

# Langage C pour le TSI

## Introduction

Marc Donias

## ► Langage C

- Un des meilleurs choix possibles
  - Langage « modèle » de référence (investissement pérenne)
  - Code « bas niveau »  $\Rightarrow$  performance (vitesse d'exécution, accès mémoire, etc.)
  - Code « haut niveau »  $\Rightarrow$  implémentations abstraites
- Un des pires choix possibles
  - Bogues
  - Fuites de mémoire

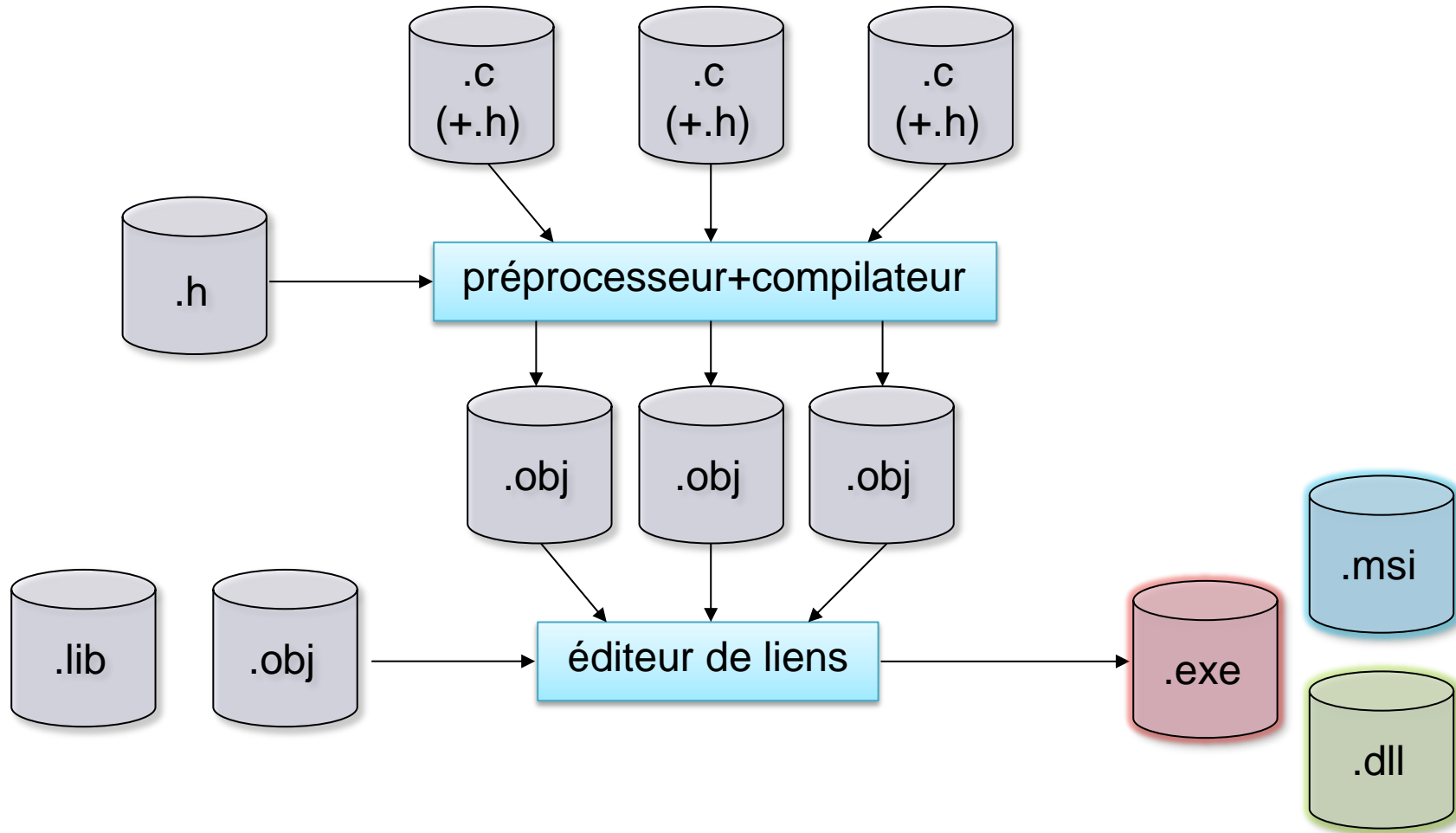
## ► « Pré-requis » des entreprises (et des étudiants)

- Vitesse d'exécution primordiale (vidéo, images 2D/3D gigantesques, etc.)
- Adéquation Matériel/Traitement du signal (portage sur cible embarquée, accélération GPU, etc.)

- ▶ Langage C pour le Traitement du Signal
  - Pré-requis : connaissance du Langage C
  - Maîtrise (parfaite) de l'utilisation des pointeurs
- ▶ Application aux traitement des images
  - Algorithmes : modification d'histogramme, filtrage, détection de contours, réduction de couleurs, ...
  - Bibliothèques dynamiques pour l'interface N'D sous Windows
- ▶ Optimisation
  - Ecriture (compréhension, réutilisabilité, maintenance)
  - Vitesse d'exécution

- ▶ Notions de base
  - Passage par référence
  - Tableaux
  - Allocation
- ▶ Types composés
- ▶ Arithmétique de pointeurs
- ▶ Notions avancées
  - Pointeurs génériques
  - Pointeurs de pointeurs
  - Pointeurs de fonctions

## Du code à l'exécutable (sous Windows)



## ► Codage : fichiers .c et .h

```
#include <compute.h>

double f(double x)
{
    return x*x-3.0*x+2.0;
}
```

compute.c

```
double f(double x);
```

compute.h

- Association « obligatoire »
- Multiples rôles du prototypage
  - Exportation (cohérence « appelé–appelant »)
  - Inclusion (cohérences « appelé–définition »)

## ► Codage : fichiers .c

- Visibilité : mot-clé `static` pour les variables et fonctions privées (non exportées)

```
static int windowHandle = 0;

static int windowCreate(void) {...}

void windowOpen()
{
    if ( !windowHandle )
        windowHandle = windowCreate();
}
```

- Ordre des fonctions

- Appelé au dessus de l'appelant
- Prototypage parfois indispensable (appels circulaires)

## ► Lisibilité

- Dénomination « intelligente » en lien (types, constantes, variables, fonctions, etc.)
- Conventions de forme (identifiables)

```
#define RAW_MAX      1000
int nbRaw, nb_raw;
int vectorGetMax(int * vector, long size);
int vector_get_max(int * vector, long size);
```

- Présentation (indentation)




```
int compute( int a,int i , int z)
{ int S
    S= i*z -a    ;
    return S; }
```

- Commentaires « utiles » : entête de fichier, de fonction (entrées/sorties, rôle) et de bloc de lignes



- Organisation : code « trié »
- Portée des variables, des fonctions (`static/export`)
  
- ▶ Modularité
  - Fonctions réutilisables
  - Conception de « boîtes noires »
  
- ▶ Exécution
  - Choix pertinents
    - Complexité calculatoire
    - Dualité algorithme/structure de donnée
  - Optimiser « utilement »
    - Identifier les lenteurs
    - Compromis Exécution/Lisibilité (souvent, optimiser = opacifier)

## ► Entiers signés (non signés)

- 8 bits char (unsigned char)   $[-128, 127]$
- 16 bits short (unsigned short)   $[-2^{15}, 2^{15}-1]$
- 32 bits long (unsigned long)  
int (unsigned int)   $[-2^{31}, 2^{31}-1]$

**1 / 2 = 0 en arithmétique entière !!!**

## ► Réels

- 32 bits float



$\sim \pm 10^{70}$

```
float a = 1.0f;
```

- 64 bits double



$\sim \pm 10^{343}$

```
double b = 2.0;
```

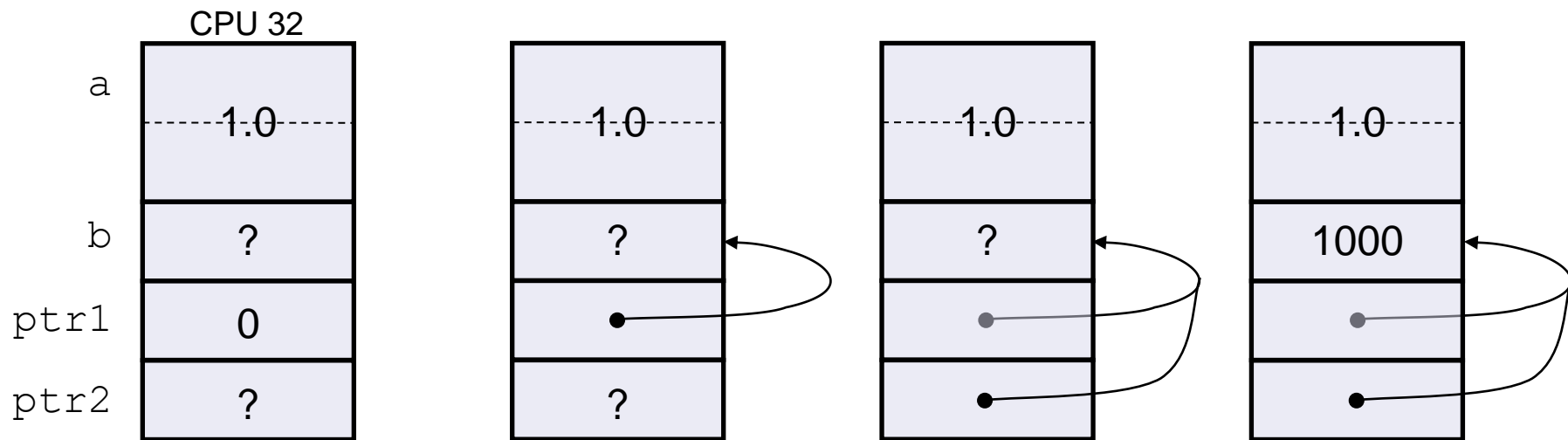
**Répartition binaire Mantisse/Exposant – Configurations impossibles !!!**

