Objectifs
Passage par référence
Copie profonde
Conclusion

Programmation « orientée système »
LANGAGE C – POINTEURS (4/5)

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

Objectifs
Passage par référence
Copie profonde
Conclusion

Objectifs du cours d'aujourd'hui

Les pointeurs en pratique :

- retour sur le « swap »
- copie profonde

Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pointeurs et passage par référence (piqûre de rappel)

Souvenez-vous de la première semaine sur les pointeurs (semaine 5) :

```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

Pourquoi n'a-t-on pas écrit :

```
void swap(int* a, int* b) {
  int* tmp = a;
  a = b;
  b = tmp;
}
```

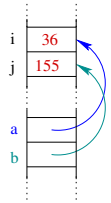
Que se passe-t-il dans ce second cas ?

Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pointeurs et passage par référence (piqûre de rappel)

```
void swap(int* a, int* b) {
  int* tmp = a;
  a = b;
  b = tmp;
}
```

Que se passe-t-il ?




```
swap(&i, &j);
```

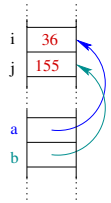
1) `a=&i` et `b=&j`

Objectifs
Passage par référence
Copie profonde
Conclusion

Pointeurs et passage par référence (piqûre de rappel)

```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

Code correct, là ça fonctionne :




```
swap(&i, &j);
```

1) a=&i et b=&j

Pointeurs et passage par référence
Et en Java ?

Comment écrire `swap` en Java ?

```
public static void swap(Object o1, Object o2) {
   Object tmp = o1;
   o1 = o2;
   o2 = tmp;
}
```

ne fonctionne pas !...où plutôt fonctionne exactement comme le mauvais exemple C précédent.

Pourquoi ?

En Java, que signifie

```
Object o1;
```

Qu'est `o1` ?
Et que signifie

```
o1 = o2;
```


Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pointeurs et passage par référence
Et en Java ?

Comment écrire `swap` en Java ?

```
public static void swap(Object o1, Object o2) {
   Object tmp = o1;
   o1 = o2;
   o2 = tmp;
}
```

ne fonctionne pas !...où plutôt fonctionne exactement comme le mauvais exemple C précédent.

Solution ? Il faut fournir l'équivalent de l'opération « `*b=*a` », c'est-à-dire un opérateur de copie de contenu :

```
public static void swap(ObjetCopiable o1, ObjetCopiable o2) {
   ObjetCopiable tmp = new ObjetCopiable();
   tmp.contentcopy(o1);
   o1.contentcopy(o2);
   o2.contentcopy(tmp);
}
```

mais **ATTENTION !** cette méthode `contentcopy` doit effectuer une **copie profonde** (i.e. appeler et redéfinir `clone()` là où nécessaire).

Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Copie profonde et copie de surface

Considérons la structure suivante en C, somme toute assez naturelle :

```
typedef struct {
  char* nom;
  int age;
} Personne;
```

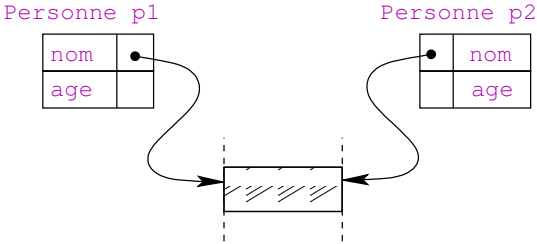
Quel(s) problème(s) potentiel(s) cette structure cache-t-elle ?

Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Copie profonde et copie de surface

J'en vois au moins deux :

1. intégrité des données (partage d'une zone pointée) :



2. problèmes plus spécifiques liés aux chaînes de caractères en C (`char*` .vs. `const char*`) :

```
quidam.nom = "Eugénie";
```


Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Copie profonde et copie de surface

Voici un code qui va nous servir d'exemple :

```c
#include <stdio.h>  /* pour printf() */
#include <stdlib.h> /* pour malloc() et free() */
#include <string.h> /* pour strncpy() */

/* on verra plus tard... */
#define MAX_NOM 100

/*
   une petite structure toute simple...
   ...mais pleine de surprises potentielles !!
*/
typedef struct {
  char* nom;
  int age;
} Personne;
```


Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

```c
/* ----- une fonction utilitaire ----- */
void print_personne(const char* entete, const Personne* p)
{
  if (entete != NULL) fputs(entete, stdout);
  if (p->nom != NULL) /* C'est la même chose que (*p).nom */
    printf("%s - %d\n", p->nom, p->age);
  else
    printf("Personne non définie\n");
}

/* ---- et maintenant : le programme ! ---- */
int main(void)
{
  /* ==== CHAPITRE 1 : const char* ========================== */

  Personne pierre = { "Pierre", 12 }; /* (1): faute : devrait au
                                         moins etre const ! */
  Personne quidam;

  print_personne("1) pierre : ", &pierre);

  strncpy(pierre.nom, "Gustave", 7); /* SEGV : illustration de (1) */

  pierre.nom = "Gustave"; /* (2) : pas mieux que (1) ! */
  print_personne("2) pierre : ", &pierre);
```

```c
/* bonne façon de faire : allocation dynamique */
  pierre.nom = calloc(MAX_NOM+1, sizeof(char));
  if (pierre.nom == NULL) { /* ... */ return 1; }
  pierre.nom[MAX_NOM] = '\0'; /* pourquoi cette ligne? */
  strncpy(pierre.nom, "Eugène", MAX_NOM); /* maintenant ça joue */
  print_personne("3) pierre : ", &pierre);


  /* ==== CHAPITRE 2 : copie de surface ==================== */
  quidam = pierre;
  print_personne("4) quidam : ", &quidam);
  /* ... */
  strncpy(quidam.nom, "Charles-Édouard", MAX_NOM);
  quidam.age = 22;

  print_personne("5) quidam : ", &quidam);
  print_personne("   pierre : ", &pierre);
  printf("adresse pointee par quidam.nom: %08x\n",
         (unsigned int) quidam.nom);
  printf("adresse pointee par pierre.nom: %08x\n",
         (unsigned int) pierre.nom);

  /* !! ne pas oublier la règle d'or de l'allocation dynamique */
  free(pierre.nom);

  return 0;
```

Copie profonde et copie de surface

(sans le SEGV) on obtient :

```
1) pierre : Pierre - 12
2) pierre : Gustave - 12
3) pierre : Eugène - 12
4) quidam : Eugène - 12
5) quidam : Charles-Édouard - 22
   pierre : Charles-Édouard - 12
adresse pointee par quidam.nom: 0804a008
adresse pointee par pierre.nom: 0804a008
```

(programme à disposition sur le site web du cours)

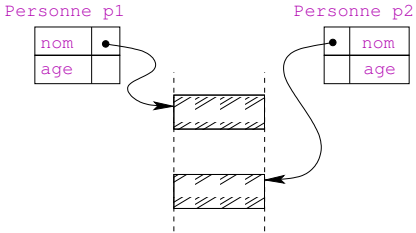
Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Copie profonde et copie de surface

Solutions ?

1. au problème 1 : faire une copie **profonde**

```
void copie(const Personne* a_copier, Personne* clone)
{ /* Attention ! Suppose ici que le clone a la place
   * pour recevoir son nom !
   * On pourrait améliorer ici en faisant de la réallocation.
   */
  strncpy(clone->nom, a_copier->nom, MAX_NOM);
  clone->age = a_copier->age;
}
```



Copie profonde et copie de surface

Solutions ?

2. au problème 2 : faire attention aux `const char*` !!

   Une chaîne de caractères littérale (par exemple `"Gustave"`) est en fait un `const char*` pointant sur de la mémoire que le programmeur n'a **pas le droit de manipuler**.

   À n'utiliser donc que pour des chaînes de caractères qui restent constantes. Sinon, utiliser une copie, par exemple :

```
/* dest a au moins taille+1 char */
strncpy(dest, "chaine", taille);
dest[taille]='\0';
```


Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Copie profonde et copie de surface

Une dernière note pour finir.
Dans **ce cas précis**, bien des soucis auraient pu être évités en choisissant comme structure :

```
/*
   une petite structure toute simple...
   ...et sans surprise.
*/
typedef struct {
  char nom[MAX_NOM+1];
  int age;
} Personne;
```

Pensez à ce genre de solutions et utilisez les lorsqu'elles sont suffisantes pour vos besoins.

Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Ce que j'ai appris aujourd'hui

À faire bien attention aux pièges usuels de l'utilisation des pointeurs :

- penser à la **copie profonde** (lorsque cela est nécessaire) ;
- faire attention aux zones mémoires pointées ;
- penser à faire la différence entre un `type*` et un `const type*` (en particulier sur les `char`).

Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE