Programmation Orientée Objets

Mémoire et allocation dynamique

Charles Meunier

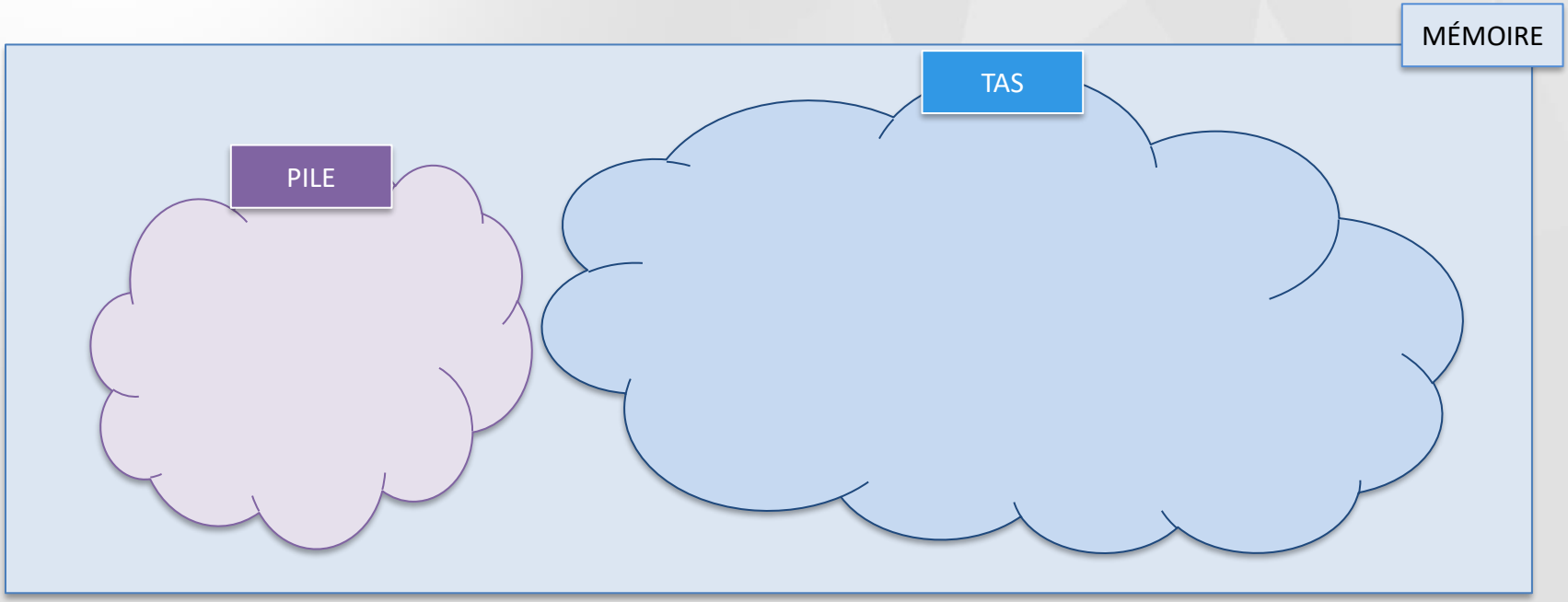
charles.meunier@u-bourgogne.fr

La mémoire

LA PILE ET LE TAS

charles.meunier@u-bourgogne.fr

UNE MÉMOIRE, DEUX ESPACES



charles.meunier@u-bourgogne.fr

PILE VS TAS

PILE

Taille fixe définie à la création du processus

Petite taille

Allocation statique

Passage de paramètres de fonctions

Retour des fonctions

Sauvegarde de contexte

TAS

Taille variable au cours de l'exécution du programme

Instructions programme

Constantes

Variables globales/statiques

Allocation dynamique

charles.meunier@u-bourgogne.fr

PRINCIPE DE LA PILE

PRINCIPE DE LA PILE

- Structure de type LIFO (last in, first out)
- Taille limitée
- Fonctionnement inversé :
  - Bas de la pile = + grande adresse
  - Haut de la pile = + petite adresse

DU C++ À L'ASSEMBLEUR

```
int somme(int a, int b)
{
     return a + b;
}

int main()
{
     int res;
     res = somme(5, 10);
}
```

main.asm

```
00991718 push 0Ah
0099171A push 5
0099171C call 01081370h
00991721 add  esp, 8
```

somme.asm

```
01081370  mov eax, dword ptr[a]
01081378  add eax, dword ptr[b]
0108137E  ret
```

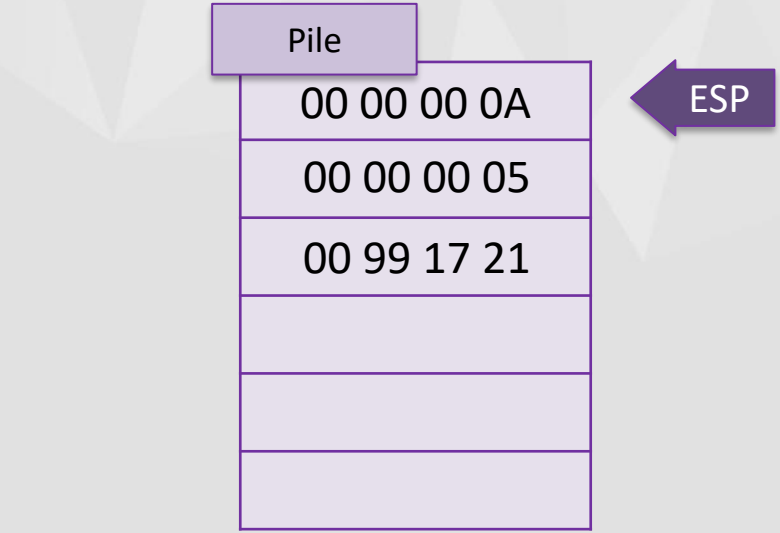
DU C++ À L'ASSEMBLEUR – PAS À PAS

main.asm

```
00991718  push 0Ah
0099171A  push 5
0099171C  call 01081370h
00991721  add  esp, 8
```

somme.asm

```
01081370   mov eax, dword ptr[a]
01081378   add eax, dword ptr[b]
0108137E   ret
```



Allocation dynamique

ALLOUER DE LA MÉMOIRE

charles.meunier@u-bourgogne.fr

DÉCLARATION SIMPLE

```
int entier;
double reel = 5.86;
std::string chaine = "Une chaîne !"
MaClasse monInstance;
```

LE COMPILATEUR S'OCCUPE DE TOUT

- Il alloue la mémoire à la déclaration, sur la pile
- Il libère la mémoire en fin de portée

DURÉE DE VIE = PORTÉE

charles.meunier@u-bourgogne.fr

AVANTAGES

- Simple à utiliser
- Gestion sans risque : le compilateur gère la mémoire

MAIS…

- Durée de vie différente de la portée ?
- Partage d'instance ?
- Allocation de grande taille ?

charles.meunier@u-bourgogne.fr

Solution

L'ALLOCATION DYNAMIQUE

charles.meunier@u-bourgogne.fr

PRINCIPE ET SYNTAXE

- Décidée à l'exécution
- Utilisation de pointeurs
- Allocation sur le tas
- Appel du constructeur

```
//Allocation dynamique d'un entier
int* entier_dyn = new int;

//Allocation dynamique d'une instance de Table
Table* table = new Table();

/**
 * Allocation dynamique d'une instance de Glace (utilisation du
 * contructeur prenant en paramètre le parfum de la glace)
 */
Glace* glace = new Glace("Fraise");
```

VIE DE L'ÉLÉMENT ALLOUÉ != VIE DU POINTEUR

```
int main(int argc, char* argv[])
{
     if(argc === 2)
     {
          Voiture* voiture = new Voiture(argv[1]);
     }

     return 0;
}
```

/!\ LA LIBÉRATION N'EST PAS AUTOMATIQUE /!