- Une adresse (donnée ou code)



- 11

## ► Pile et déclaration de variables

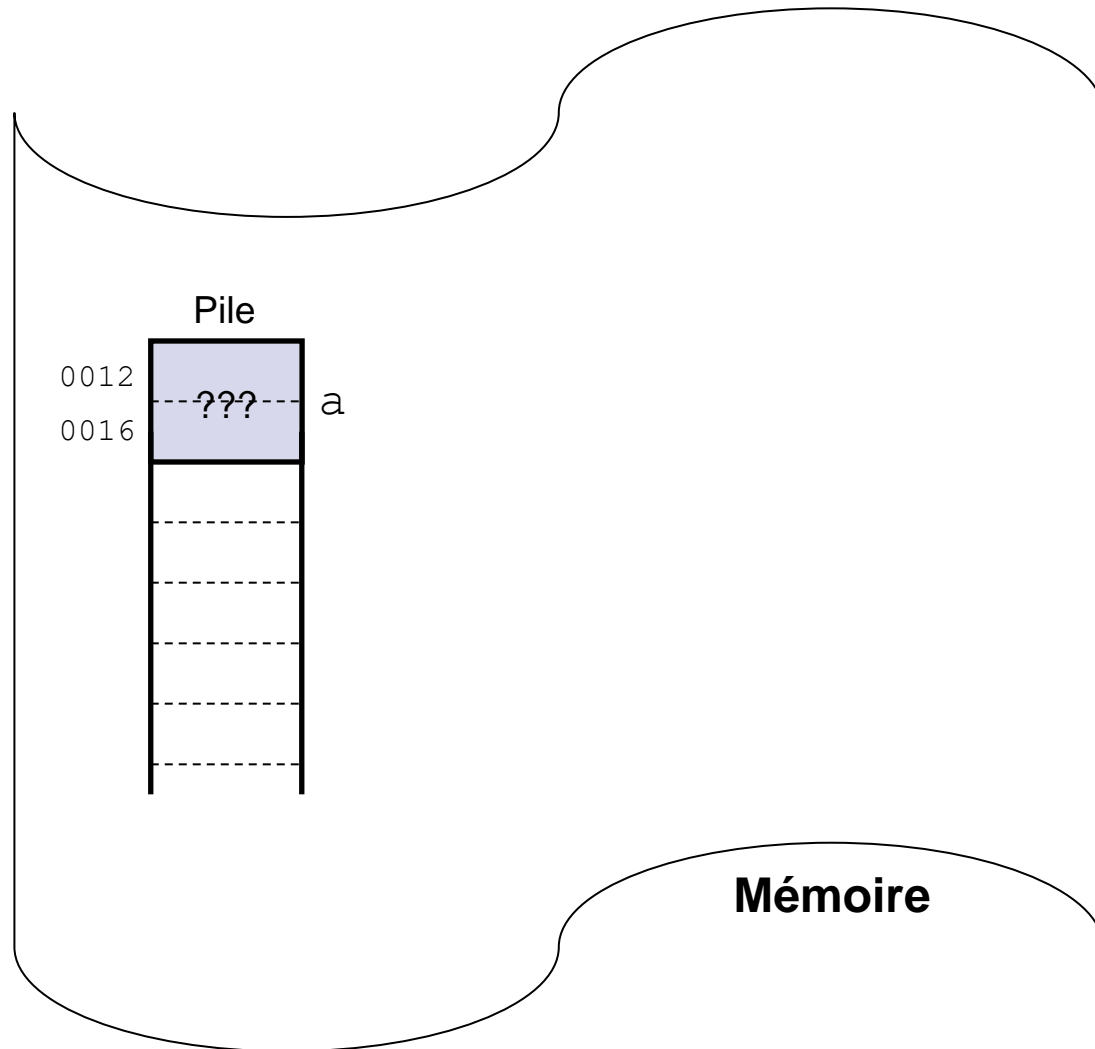


```
double a = 1.0;    ptr1 = &b;    ptr2 = ptr1;    *ptr1 = 1000;  
int b;  
int * ptr1 = NULL;  
int * ptr2;
```

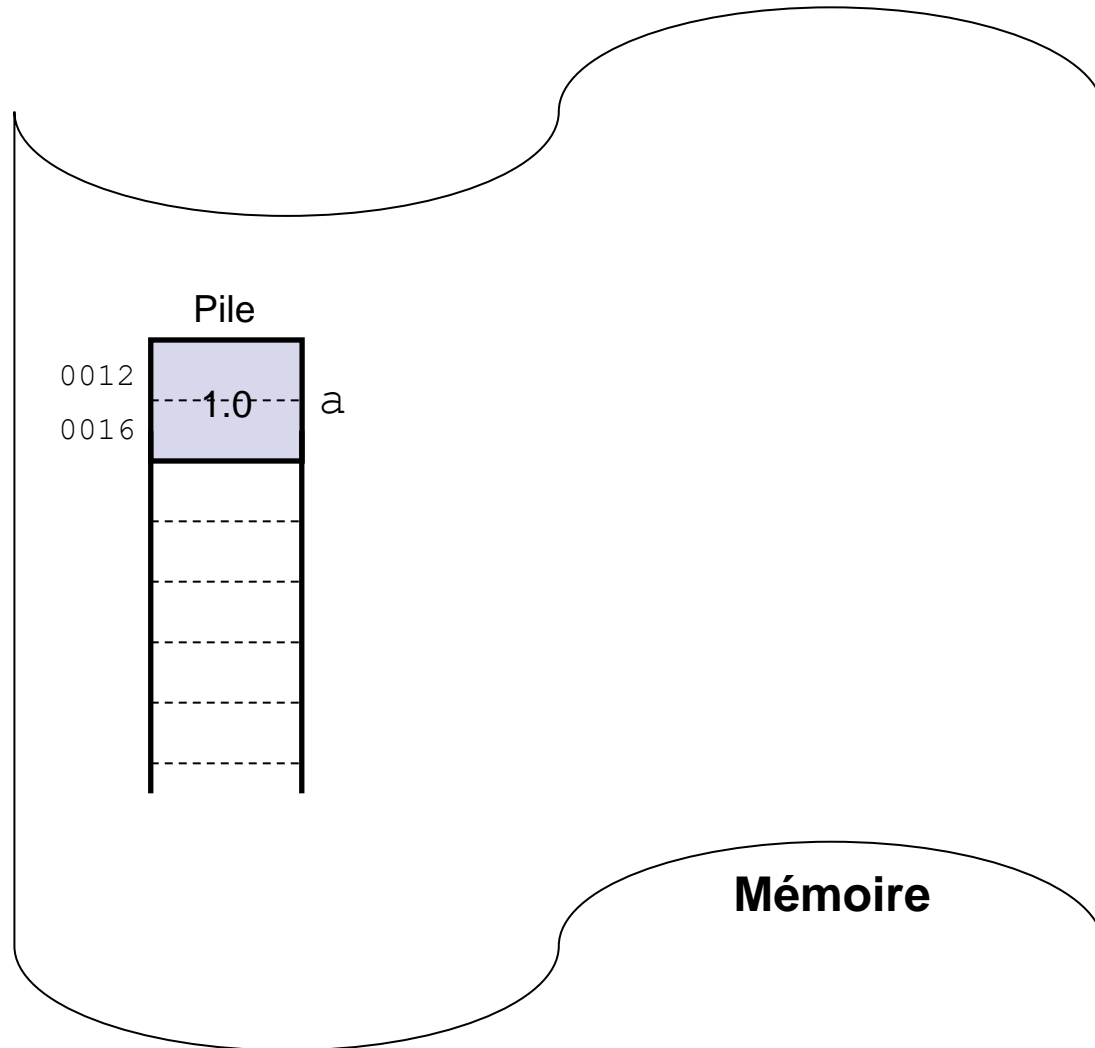
## ► Gestion de la mémoire

- Allocation : `malloc`, `calloc`, `realloc`
- Libération (« désallocation ») : `free`
  - Pour une région de mémoire déjà allouée
  - Affectation `NULL` non incluse
- Couple « indissociable »

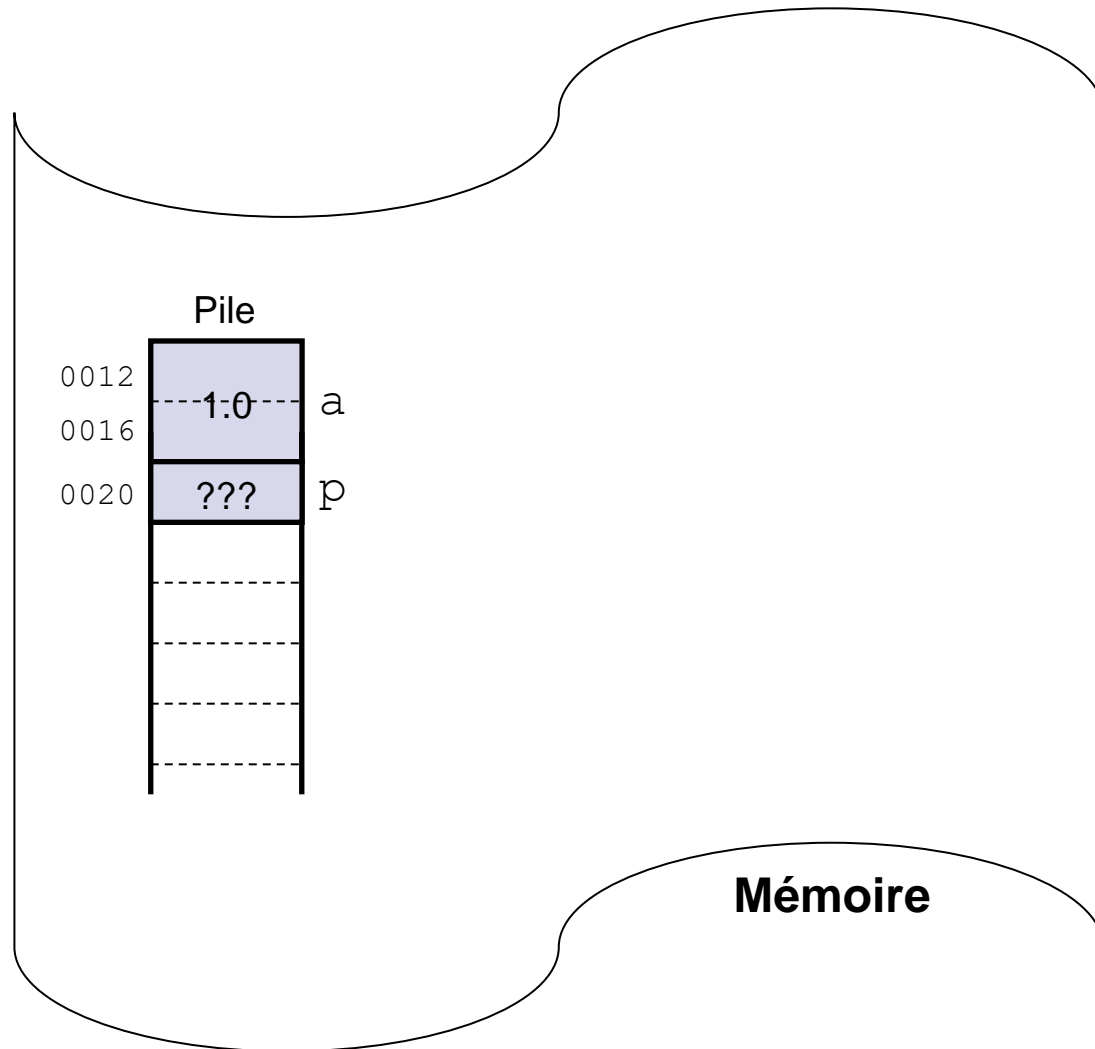
```
double a;
```



```
double a;  
a = 1.0;
```



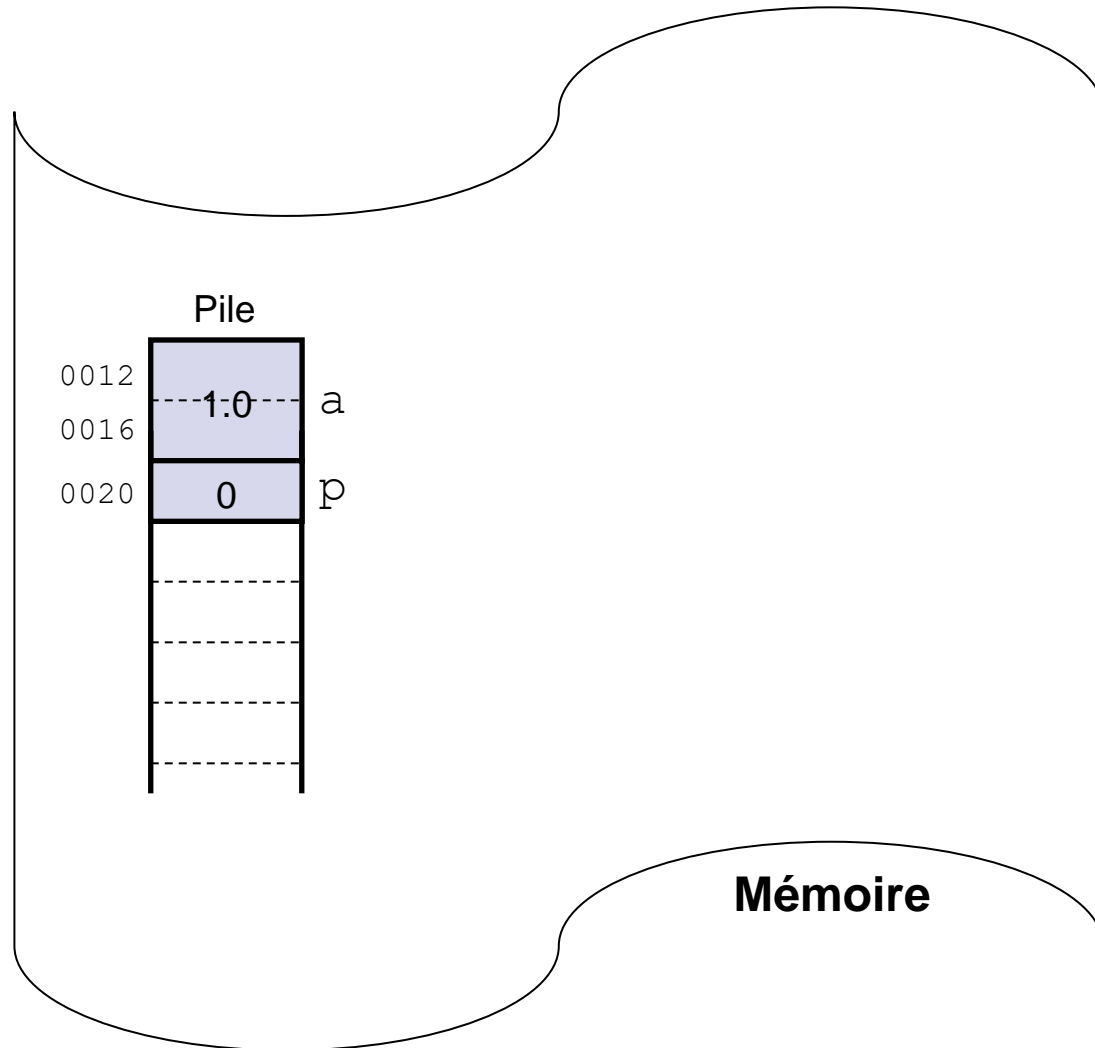
```
double a;  
a = 1.0;  
double * p;
```





# Pointeurs (4/4)

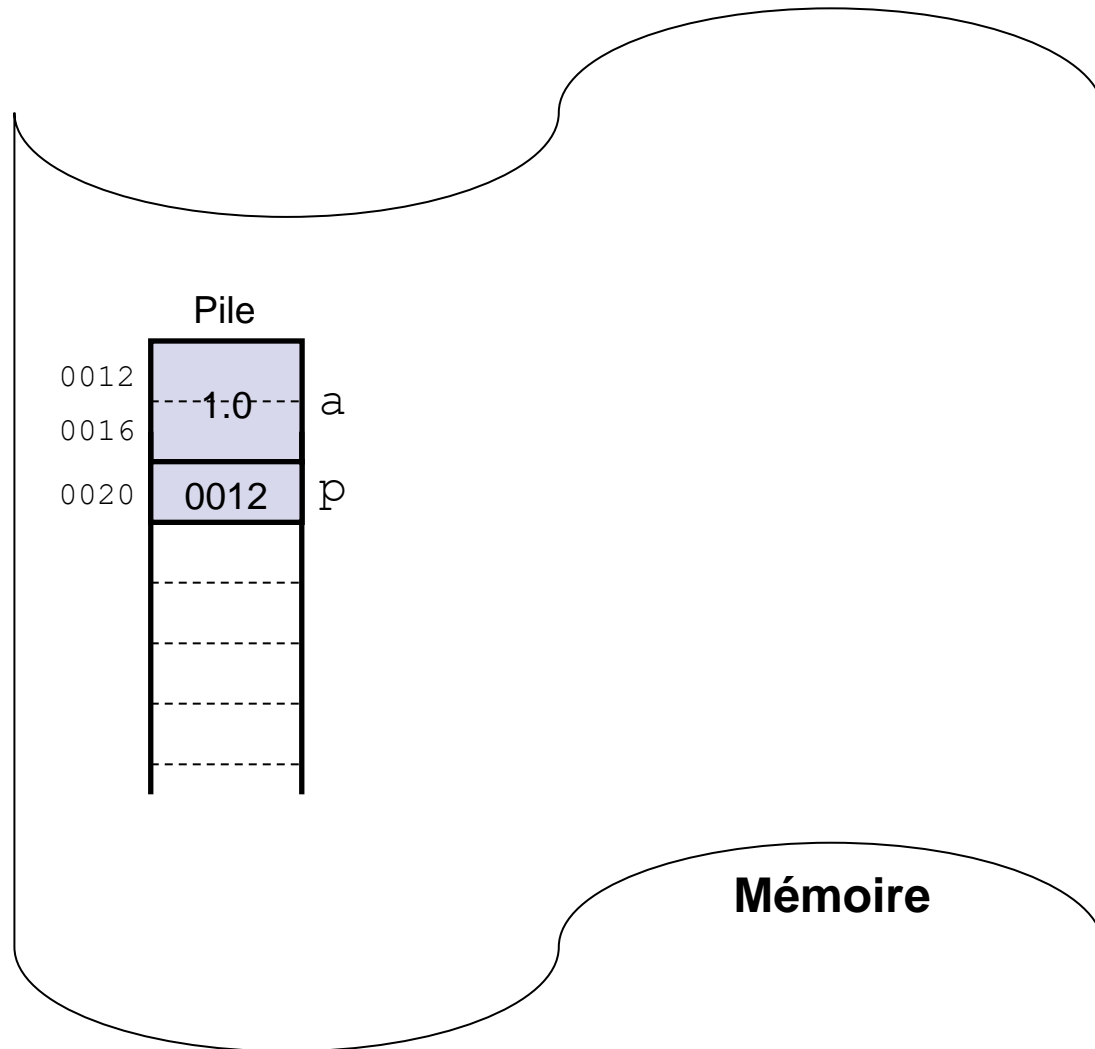
```
double a;  
a = 1.0;  
double * p;  
p = NULL;
```





# Pointeurs (4/4)

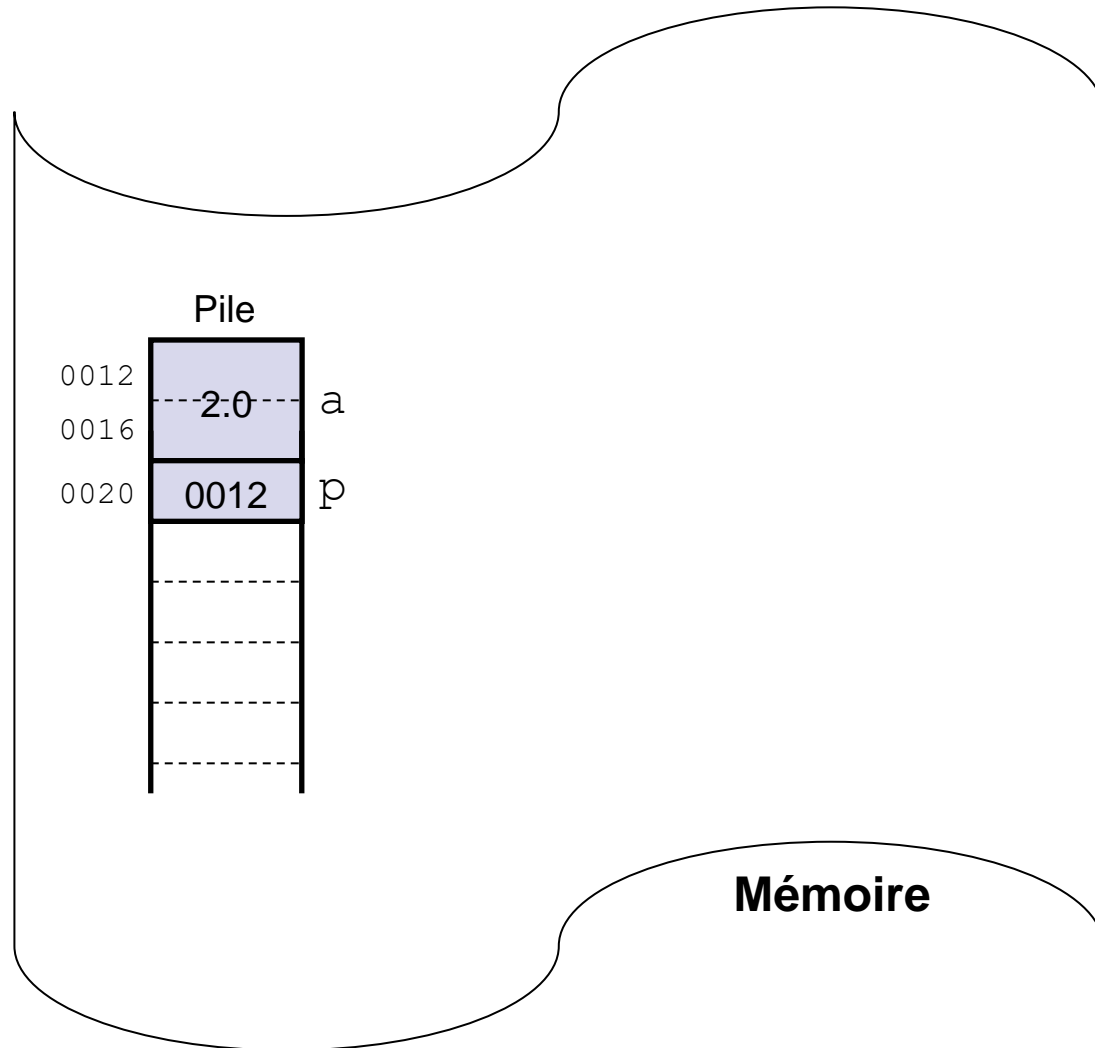
```
double a;  
a = 1.0;  
double * p;  
p = NULL;  
p = &a;
```





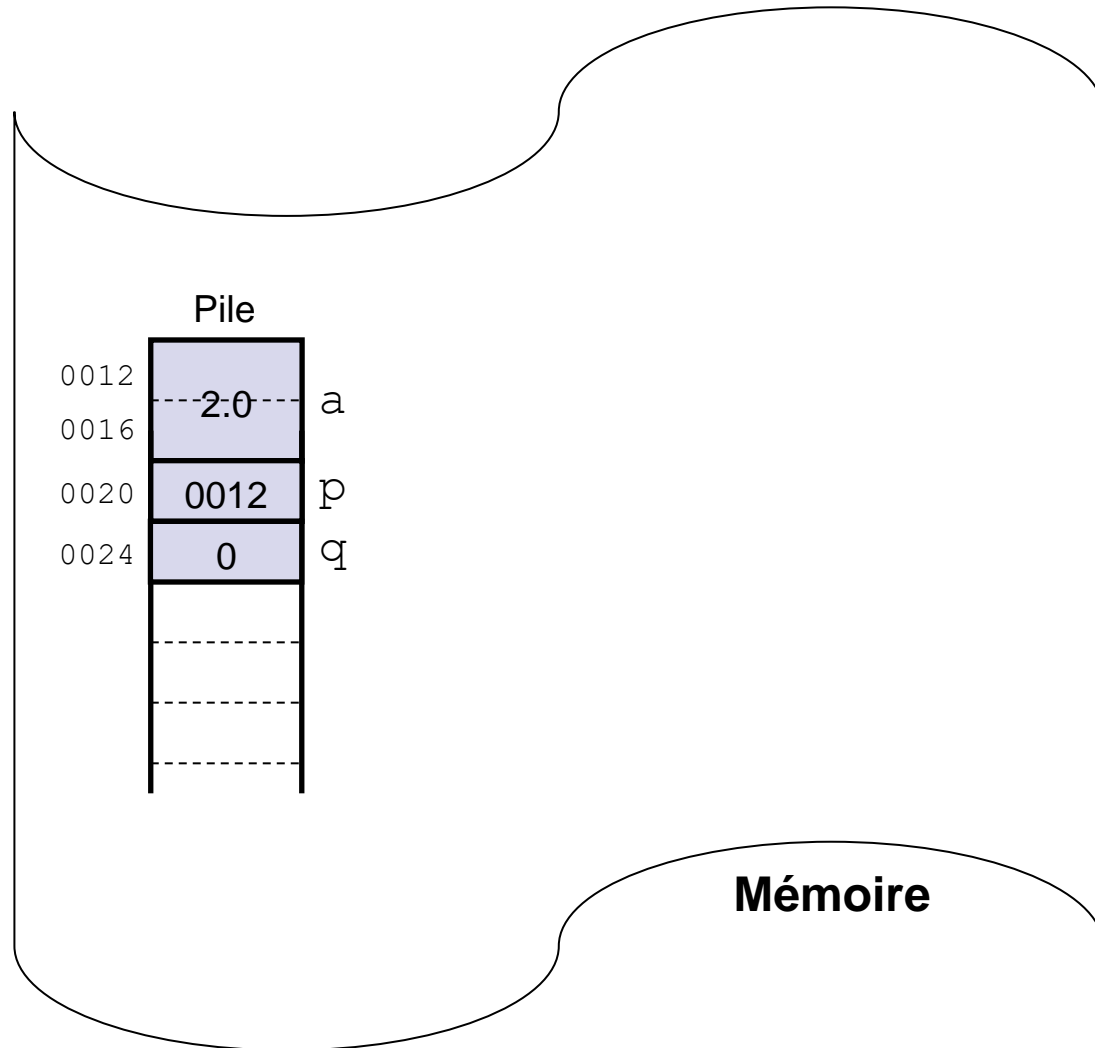
# Pointeurs (4/4)

```
double a;  
a = 1.0;  
double * p;  
p = NULL;  
p = &a;  
*p = 2.0;
```

