

Introduction à C++

Applications mobiles et objets connectés 420-W48-SF

Objectifs

- Syntaxe de base
- Type de base, références, pointeurs
- Fonction
 - Déclaration
 - Définition
- Fichiers d'entête

Syntaxe – Instructions et blocs

- Les **instructions** sont séparées par des « ; »
 - `int unNombre = 0;`
- Un **bloc** permet de grouper des **plusieurs instructions** qui s'exécutent dans un même **contexte**.
- Pour définir un bloc, on l'encadre par une paire d'accolades « { *instruction*; ...; *instruction*; } »

```
{  
    int unNombre = 0;  
    int unNombrePlus1 = unNombre + 1;  
}
```

Syntaxe – Alternatives – Si ... (if)

```
if (<predicat>)
    <instruction> | <bLoc>
[else
    <instruction> | <bLoc>]
```

Donc formes possibles :

```
if (<predicat>)
    <instruction> | <bLoc>
```

```
if (<predicat>)
    <instruction> | <bLoc>
else
    <instruction> | <bLoc>
```

Syntaxe – Alternatives – SI ... (if) – Exemples

```
bool estNombreNegatif = false;  
if (nombre < 0)  
{  
    estNombreNegatif = true;  
}
```

```
bool estNombreNegatif = nombre < 0;
```

```
bool estNombrePair;  
if (nombre % 2 == 0)  
{  
    estNombrePair = true;  
}  
else  
{  
    estNombrePair = false;  
}
```

```
bool estNombrePair = nombre % 2 == 0;
```

Normes : toujours utiliser des blocs pour les alternatives et les répétitions

Syntaxe – Alternatives – SELON (switch)

```
switch (expression)
{
    case <valeur constante>:
        [<instructions> | <bloc>]*
        [break;]
    [case <valeur constante>:
        [<instructions> | <bloc>]*
        [break;]
    [default:
        [<instructions> | <bloc>]*
        [break;]]
}
```

break : cette instruction permet de sortir du bloc courant, sinon le programme continue à exécuter les instructions suivantes même si elles ne sont pas dans le même cas

Normes : toujours mettre un cas par défaut comme dernier cas, même s'il est vide

Syntaxe – Alternatives – SELON (switch) – Exemple

```
switch (c)
{
    case 'A':
        maja = maja + 1;
        break;
    case 'a':
        mina = mina + 1;
        break;
    default:
        nona = nona + 1;
}
```

Exemple d'interprétation :

```
if (c == 'A')
{
    maja = maja + 1;
}
else if (c == 'a')
{ // *
    mina = mina + 1;
}
else
{
    nona = nona + 1;
}
```

Normes : cours pas de début de bloc pour le if après le sinon, mais obligatoire pour le alors

Syntaxe – Alternatives – SELON (switch) – Exemple sans break

```
switch (c)
{
    case 'A':
        maja = maja + 1;
    case 'a':
        mina = mina + 1;
    default:
        nona = nona + 1;
}
```



Exemple d'interprétation :

```
if (c == 'A')
{
    maja = maja + 1;
    mina = mina + 1;
    nona = nona + 1;
}
else if (c == 'a')
{
    mina = mina + 1;
    nona = nona + 1;
}
else
{
    nona = nona + 1;
}
```

Syntaxe – Répétition – TANT QUE ... (WHILE)

But : Exécuter les instructions **tant que** le prédicat est vrai

```
while (<predicat>
       <instruction> | <bloc>
```

```
int n = 1;
int fact = 1;

while (n < 10) {
    fact *= n;
    n = n + 1;
}
```

Syntaxe – Répétition – FAIRE ... TANT QUE (DO ... While)

But : Exécuter les instructions **une fois et le refaire** tant que le prédicat est vrai

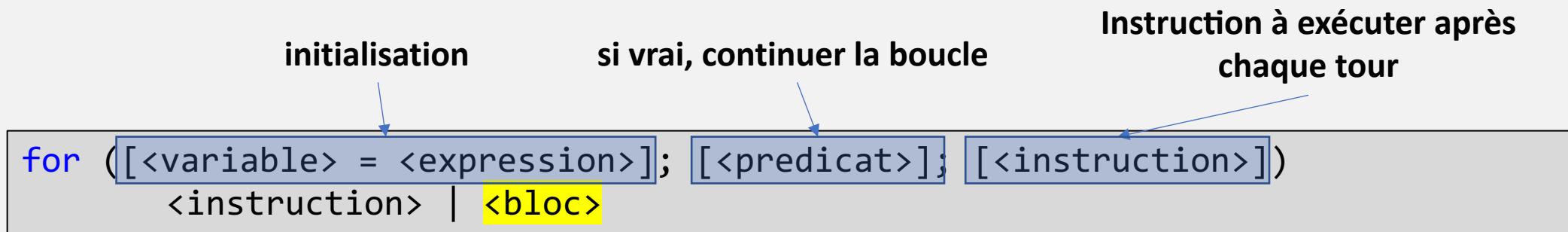
```
do
    <instruction> | <bloc>
while (<predicat>);
```

```
int n = 1;
int fact = 1;

do {
    fact *= n;
    n = n + 1;
} while (n < 10);
```

Syntaxe – Répétition – POUR ... (FOR)

But : Exécuter les instructions **tant que** le prédicat est vrai avec une instruction de déclaration / initialisation et instruction d'incrément



```
int fact = 1;

for (int n = 1; n < 10; ++n)
{
    fact *= n;
}
```

Types de base – Arduino Uno

Type	Octets	Min	Max	Remarque
bool	1	false, true		
byte	1	0	255	Non signé
char	1	-128	127	Pour un non signé, utiliser byte plutôt Utiliser char pour des caractères
float, double	4	-3.4028235E+38	3.4028235E+38	Double équivalent à float
int, short	2	-32.768	32.767	
long	4	-2.147.483.648	2.147.483.647	La valeur dans le code doit se terminer par L Ex. : 300000L
(string)	Nb caractères + 1	"", "Bonjour à tous !"		Se décompose en tableau de char. Le dernier caractère est null ('\0'). L'objet correspondant est de type « <u>String</u> »

unsigned <type> : rend le type non signé pour les entiers

Types de base – Entiers

- Pour simplifier certains codes, on peut utiliser des constantes entières écrites dans les bases 2 (binaire), 8 (octale) et 16 (hexadécimale)

```
int valeurAPartirDecimal = 254;
int valeurAPartirBinaire = 0b11111110;
int valeurAPartirOctal = 0376;
int valeurAPartirHexadecimal = 0xFE;
```

Opérateurs de comparaisons

- Syntaxe : <expression> <opérateur> <expression> -> booléen
- Où <opérateur> :
 - == | != : égale, différent
 - < | <= : inférieur, inférieur ou égale
 - > | >= : supérieur, supérieur ou égale
- Exemple : $42 > 13 \Leftrightarrow \text{true}$

Opérateurs arithmétiques sur les entiers / réels

- Syntaxe : <expression> <opérateur> <expression> -> entier / réel
- Où <opérateur> :
 - * | / : multiplication
 - + | - : addition, soustraction
- Exemple : $42.0 / 10.0 \Leftrightarrow 4.2$
- La multiplication, la division sont prioritaires sur l'addition et la soustraction