POINTEURS ET CLASSES

L'opérateur permettant d'accéder aux membres d'une instance n'est pas le même en fonction que l'on y accède via une valeur ou un pointeur (une adresse).

```
//Allocation statique (voiture est une valeur)
Voiture voiture;
voiture.setConstructeur("Opel");

//Allocation dynamique (voiture est une adresse)
Voiture* voiture = new Voiture;
voiture->setConstructeur("Opel");
```

Allocation dynamique

LIBÉRATION DE LA MÉMOIRE

charles.meunier@u-bourgogne.fr

PRINCIPE ET SYNTAXE

- Pas de Garbage Collector
- Libération manuelle de la mémoire
- 1 new = 1 delete
- Appel du destructeur

```
//Allocation dynamique d'un entier
int* entier_dyn = new int;

//Libération de l'entier alloué
delete entier_dyn;
```

charles.meunier@u-bourgogne.fr

Allocation dynamique

ALLOCATION DE PLUSIEURS OBJETS

charles.meunier@u-bourgogne.fr

PRINCIPE ET SYNTAXE

- Allocation via l'opérateur new[]
- Indication du nombre d'objets à créer
- Utilisation de variables possibles
- Libération via l'opérateur delete[]

```
//Propriétés d'une image
int largeur = 1920;
int hauteur = 1080;
int composantes = 3; // Rouge, Vert et Bleu

//Allocation dynamique des pixels d'une image
char* image = new char[largeur * hauteur * composantes];


//Utilisation de l'image
…


//Libération de la mémoire
delete[] image;
```

Allocation dynamique

EN C++ MODERNE

charles.meunier@u-bourgogne.fr

CONTRAINTES DE L'ALLOCATION DYNAMIQUE

- 1 new = 1 delete
- La libération (delete) peut intervenir dans une toute autre partie du code
- Risque d'oublier de libérer la mémoire
- Risque de fuites mémoire

POINTEURS INTELLIGENTS

La bibliothèque standard propose des pointeurs "intelligents" :

- Associe l'objet alloué en mémoire à un block de contrôle
- Le block de contrôle compte le nombre de pointeurs sur l'objet alloué
- Quand le nombre de pointeurs atteint 0, l'objet est supprimé

2 TYPES DE POINTEURS INTELLIGENTS

`shared_ptr` : l'objet peut être pointé par plusieurs pointeurs

- Peut être utilisé pour représenté une agrégation

`unique_ptr` : l'objet ne peut être pointé qu'une fois à un instant t

- Peut être utilisé pour représenté une composition

SYNTAXE - DÉCLARATION

```
//Alloue dynamiquement un entier initialisé avec la valeur 1732
std::shared_ptr<int> pEntierPartage = std::make_shared<int>(1732);

//Crée un nouveau pointeur qui pointe sur l'entier alloué précédemment
std::shared_ptr<int> pEntier2 = pEntierPartage;


//Alloue dynamiquement un entier initialisé avec la valeur 384
std::unique_ptr<int> pEntierUnique = std::make_unique<int>(384);

//Tentative de partager le pointeur unique
std::unique_ptr<int> pEntier3 = pEntierUnique;   // ERREUR
std::shared_ptr<int> pEntier4 = pEntierUnique;   // ERREUR
```

SYNTAXE - UTILISATION

Les pointeurs intelligents s'utilisent comme des pointeurs standards :

```
//Alloue dynamiquement un entier initialisé avec la valeur 1732
std::shared_ptr<int> pEntier = std::make_shared<int>(1732);

//Affiche la valeur de l'entier
std::cout << *pEntier << std::endl;

//Affiche l'adresse de l'entier
std::cout << pEntier << std::endl;
```

INFÉRENCE DE TYPES

Le mot clé **auto** permet de détecter automatiquement le type de la variable à partir de la valeur avec laquelle elle est initialisée :

```
//Sans l'inférence de type :
std::shared_ptr<int> pEntier = std::make_shared<int>(1732);

//Avec l'inférence de type :
auto pEntier2 = std::make_shared<int>(1732);
```