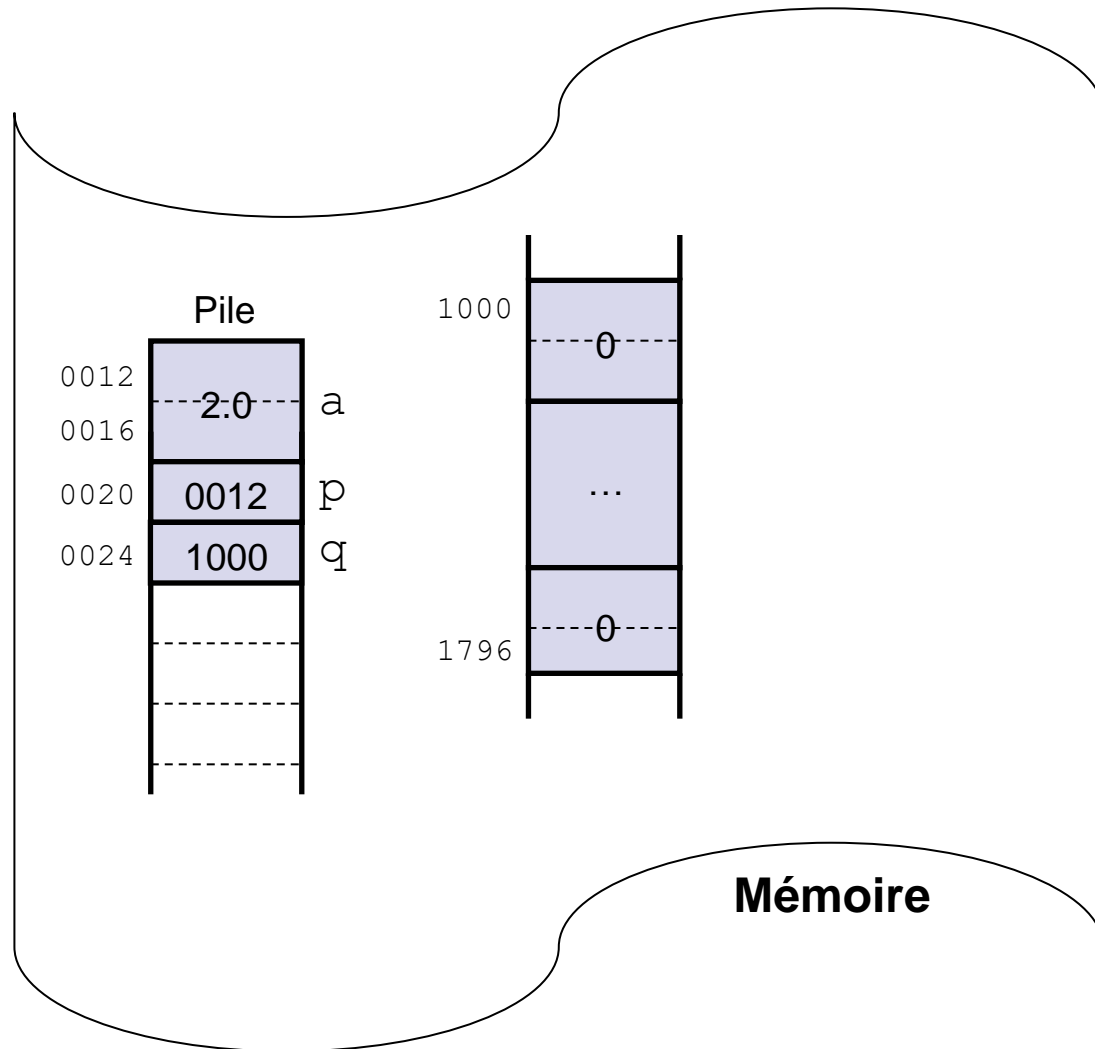




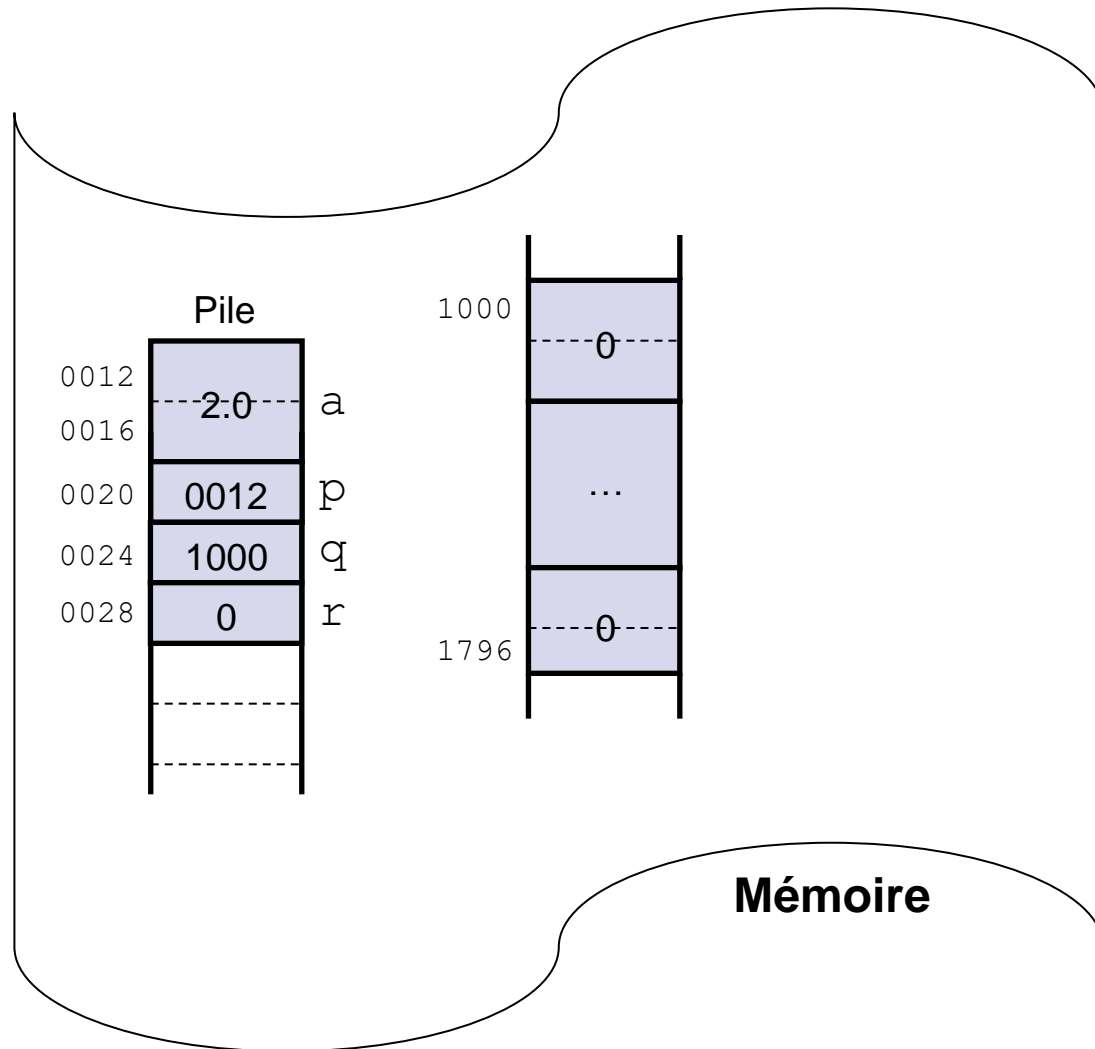
```
double a;  
a = 1.0;  
double * p;  
p = NULL;  
p = &a;  
*p = 2.0;  
double * q = NULL;
```



```
double a;  
a = 1.0;  
double * p;  
P = NULL;  
P = &a;  
*p = 2.0;  
double * q = NULL;  
q = (double *)calloc (100,  
sizeof(double));
```



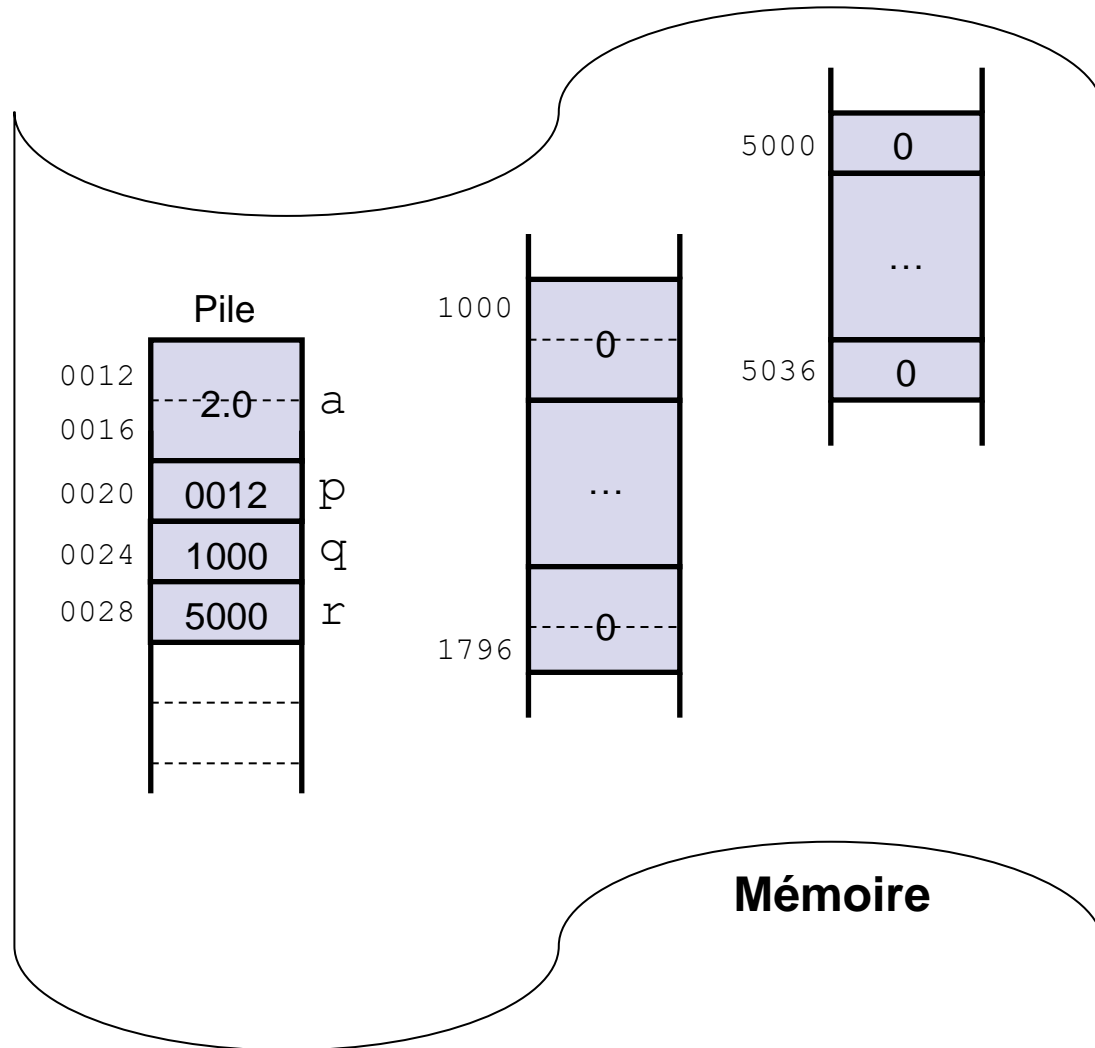
```
double a;  
a = 1.0;  
double * p;  
P = NULL;  
P = &a;  
*p = 2.0;  
double * q = NULL;  
q = (double *)calloc (100,  
sizeof(double));  
double ** r = NULL;
```





