

# Algorithmique et programmation

## Programmation générique

R.Gosswiller

- 1 Fonctions et pointeurs génériques
- 2 Fonction à nombre variable d'arguments

# Fonctions et pointeurs génériques

---

# Principes

## Principe

Un pointeur générique ne peut pas pointer vers une fonction, seulement vers une variable (un 'objet').

## Syntaxe

On utilise à la place une autre syntaxe : `typeRetour (*pointeur)();`

## Exemple

```
1      int  (*pf)();
```

## Principe

On ne spécifie pas la/les valeur/s d'entrée, ce qui permet d'affecter ce pointeur à différentes fonctions

# Exemples

## Exemples

```
1  int (*f)(int, int);
2  int (*g)(char, char, double);
3  double (*h)(void);
4  int (*pf)();
5
6  pf = f; // Ok.
7  pf = g; // Ok.
8  pf = h; // Erreur car le type de retour de 'h' est 'double'.
9  f = pf; // Ok.
```

## Attention

Un pointeur générique de fonction peut avoir l'adresse de différentes fonctions

Lors de l'appel, les arguments sont envoyés peut importe leur type

## Attention

C'est au développeur de faire attention !

## Syntaxe

```
1  int triple(int a){ return a*3; }
2  int (*pf)() = triple;
3
4  (*pf)(3); //correct
5  (*pf)('a'); //Erreur!
```

# Promotion des arguments

## Principe

Afin de limiter les types possibles, le compilateur va effectuer certaines approximations : les promotions d'arguments

## Promotions

Types plus petits que `int` -> `int` (notamment `char` -> `int` et `short` -> `int`)  
(ou `unsigned int` pour des valeurs extrêmes)

Types entre `int` et `double` -> `double` (notamment `float` -> `double`)

## Attention

Une fonction passée par un pointeur générique ne pourra jamais avoir ces arguments

# Exemple de Promotion d'arguments

## Syntaxe

```
1  #include <stdio.h>
2
3  int triple(int a){ return a*3; }
4  float augmente(float f){return 3.5 * f;}
5  short shortQuadruple(short n){return 4 * n;}
6
7  int main(void)
8  {
9      int (*ptTri)() = &triple;
10     float (*ptAug)() = &augmente;
11     short (*ptQuad)() = &shortQuadruple;
12
13
14     printf("triple=%d.\n", (*ptTri)(3)); //Affiche 9
15     printf("augmente=%f.\n", (*ptAug)(3)); // Erreur
16     printf("shortQuadruple=%d.\n", (*ptQuad)(2)); // Erreur
17     return 0;
18 }
```



# NULL et les appels de fonction

## Attention

La constante NULL a deux valeurs possibles : (void \*)0 ou 0, le choix étant laissé aux différents systèmes.

## Problème

Ici 0 est un Int, non pas l'adresse invalide

## Pointeur comme argument

```
1 void affiche(char *chaine) {  
2     if (chaine != NULL){ do; }  
3 }  
4  
5 void (*pf)() = &affiche;  
6  
7 (*pf)(NULL); /* Faux. */  
8 (*pf)(0); /* Faux. */  
9 (*pf)((char*)0); /* Ok. */
```

## Fonction à nombre variable d'arguments

# Fonctions à arguments variables

## Définition

Une fonction à nombre d'arguments variables est une fonction qui peut recevoir entre 1 et n arguments qui peuvent être de types différents.

## Syntaxe

valeurRetour nomFonction(type1 param1, type2 param2, ...);

- ❶ Il doit impérativement exister un argument fixe
- ❷ Le symbole '...' doit être le dernier de la liste

## Pointeur comme argument

```
1 void afficheSuite(int n, ...);  
2  
3 afficheSuite(1, 20);  
4 afficheSuite(9, 20, 30, 40, 50, 60, 70, 80, 90, 100);  
5 afficheSuite(10);
```

## <stdarg.h>

### <stdarg.h>

La bibliothèque <stdarg.h> donne accès aux fonctions utilisées pour manipuler les arguments variables (Variable Arguments en anglais).

## <stdarg.h>

va\_list

va\_list est un type de liste qui peut contenir des arguments.

### Syntaxe

```
1 va_list ap;
```

va\_start

void va\_start(va\_list l, type dernierArgumentFixe) permet de ranger dans une va\_list les arguments variables qui ont été donné à la fonction et d'initialiser un indicateur de parcours au début.

### Syntaxe

```
1 va_start(ap, monDerneirArgumentFixe);
```

## &lt;stdarg.h&gt;

va\_arg

type1 va\_arg(va\_list l, type type1) renvoie le prochain argument de la liste l (d'après son indicateur de parcours) comme s'il était de type 'type1'.

## Syntaxe

```
1 monEntier = va_arg(ap, int);
```

va\_end

void va\_end(va\_list l) met fin au parcours des arguments de la va\_list l. Son appel est obligatoire !

## Syntaxe

```
1 va_end(ap);
```

## &lt;stdarg.h&gt;

## Exemple

```
1  #include <stdarg.h>
2  #include <stdio.h>
3
4  void affiche_suite(int nb, ...){
5      va_list ap;
6      va_start(ap, nb);
7      int n;
8
9      while (nb > 0) {
10         n = va_arg(ap, int);
11         printf("%d.\n", n);
12         --nb;
13     }
14     va_end(ap);
15 }
16
17 int main(void){
18     affiche_suite(10, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100);
19     return 0;}
```

# A noter

## Dangers

va\_arg fait confiance au développeur. C'est à lui de faire en sorte de compter correctement et de ne pas dépasser le compte d'arguments !

## Utilisations

Il existe 3 modèles typique de fonctions à arguments variables :

- ❶ Les fonctions à chaîne de format (printf, scanf)
- ❷ Les fonctions à arguments de même type, pour gérer des arguments de nombre inconnu
- ❸ Les fonctions à pivot, pour gérer les arguments de type inconnu



# Exemple d'argument de même type

## Exemple

```
1  #include <stdarg.h>
2  #include <stddef.h>
3  #include <stdio.h>
4  #include <string.h>
5
6  static void afficheSuite(char *chaine, ...)
7  {
8      va_list ap;
9      va_start(ap, chaine);
10     do{
11         puts(chaine);
12         chaine = va_arg(ap, char *);
13     } while(chaine != NULL);
14     va_end(ap);
15 }
16
17 int main(void){
18     afficheSuite("un", "deux", "trois", (char *)0);
19     return 0;}
```

# Exemple d'argument de type different

## Exemple

```
1  #include <stdarg.h>
2  #include <stdio.h>
3  enum type { TYPE_INT, TYPE_DOUBLE }; //pas vu en cours
4
5  static void affiche(enum type typeV, ...){
6      va_list ap;
7      va_start(ap, typeV);
8
9      switch (typeV)
10     {
11     case TYPE_INT:
12         printf("entier:%d.\n", va_arg(ap, int));
13         break;
14
15     case TYPE_DOUBLE:
16         printf("double:%f.\n", va_arg(ap, double));
17         break;
18     }
19     va_end(ap);}
```

# Exemple d'argument de type different

## Exemple

```
1  int main(void){  
2      affiche(TYPE_INT, 10);  
3      affiche(TYPE_DOUBLE, 3.14);  
4  
5      return 0;  
6  }
```

# Conclusion

- Les fonctions peuvent aussi avoir des pointeurs génériques
- On peut donner un nombre variable d'arguments à une fonction avec `""...` et `stdarg.h`
- Il faut avoir un moyen de savoir ou de compter le nombre d'arguments