau: 15 162 1
```

Fonctions

- Passage de tableaux 2D en paramètres

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

```
#define NB_LIG 100
#define NB_COL 100
```

Le nombre d'éléments de la dimension ligne peut être omis dans la signature de la fonction

```
void saisieTab2D(int t[][NB_COL], int nbL, int nbC){
    int i,j;
    for(i=0;i<nbL;i++){
        for(j=0;j<nbC;j++){
            printf("\ntab[%d][%d]?",i,j);
            fflush(stdout);
            scanf("%d", &t[i][j]);
        }
    }
}
```

Nombre de lignes et de colonnes effectivement remplies dans le tableau 2D

```
void afficheTab2D(int t[][NB_COL], int nbL, int nbC){
    int i,j;
    printf("Affichage tableau 2D: \n");
    for(i=0;i<nbL;i++){
        for(j=0;j<nbC;j++){
            printf("%d\t",t[i][j]);
        }
        printf("\n");
    }
}
```

Fonctions

- Passage de tableaux 2D en paramètres

```
int main(){
    int tab2D[NB_LIG][NB_COL], nLignes, nColonnes;
    do{
        printf("Combien de lignes dans le tableau(<%d)? ", NB_LIG);
        fflush(stdout);
        scanf("%d",&nLignes);
    }while(nLignes>NB_LIG||nLignes<1);
    do{
        printf("Combien de colonnes dans le tableau(<%d)? ", NB_COL);
        fflush(stdout);
        scanf("%d",&nColonnes);
    }while(nColonnes>NB_COL||nColonnes<1);

    saisieTab2D(tab2D,nLignes, nColonnes);
    afficheTab2D(tab2D,nLignes, nColonnes);

    return EXIT_SUCCESS;
}
```

tab2D ⇔ &tab2D[0][0]



Les tableaux sont toujours
passés par adresse

Fonctions

- Récursivité
 - En C, les fonctions peuvent être utilisées de façon récursive
- ⇔ possible car la gestion des appels de fonction est assurée par une pile
- 2 points importants
 - Test d'arrêt de la récursivité placé au début de la fonction
 - Appel récursif à déterminer en fonction de la définition récursive de la fonction

Fonctions

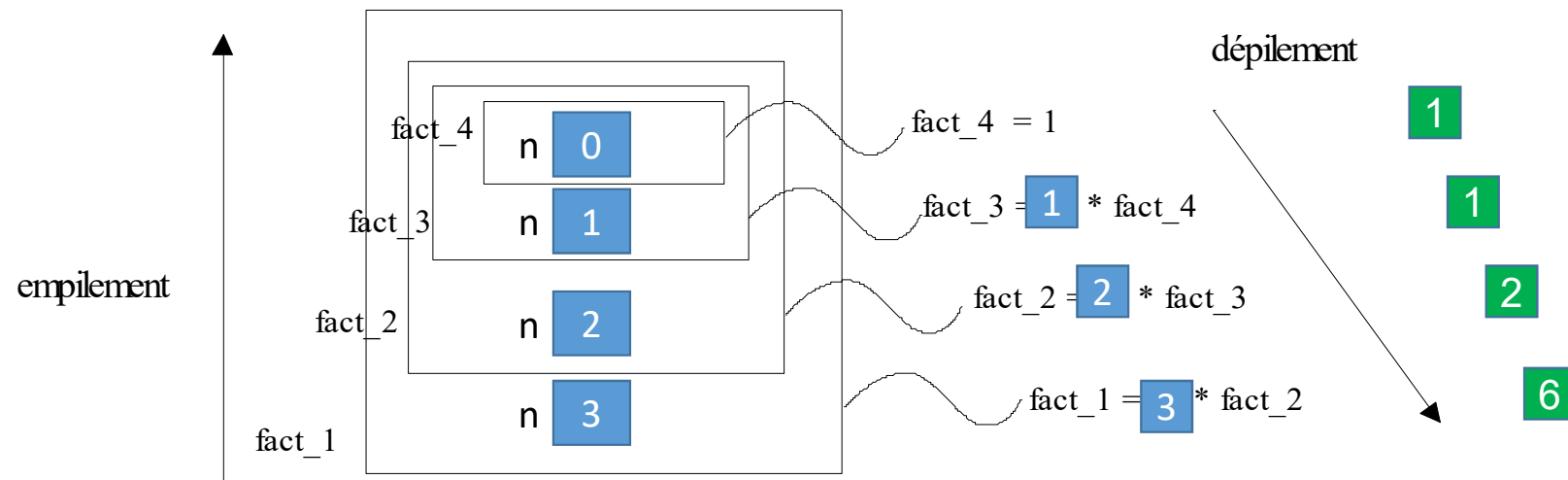
- Réversivité - Illustration

Définition mathématique	Implémentation récursive en C
<p>Fonction factorielle</p> <p>Définition itérative</p> $0! = 1$ $n! = n * (n - 1) * (n - 2) * \dots * 2 * 1$ <p>Ou</p> <p>Définition récursive</p> $0! = 1$ $n! = n * (n-1)!$	<pre>int factorielle(int n){ if (n == 0) Test d'arrêt de la return 1; récursion return(n * factorielle (n-1)); } int main(){ int nb=3; printf(" %d != %d\n", nb , factorielle(nb)); return 1; }</pre> <p>Appel récursif</p>

Fonctions

- Récurtivité - Illustration

fact_i : prologue de la fonction factorielle au i^{ème} appel



Exemple d'exécution de la fonction factorielle avec au départ $n=3$