# Pointeurs (4/4)

```
double a;  
a = 1.0;  
double * p;  
P = NULL;  
P = &a;  
*p = 2.0;  
double * q = NULL;  
q = (double *)calloc (100,  
sizeof(double));  
double ** r = NULL;  
r = (double **)calloc(10,  
sizeof(double *));
```

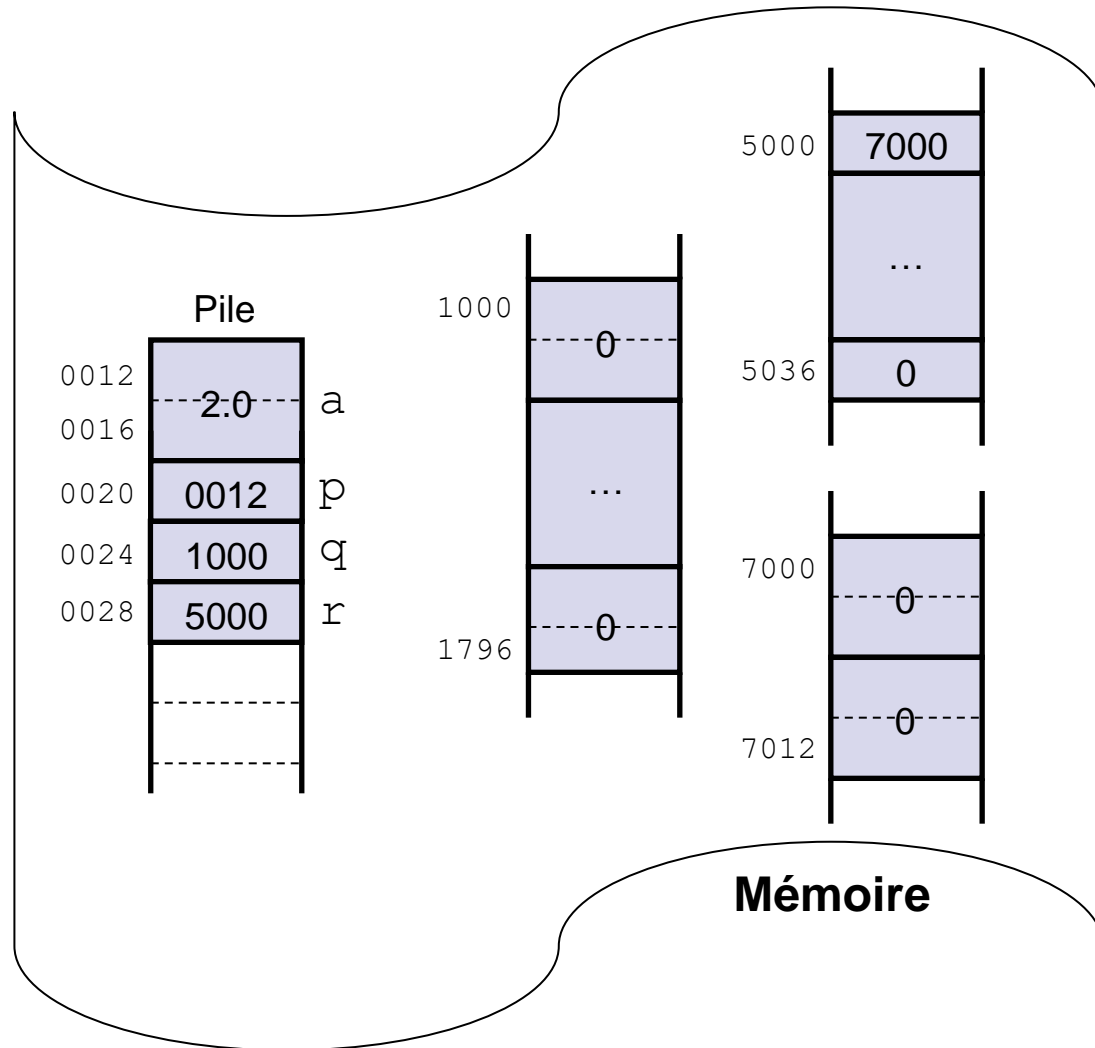






# Pointeurs (4/4)

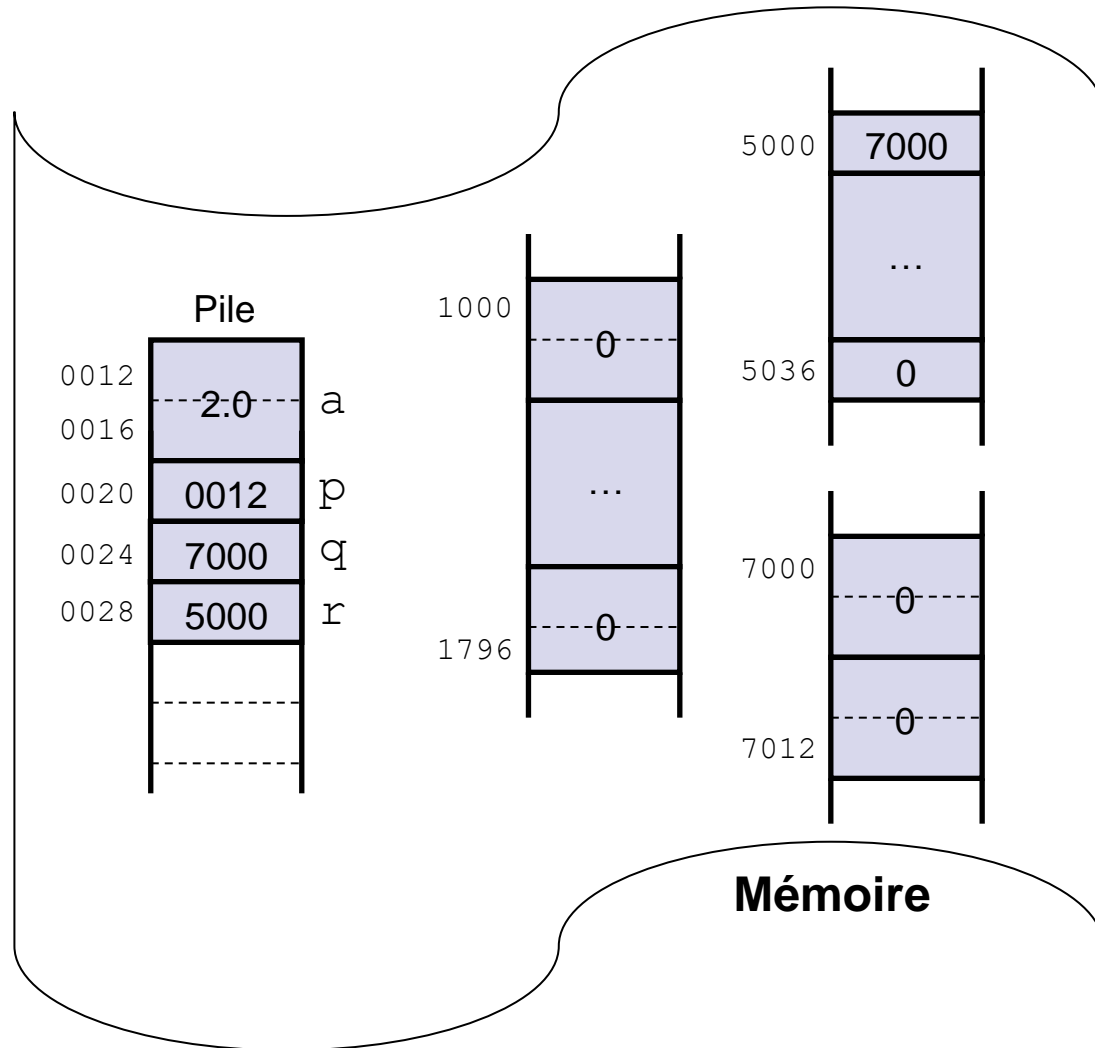
```
double a;  
a = 1.0;  
double * p;  
P = NULL;  
P = &a;  
*p = 2.0;  
double * q = NULL;  
q = (double *)calloc (100,  
sizeof(double));  
double ** r = NULL;  
r = (double **)calloc(10,  
sizeof(double *));  
r[0] = (double *)calloc(2,  
sizeof(double));
```