Opérateurs sur les « String » (Arduino)

```
String maChaine = " Bonjour à tous ";
int longeurChaine = maChaine.length();
char premierCaractere = maChaine[0];

String autreInitialisation1 = String('a');
String autreInitialisation2 = String("Bonjour à tous");
String autreInitialisation3 = String(autreInitialisation2 + " !");

maChaine.trim(); // String("Bonjour à tous")
maChaine.toLowerCase(); // String("bonjour à tous")    !!!String d'Arduino est mutable !!!
maChaine.toUpperCase(); // String("BONJOUR À TOUS")

int valeurEntiere = String("123").toInt();
int valeurFloatante = String("123.4").toFloat();

bool chaineEgales = maChaine.equals(String("bonjour"));
bool chaineEgales2 = maChaine == String("bonjour");
bool chaineEgales3 = maChaine == "bonjour";

const char *chaineEnCPointeur = maChaine.c_str();
```

<https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/>

<https://www.arduino.cc/en/Tutorial/BuiltInExamples#8-strings>

Manipulation bit à bit

Opérateur	Symbole C++	Utilisation	Opération réel
décalage à gauche	<code><<</code>	<code>x << y</code>	Tous les bits de <code>x</code> sont décalés de <code>y</code> positions vers la gauche
décalage à droite	<code>>></code>	<code>x >> y</code>	Tous les bits de <code>x</code> sont décalés de <code>y</code> positions vers la droite
NON bit à bit	<code>~</code>	<code>~x</code>	Tous les bits de <code>x</code> sont inversés
ET bit à bit (AND)	<code>&</code>	<code>x & y</code>	Chaque bit de <code>x</code> AND chaque bit de <code>y</code>
OU bit à bit (OR)	<code> </code>	<code>x y</code>	Chaque bit de <code>x</code> OR chaque bit de <code>y</code>

Exemple de manipulation bit à bit

- En informatique, généralement, une couleur est représentée par trois composantes dans l'espace RVB : rouge, vert et bleu
- Chaque composante a une valeur variant de 0 à 255, donc codée sur 8 bits (1 octet)
- Nous avons besoin de $8 * 3 = 24$ bits

Exemple de manipulation bit à bit

- Une façon de représenter une couleur est donc d'utiliser un type de variable ayant au moins 24 bits
- Un entier représenté avec un long en à 32.
- Une façon de représenter la couleur est donc de coder les 3 composantes dans un entier
- On utilise aussi souvent les 8 bits restant pour coder la composante alpha, qui correspond à la transparence
- Ex : AAAA AAAA **RRRR RRRR** **VVVV VVVV** **BBBB BBBB**

Exemple de manipulation bit à bit

- Codage d'une couleur (alpha, rouge, vert, bleu respectivement égale à valeur composante alpha, rouge, vert et bleu)
- `long maCouleur = (alpha << 24) | (rouge << 16) | (vert << 8) | bleu;`

```
AAAAA AAAA << 24 == AAAAA AAAA 0000 0000 0000 0000 0000 0000 0000 0000  
RRRR RRRR << 16 == 0000 0000 RRRR RRRR 0000 0000 0000 0000 0000 0000  
VVVV VVVV << 8 == 0000 0000 0000 0000 VVVV VVVV 0000 0000 0000 0000  
BBBB BBBB      == 0000 0000 0000 0000 0000 0000 BBBB BBBB
```

Exemple de manipulation bit à bit

- Codage d'une couleur (alpha, rouge, vert, bleu respectivement égale à valeur composante alpha, rouge, vert et bleu)
- `long maCouleur = (alpha << 24) | (rouge << 16) | (vert << 8) | bleu;`

AAAAA	AAAAA	0000	0000	0000	0000	0000	0000	0000
	0000	0000	RRRR	RRRR	0000	0000	0000	0000
	0000	0000	0000	0000	VVVV	VVVV	0000	0000
	0000	0000	0000	0000	0000	0000	BBBB	BBBB
==	AAAAA	AAAAA	RRRR	RRRR	VVVV	VVVV	BBBB	BBBB

Exemple de manipulation bit à bit

- Accès aux composantes :

- `int vAlpha = (maCouleur >> 24) & 0xFF;`
- `int vRouge = (maCouleur >> 16) & 0xFF;`
- `int vVert = (maCouleur >> 8) & 0xFF;`
- `int vBleu = maCouleur & 0xFF;`

```
AAAAA AAAA RRRR RRRR VVVV VVVV BBBB BBBB >> 24
== 0000 0000 0000 0000 0000 0000 AAAA AAAA

0000 0000 0000 0000 0000 0000 AAAA AAAA
& 0000 0000 0000 0000 0000 0000 1111 1111      (255 ou 0xFF)
== 0000 0000 0000 0000 0000 0000 AAAA AAAA
```

Exemple de manipulation bit à bit

- Acces aux composantes :
 - int vAlpha = (maCouleur >> 24) & 0xFF;
 - int vRouge = (maCouleur >> 16) & 0xFF;
 - int vVert = (maCouleur >> 8) & 0xFF;
 - int vBleu = maCouleur & 0xFF;

```
AAAAA AAAA  RRRR  RRRR  VVVV  VVVV  BBBB  BBBB  >> 16
== 0000  0000  0000  0000  AAAA  AAAA  RRRR  RRRR

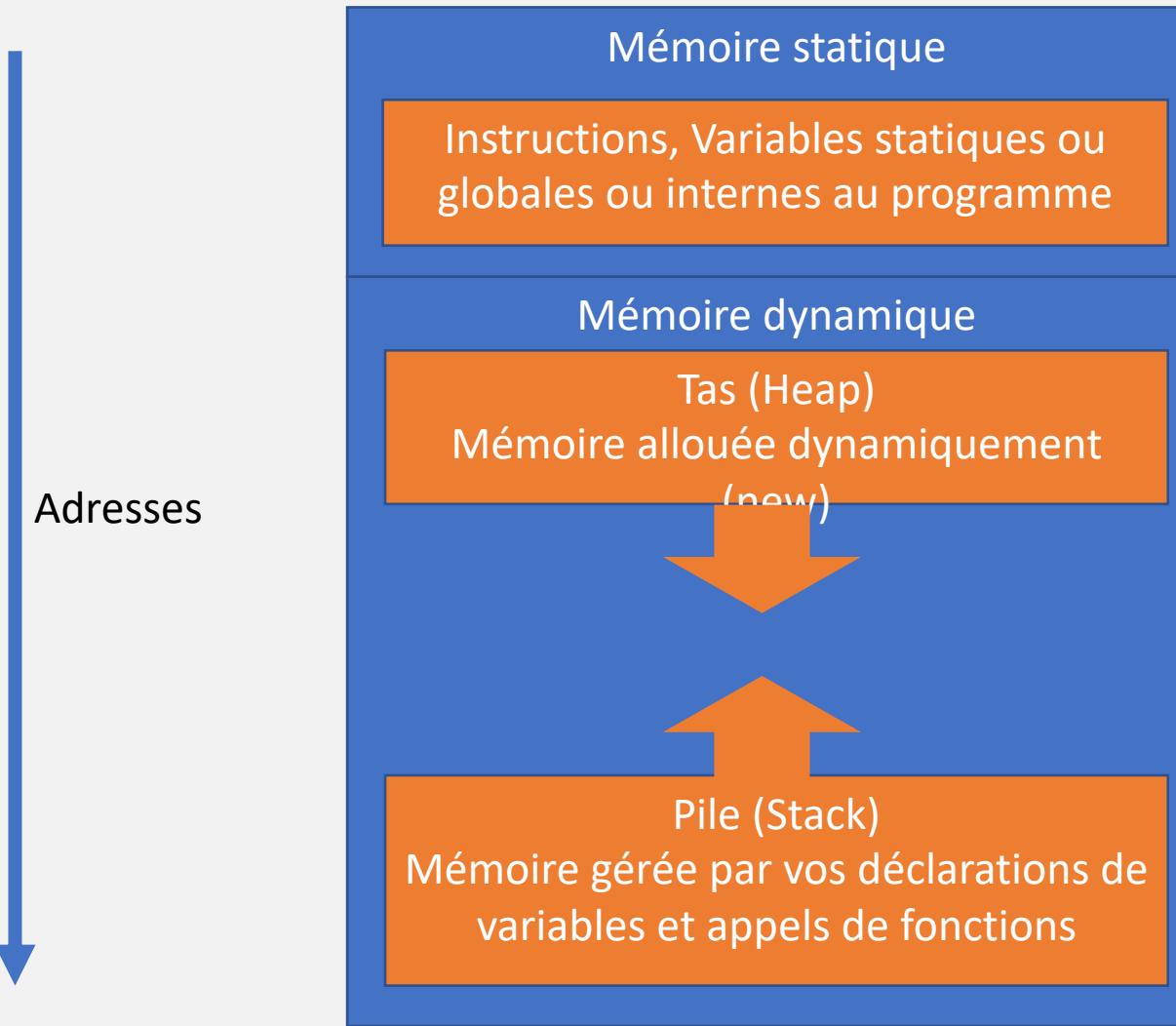
          0000  0000  0000  0000  AAAA  AAAA  RRRR  RRRR
& 0000  0000  0000  0000  0000  0000  1111  1111      (0xFF ou 255)
== 0000  0000  0000  0000  0000  0000  RRRR  RRRR
```

Tableaux

- Syntaxes :
 - <type> <nomVariable>[<capacité>]
 - <type> <nomVariable>[] = { valeur1, valeur2, [...], valeurN };
- Exemple :

```
int tableau1[3];
int tableau2[] = { 13, 42, 12};
```

Pointeurs – organisation de la mémoire (Schématique)



Pointeurs – Mémoire

- Un pointeur est un type de données qui contient une adresse mémoire

Address	Content	Name	Type	Value
90000000	00	iii	int	000000FF (255 ₁₀)
90000001	00			
90000002	00			
90000003	FF	sss	short	FFFF (-1 ₁₀)
90000004	FF			
90000005	FF	ddd	double	1FFFFFFFFFFFFFFF (4.4501477170144023E-308 ₁₀)
90000006	1F			
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90			
9000000F	00	ptr	int*	90000000
90000010	00			
90000011	00			

Revenons au C++ - Déclaration

- La déclaration utilise *

- `int* p;`
- `int i = 0;`

- & est l'opérateur pour obtenir l'adresse d'une zone mémoire

- `p = &i;`

- * est l'opérateur de déréférencement et permet d'accéder à la valeur qui se trouve à une adresse :

- `*p = 15;`

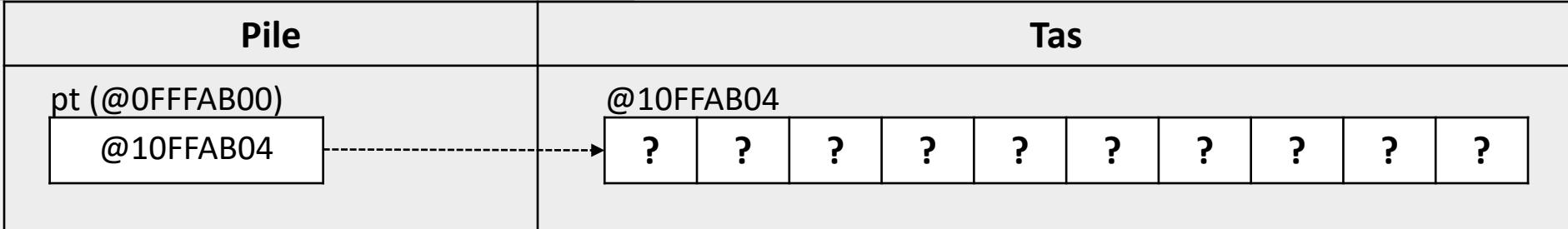
Pile	Tas
<code>p (@0FFFAB00) i (@0FFFAB04)</code> ? 0	

Pile	Tas
<code>p (@0FFFAB00) i (@0FFFAB04)</code> @0FFFAB04 → 0	

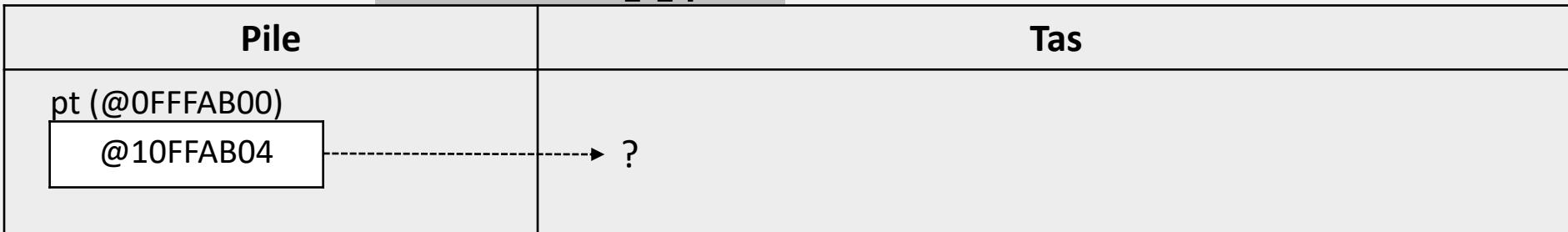
Pile	Tas
<code>p (@0FFFAB00) i (@0FFFAB04)</code> @0FFFAB04 → 15	

Allocation dynamique de mémoire

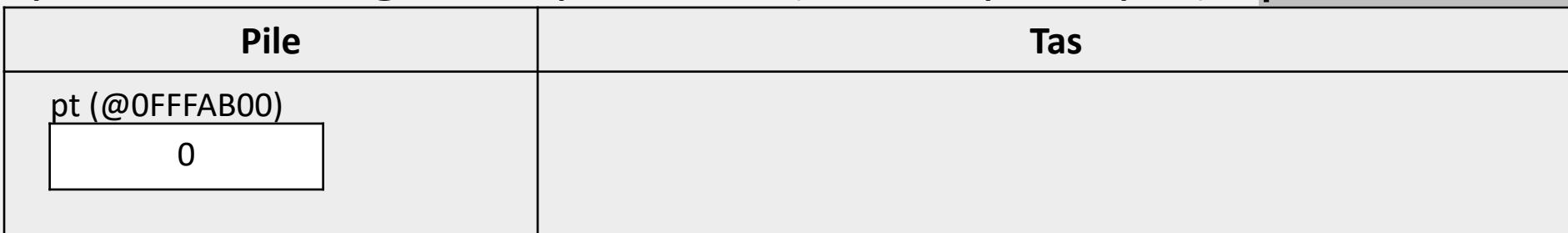
- `int* pt = new int[10];`



- Libérer la mémoire `delete []pt;`



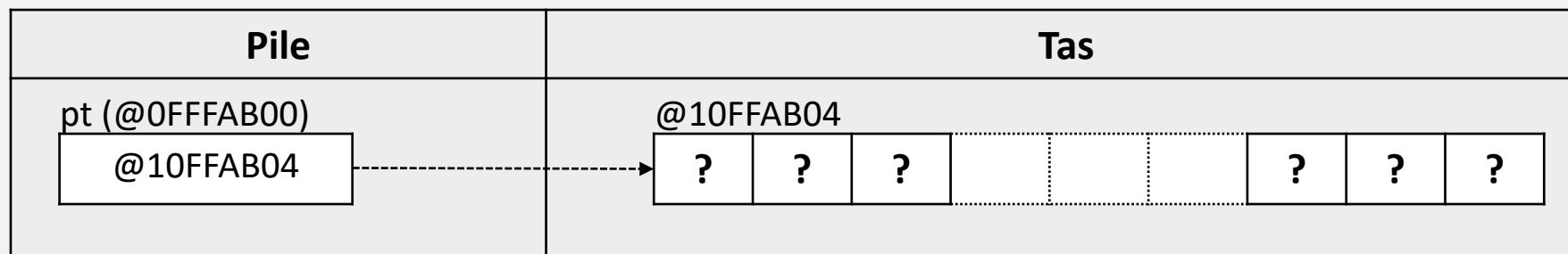
- `NULL` peut être assigné au pointeur (bonne pratique) : `pt = NULL;`



Fuite de mémoire (memory leak)

```
void maFonction()
{
    int *pt = new int[100];
    // sans delete
}

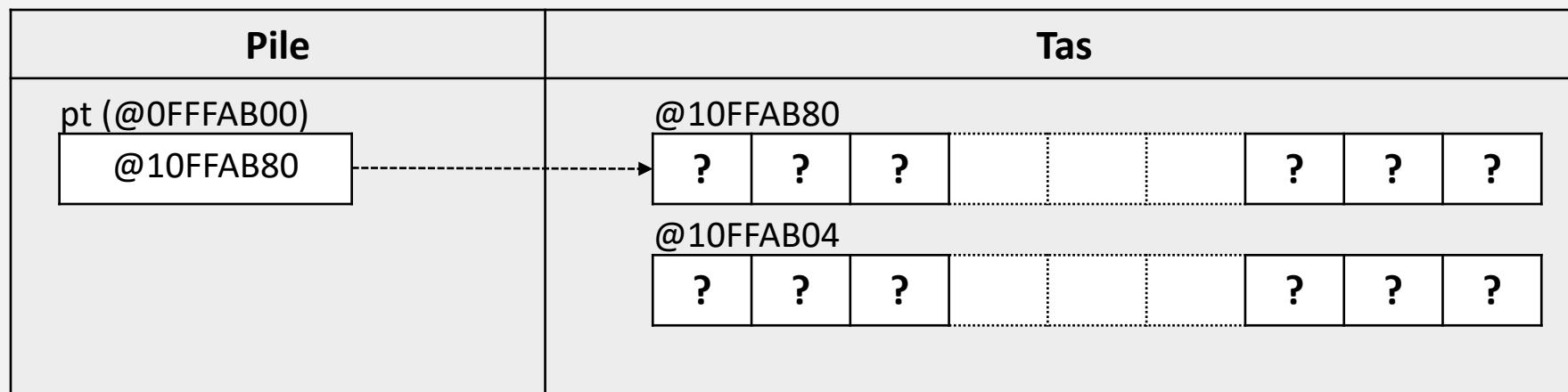
for(;;)
{
    maFonction();
}
```



Fuite de mémoire (memory leak)

```
void maFonction()
{
    int *pt = new int[100];
    // sans delete
}

for(;;)
{
    maFonction();
}
```



Fuite de mémoire (memory leak)

```
void maFonction()
{
    int *pt = new int[100];
    // sans delete
}

for(;;)
{
    maFonction();
}
```

Pile	Tas
pt (@0FFFAB00) ?	@10FFAB00 ? ? ? ? ? ? @10FFAB04 ? ? ? ? ? ? @10FFAC04 ? ? ? ? ? ? ...

=> Out of memory

Références en C++

- Une référence permet de référencer une zone mémoire
- Pour déclarer une variable de type référence d'un certain type, on utilise « & »
- La référence est invariable et ne peut être modifiée une fois déclarée
- Une fois déclarée, elle se comporte comme les autres variables : pas besoin d'artifice pour y accéder
- Elle agit comme un **synonyme**, comme un autre nom

```
int valeur = 42;
int &refValeur = valeur;
```

Pile	Tas
valeur, refValeur (@0FFFAB00) 42	

Fonctions – Déclaration / définition

- Déclaration :

```
<type retour> <nom de la fonction>([<type paramètre> <nomParamètre>]*);
```

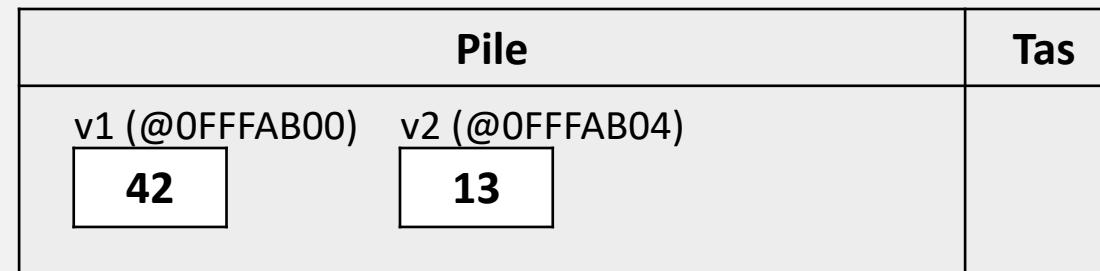
- Définition / implantation

```
<type retour> <nom de la fonction>([<type paramètre> <nomParamètre>]*) {  
    // [...]  
    [return <expression du type de retour>;]  
}
```

- Les paramètres peuvent être des valeurs, pointeurs (i.e. valeur), tableaux (i.e. pointeur, donc valeur) et des références

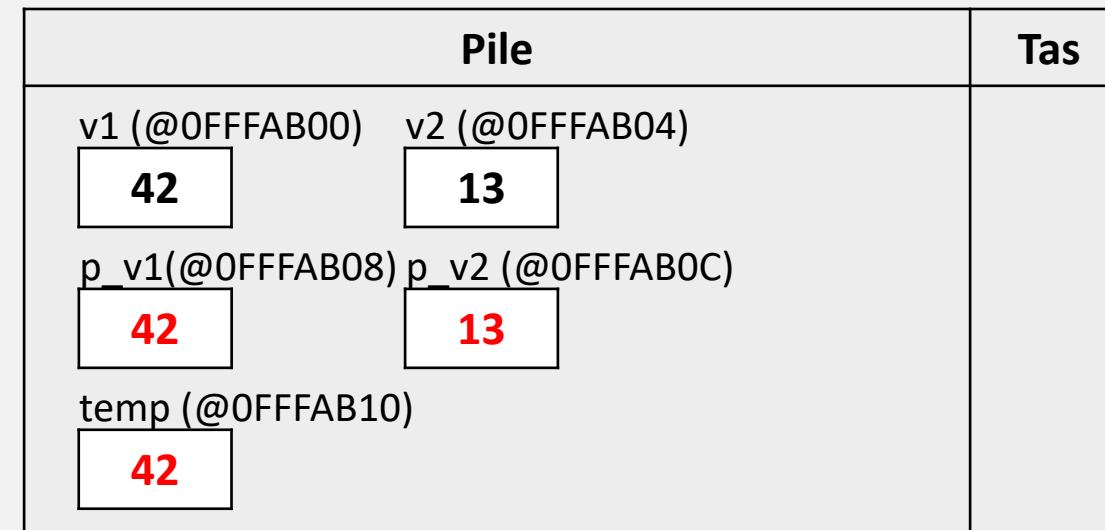
Fonctions – Exemples – Passage par valeur

```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```



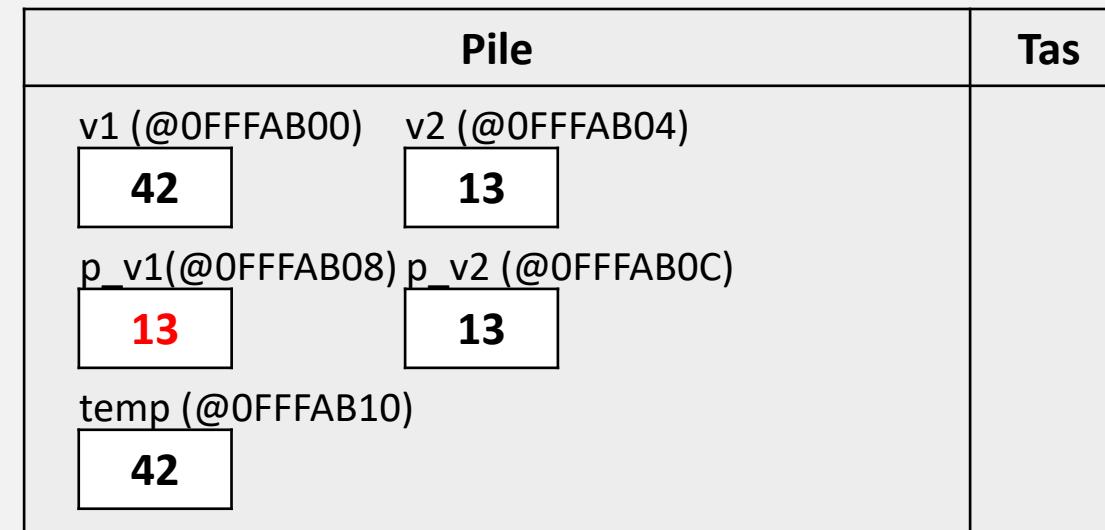
Fonctions – Exemples – Passage par valeur

```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```



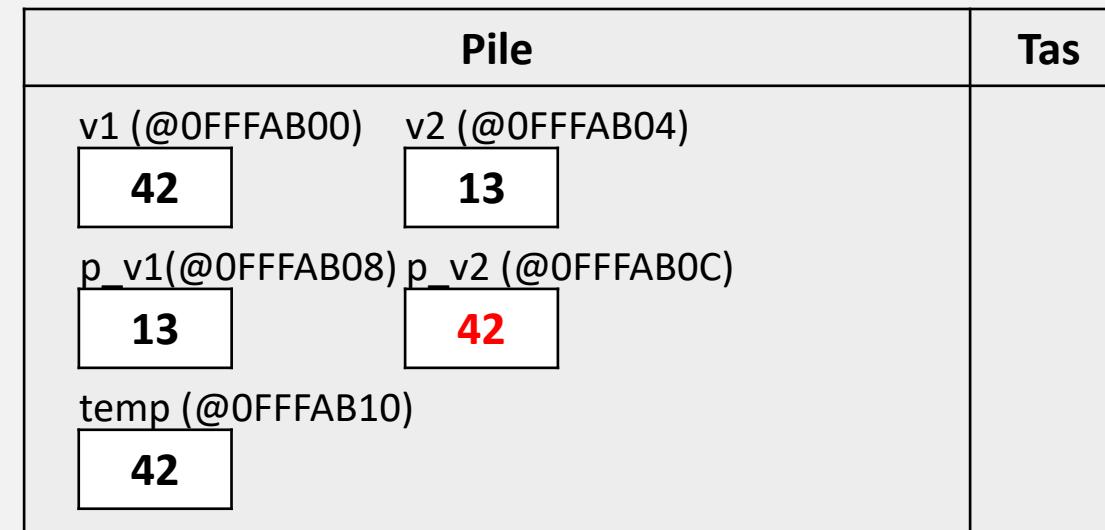
Fonctions – Exemples – Passage par valeur

```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```



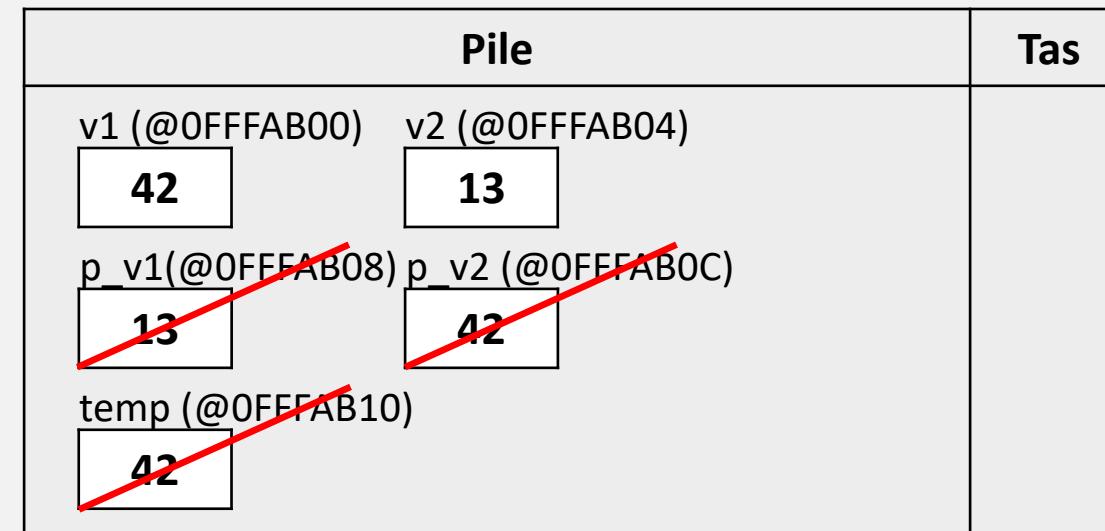
Fonctions – Exemples – Passage par valeur

```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```



Fonctions – Exemples – Passage par valeur

```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```



Fonctions – Exemples – Passage par valeur

```
void echanger(int p_v1, int p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echanger(v1, v2);  
// ...
```

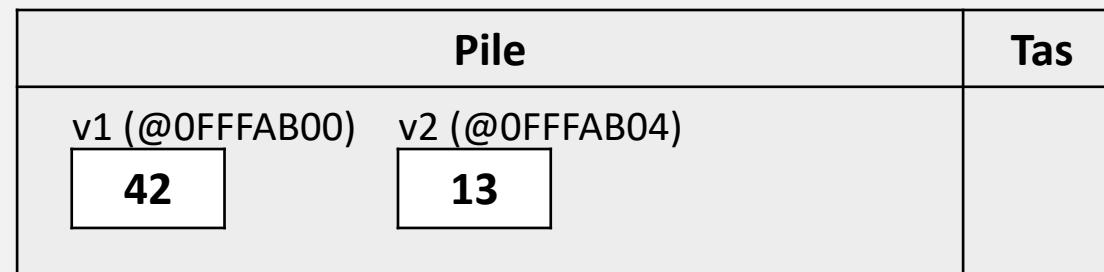


Pile	Tas
v1 (@0FFFAB00) 42 v2 (@0FFFAB04) 13	

v1 et v2 n'ont pas été modifié !

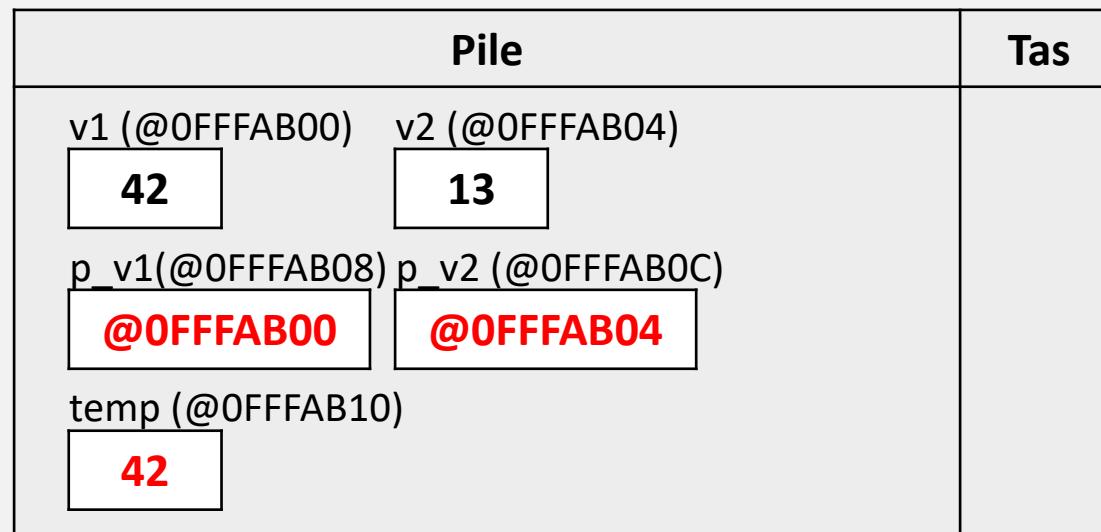
Fonctions – Exemples – Passage par pointeur

```
void echangerPointeurs(int *p_v1, int *p_v2) {  
    int temp = *p_v1;  
    *p_v1 = *p_v2;  
    *p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerPointeurs(&v1, &v2);  
// ...
```



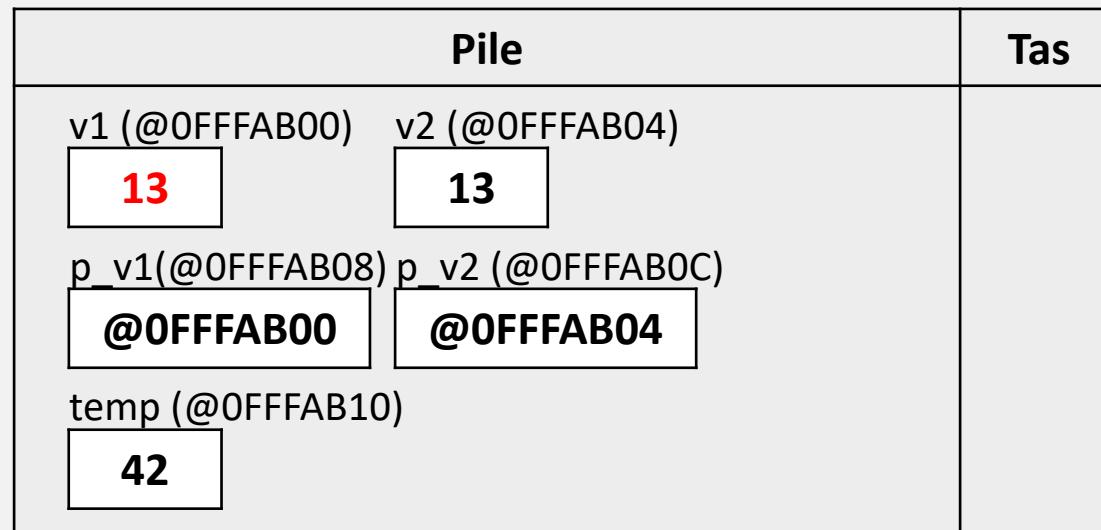
Fonctions – Exemples – Passage par pointeur

```
void echangerPointeurs(int *p_v1, int *p_v2) {  
    int temp = *p_v1;  
    *p_v1 = *p_v2;  
    *p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerPointeurs(&v1, &v2);  
// ...
```



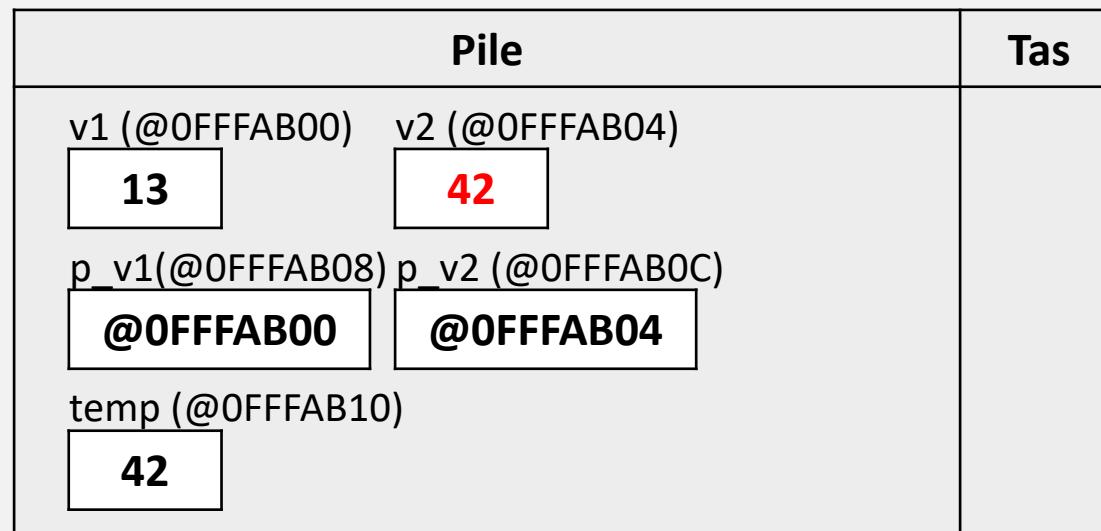
Fonctions – Exemples – Passage par pointeur

```
void echangerPointeurs(int *p_v1, int *p_v2) {  
    int temp = *p_v1;  
    *p_v1 = *p_v2;  
    *p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerPointeurs(&v1, &v2);  
// ...
```



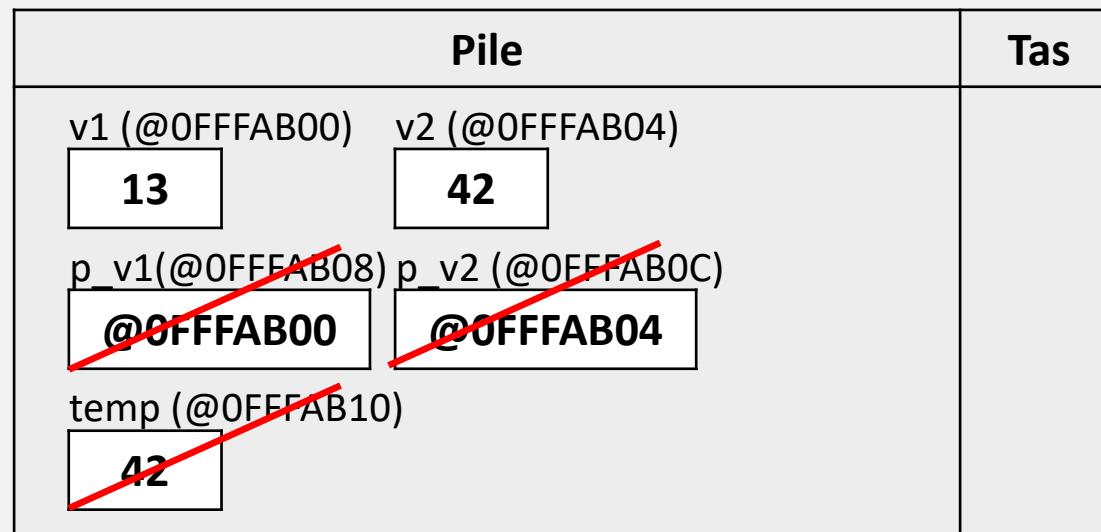
Fonctions – Exemples – Passage par pointeur

```
void echangerPointeurs(int *p_v1, int *p_v2) {  
    int temp = *p_v1;  
    *p_v1 = *p_v2;  
    *p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerPointeurs(&v1, &v2);  
// ...
```



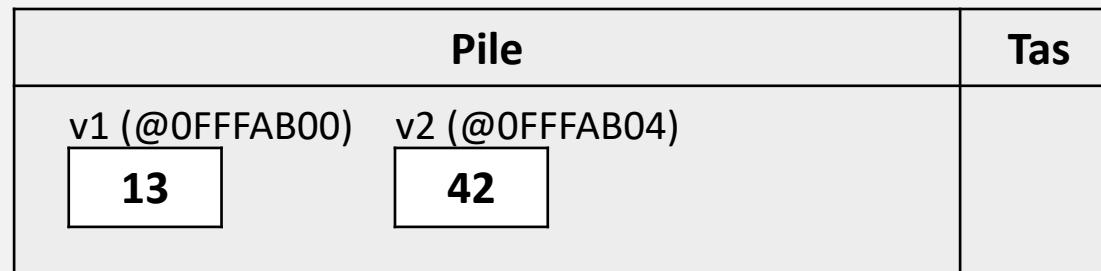
Fonctions – Exemples – Passage par pointeur

```
void echangerPointeurs(int *p_v1, int *p_v2) {  
    int temp = *p_v1;  
    *p_v1 = *p_v2;  
    *p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerPointeurs(&v1, &v2);  
// ...
```



Fonctions – Exemples – Passage par pointeur

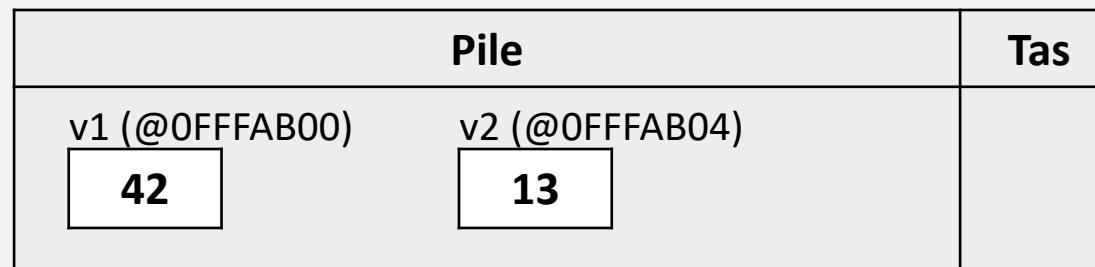
```
void echangerPointeurs(int *p_v1, int *p_v2) {  
    int temp = *p_v1;  
    *p_v1 = *p_v2;  
    *p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerPointeurs(&v1, &v2);  
// ...
```



v1 et v2 modifiées !

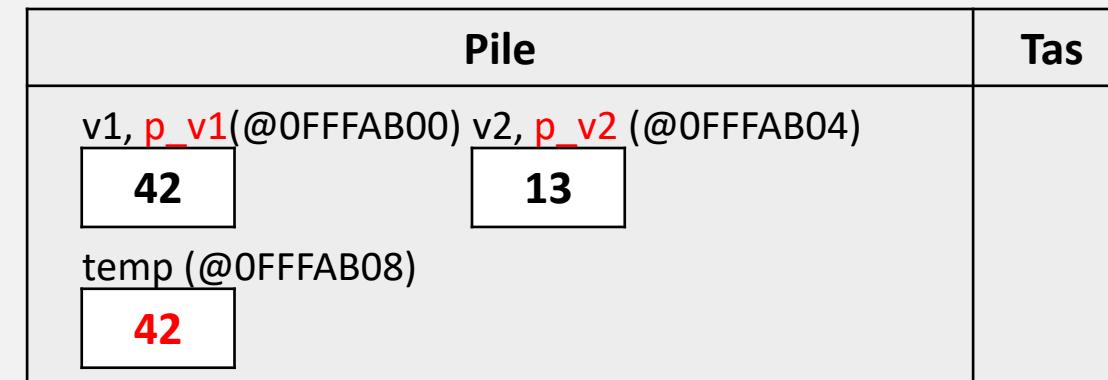
Fonctions – Exemples – Passage par référence