# Pointeurs (4/4)

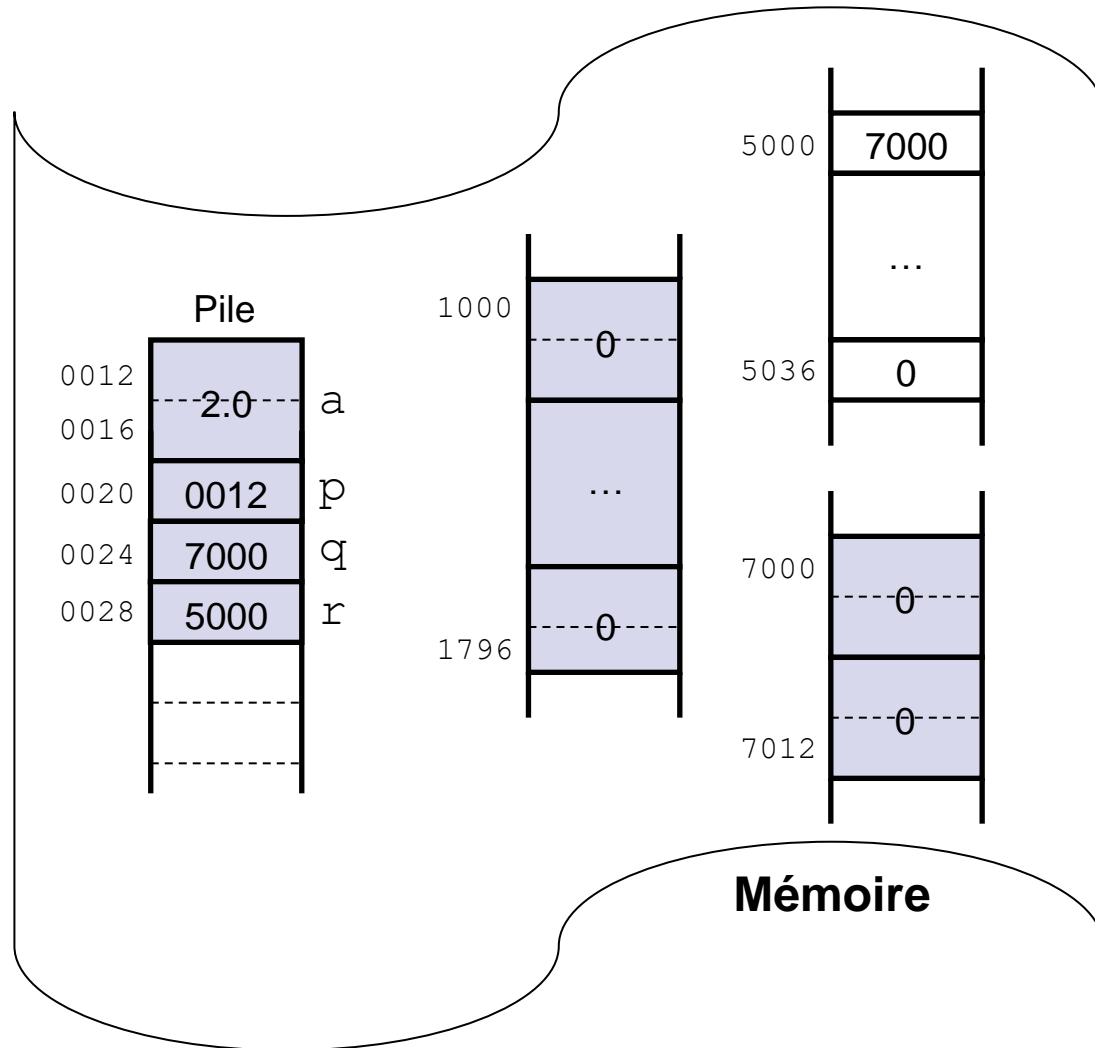
```
double a;  
a = 1.0;  
double * p;  
P = NULL;  
P = &a;  
*p = 2.0;  
double * q = NULL;  
q = (double *)calloc (100,  
sizeof(double));  
double ** r = NULL;  
r = (double **)calloc(10,  
sizeof(double *));  
r[0] = (double *)calloc(5,  
sizeof(double));  
q=r[0];
```



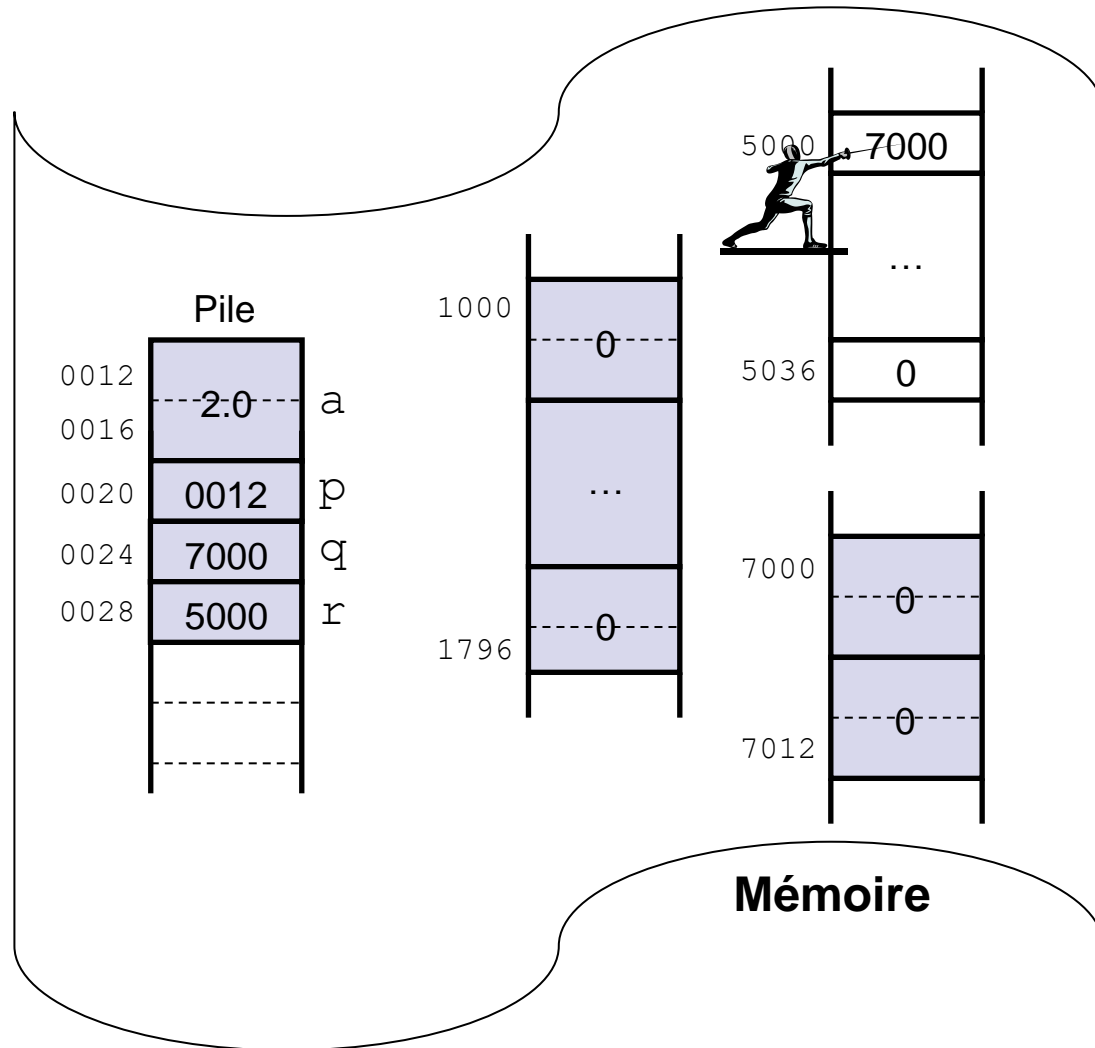


# Pointeurs (4/4)

```
double a;  
a = 1.0;  
double * p;  
P = NULL;  
P = &a;  
*p = 2.0;  
double * q = NULL;  
q = (double *)calloc (100,  
sizeof(double));  
double ** r = NULL;  
r = (double **)calloc(10,  
sizeof(double *));  
r[0] = (double *)calloc(5,  
sizeof(double));  
q=r[0];  
free(r);
```



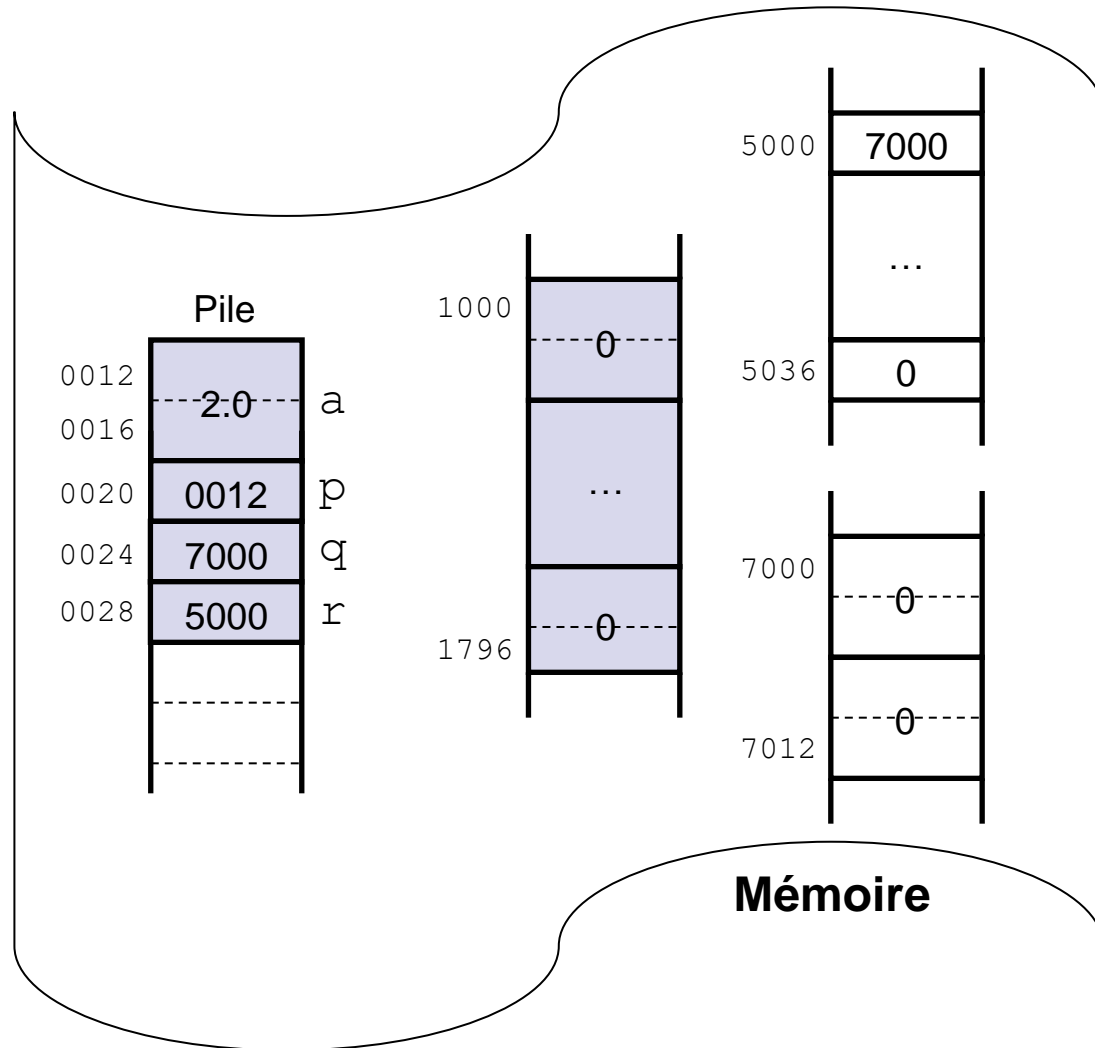
```
double a;  
a = 1.0;  
double * p;  
P = NULL;  
P = &a;  
*p = 2.0;  
double * q = NULL;  
q = (double *)calloc (100,  
sizeof(double));  
double ** r = NULL;  
r = (double **)calloc(10,  
sizeof(double *));  
r[0] = (double *)calloc(5,  
sizeof(double));  
q=r[0];  
free(r);  
free(r[0]);
```





# Pointeurs (4/4)

```
double a;  
a = 1.0;  
double * p;  
P = NULL;  
P = &a;  
*p = 2.0;  
double * q = NULL;  
q = (double *)calloc (100,  
sizeof(double));  
double ** r = NULL;  
r = (double **)calloc(10,  
sizeof(double *));  
r[0] = (double *)calloc(5,  
sizeof(double));  
q=r[0];  
free(r);  
free(q);
```

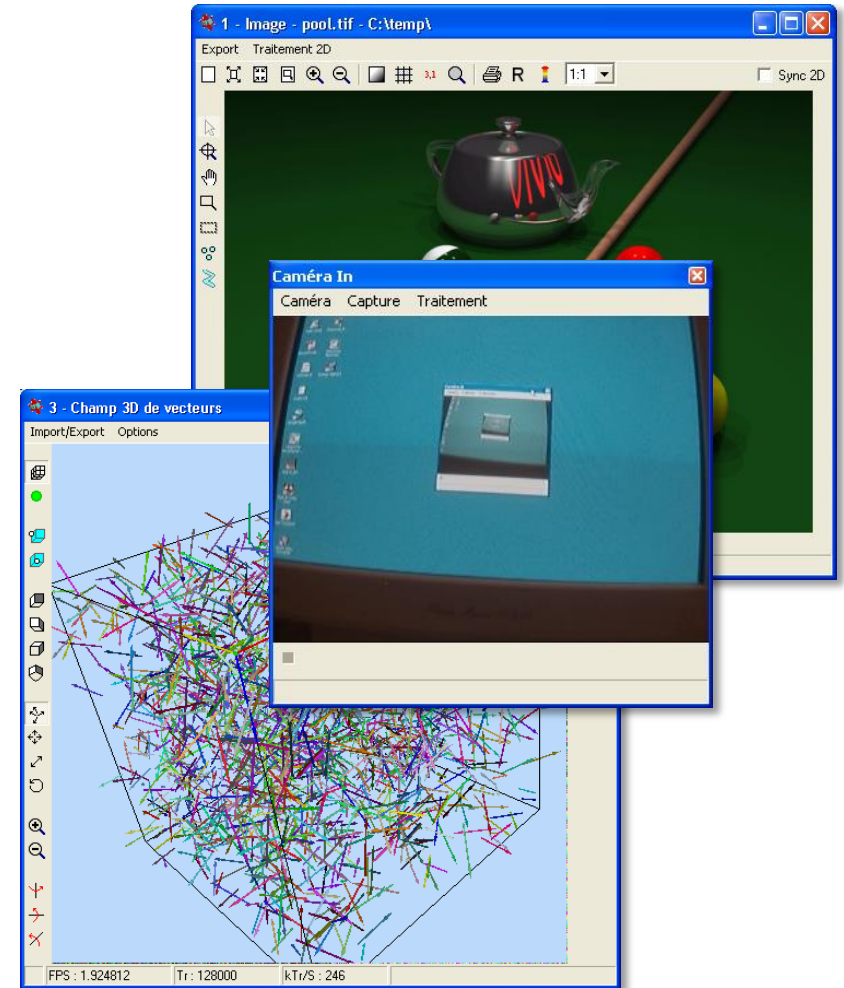
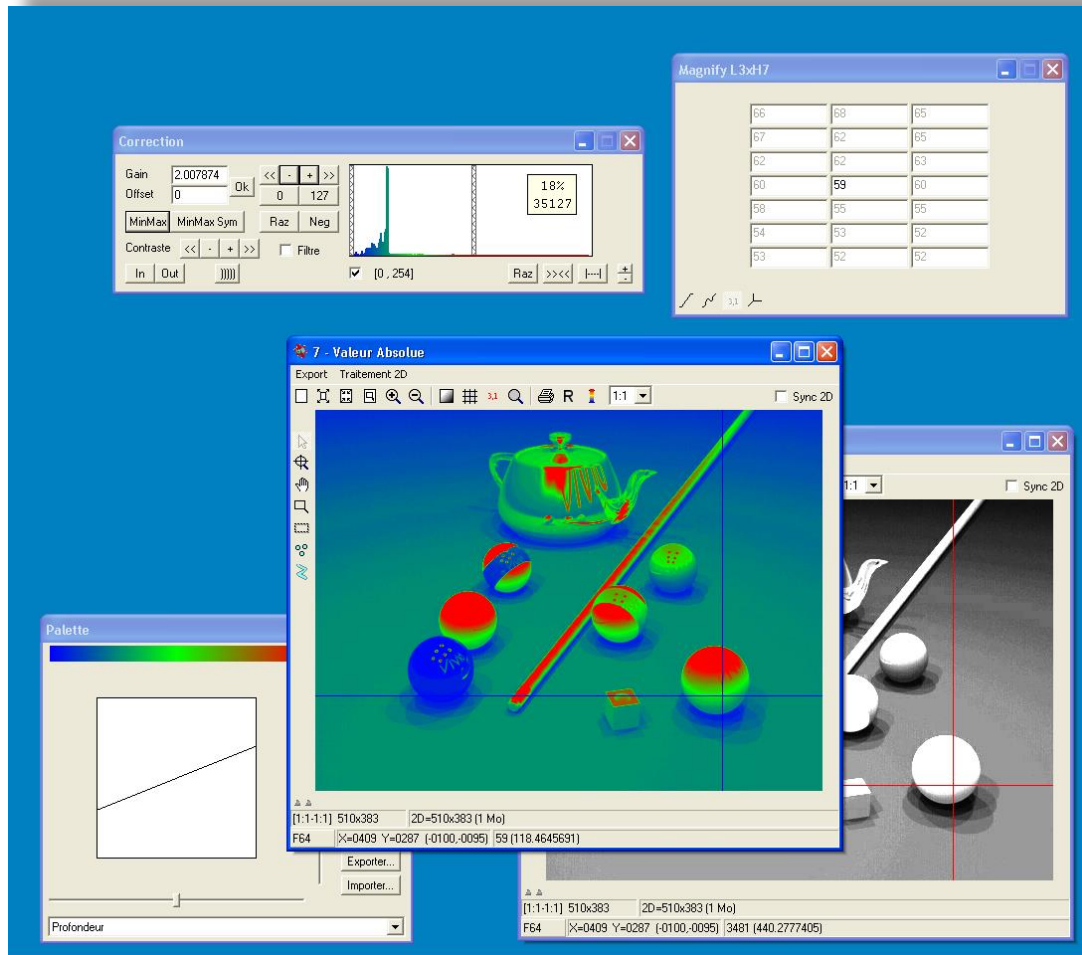


- ▶ Integrated Development Environment
  - Microsoft Visual Studio Community
- ▶ Un premier exemple, « Hello, World »

```
#include <stdio.h>

int main(void)
{
    printf("Hello, World\n");
    getchar();
    return 0;
}
```

## ► Interface N'D



## ► Arborescence

### ○ Tutorial\_Part1

#### ■ Compilation

- ✓ Compile\_Dll.sln
- ✓ Dll.vcxproj
- ✓ Debug ←



Fichiers de définition

Fichiers intermédiaires

#### ■ Dll

- ✓ include
- ✓ private
- ✓ src



Codes

### ○ Tutorial\_Part2

### ○ ...

### ○ Bin

- Debug
- Release



Formes binaires



Solution  
Projet