```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```



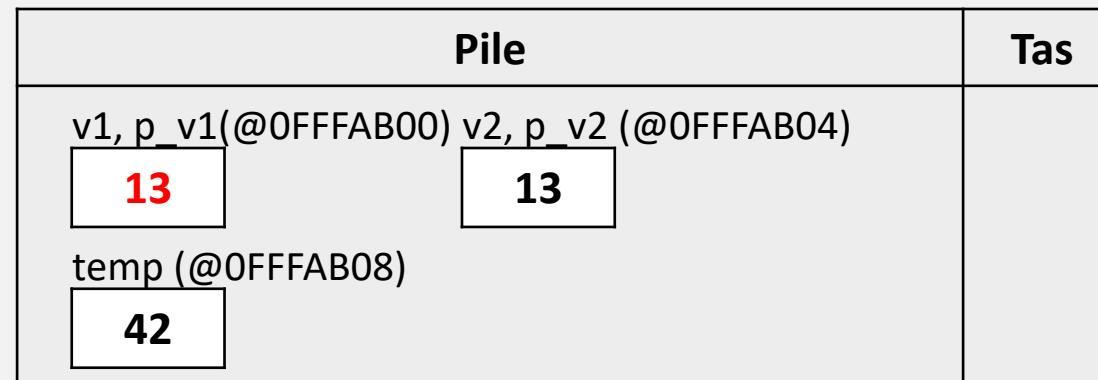
Fonctions – Exemples – Passage par référence

```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```



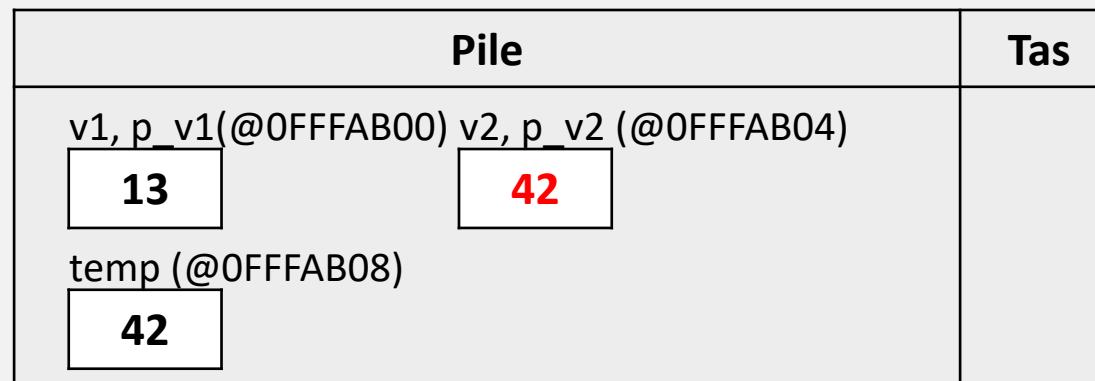
Fonctions – Exemples – Passage par référence

```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```



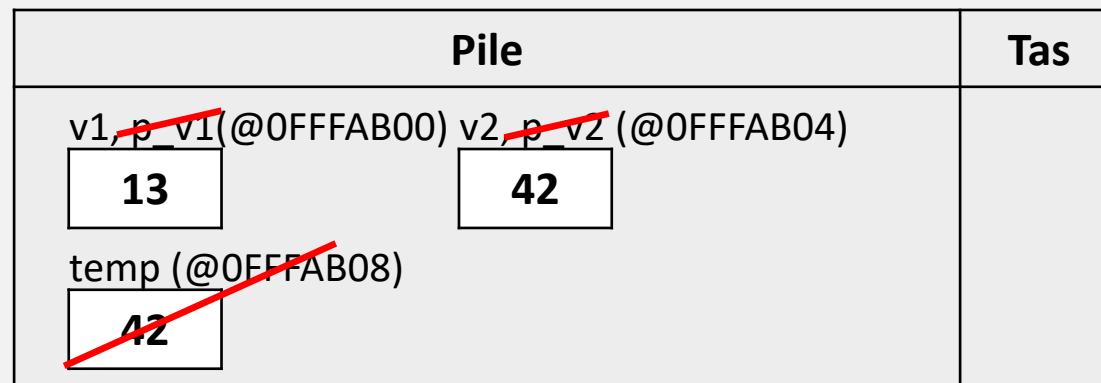
Fonctions – Exemples – Passage par référence

```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```



Fonctions – Exemples – Passage par référence

```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```



Fonctions – Exemples – Passage par référence

```
void echangerReferences(int &p_v1, int &p_v2) {  
    int temp = p_v1;  
    p_v1 = p_v2;  
    p_v2 = temp;  
}  
  
int v1 = 42;  
int v2 = 13;  
echangerReferences(v1, v2);  
// ...
```



Pile	Tas
v1 (@0FFFAB00) 13	v2 (@0FFFAB04) 42

Fichiers d'en-tête

- Fichiers de déclarations ou d'entêtes (.h)
 - Ces fichiers servent à déclarer des fonctions ou des classes
 - Il ne contient donc généralement pas de définition (c.à.d.. de corps de fonction / méthode)
- La première ligne doit contenir la directive de précompilation « `#pragma once` » (Version simple)
- Des fichiers d'entête peuvent être inclus dans un autre fichier d'entête ou un fichier de définition avec la directive de pré-compilation « `#include <nomFichier>` »
- `<nomFichier>` :
 - Si le nom de fichier est encadré par « < » et « > » : le fichier est cherché dans les répertoires configurés au moment de la compilation
 - Si le nom de fichier est encadré par « " » : le fichier est cherché à partir du fichier courant (chemin relatif)

Déclaration / Définition – Exemple

Util.h

```
#pragma once

int constraintValeur(int p_valeur, int p_min, int p_max);
```

Util.cpp

```
#include "Util.h"

int constraintValeur(int p_valeur, int p_min, int p_max) {
    int resultat = p_valeur;

    if (p_valeur < p_min) {
        p_valeur = p_min;
    }
    if (p_valeur > p_max) {
        p_valeur = p_max;
    }

    return p_valeur;
}
```

Déclaration / Définition – Exemple

AnimationLed.h

```
#pragma once

void animerLed(int p_pinDel,
                int p_increment = 100, int p_nombreCycles = 100,
                int p_delaiEteinte = 100);
```

AnimationLed.cpp

```
#include <Arduino.h>

#include "AnimationLed.h"
#include "Util.h"

void animerLed(int p_pinDel,
                int p_increment, int p_nombreCycles,
                int p_delaiEteinte) {
    int delai = 0;

    for (int numeroCycle = 0; numeroCycle < p_nombreCycles; ++numeroCycle) {
        digitalWrite(p_pinDel, HIGH);
        delay(delai);
        digitalWrite(p_pinDel, LOW);
        delay(p_delaiEteinte);

        delai = constraintValeur(delai + p_increment, 0, 32767);
    }
}
```

Déclaration / Définition – Exemple

MonFabuleuseAnimation.ino

```
#include "AnimationLed.h"

const int pinDel = 13;

void setup() {
    pinMode(pinDel, OUTPUT);
}

void loop() {
    animerLed(pinDel);
}
```

Déclaration / Définition – Exemple

Flasher.h

```
#pragma once

class Flasher {
private:
    int m_pinDel;
    int m_dureeAllumeeEteinte;

public:
    Flasher(int p_pinDel, int p_dureeAllumeeEteinte);
    void FaireClignoter(int p_nombreCycles);
};
```

Déclaration / Définition – Exemple

Flasher.cpp

```
#include <Arduino.h>

#include "Flasher.h"

Flasher::Flasher(int p_pinDel, int p_dureeAllumeeEteinte)
    : m_pinDel(p_pinDel),
      m_dureeAllumeeEteinte(p_dureeAllumeeEteinte) {
    pinMode(this->m_pinDel, OUTPUT);
    digitalWrite(this->m_pinDel, LOW);
}

void Flasher::FaireClignoter(int p_nombreCycles) {
    for (int numCycle = 0; numCycle < p_nombreCycles; ++numCycle) {
        digitalWrite(this->m_pinDel, HIGH);
        delay(this->m_dureeAllumeeEteinte);

        digitalWrite(this->m_pinDel, LOW);
        delay(this->m_dureeAllumeeEteinte);
    }
}
```

Déclaration / Définition – Exemple

MonFabuleuseAnimation.ino

```
#include "Flasher.h"

const int pinDel = 13;
Flasher flasherLent(pinDel, 500);
Flasher flasherRapide(pinDel, 100);
void setup() {
    ;
}

void loop() {
    flasherLent.FaireClignoter(5);
    flasherRapide.FaireClignoter(25);
}
```