Information, Calcul et Communication

Composante Pratique: Programmation C++

MOOC sem4 traitée sur 2 semaines: fonction (2)

Passage par référence et surcharge

Surcharge ou valeurs par défaut des paramètres ?

Récursivité et pile

Eléments complémentaires série6 pour le projet

Méthode de conception d'une fonction en 5 étapes

Passage par référence et surcharge

Rappel: chaque appel transmet à la fonction **f** un moyen pour cette fonction de pouvoir accéder à la variable dont le nom est fourni en argument.

-> il s'agit de **l'adresse** de la variable qui permet d'accéder à la mémoire de manière transparente.

A chaque appel le paramètre formel existe à la même adresse que la variable transmise par référence.

On doit mettre en œuvre la surcharge pour pouvoir gérer des variables de type différents

.

Illustration sur le code
variable_et_reference_surcharge.cc

Ce code a un bug:

```
void f(int &val);

int main()
{
  int max(100);
  f(max);
  cout << max << endl;

  float min(0);
  f(min);
  cout << min << endl;
}

void f(int &val)
{
  val = 33;
}
```

Surcharge ou valeurs par défaut des paramètres ?

Dans le langage courant un même verbe d'action peut s'appliquer à des contextes très différents sans qu'il soit nécessaire d'ajouter des précisions sur le verbe de l'action ; par exemple: laver la vaisselle, laver ses dents.

SURCHARGE: le compilateur utilise le ***nombre et le type des paramètres*** d'une fonction pour distinguer les contextes et savoir QUELLE fonction est appelée.

USAGE A PRIVILEGIER: surcharger lorsque les contextes sont indépendants => pas ou très peu de code commun

USAGE A EVITER: beaucoup de code commun entre les différentes versions surchargées (risques de bug copier-coller, coût de maintenance)

La **VALEUR PAR DEFAUT** des paramètres d'une fonction est recommandée pour éviter la duplication de code. C'est la valeur courante du paramètre.

Justification très rare de la surcharge: efficacité du code (moins de paramètres)

Surcharge ou valeurs par défaut des paramètres ? (2)

La limitation de l'approche de la VALEUR PAR DEFAUT des paramètres justifie parfois de mettre en œuvre l'approche par SURCHARGE.

Règle: les paramètres avec une valeur par défaut sont EN DERNIER (à droite)
car le compilateur *initialise les arguments effectifs de la Gauche vers la Droite*

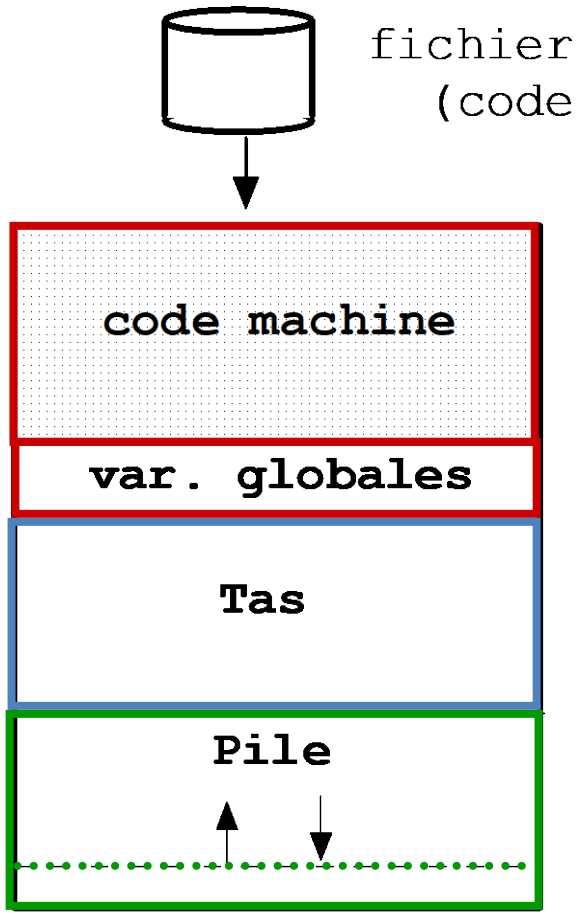
Exemple de **prototype**: void **init_produit**(int **ref**, int **nb**, double **prix=0**., int **ref_fournisseur=0**);

Remarque : c'est le seul contexte où **l'entête de la fonction** doit être différente du **prototype**
=> on ne montre PAS les valeurs par défaut dans l'en-tête d'une fonction

Code **valpardefaut.cc**

Récursivité et pile

Organisation de la mémoire pour l'exécution d'un programme



Le système d'exploitation charge le code exécutable dans la mémoire vive, à la suite d'une zone réserveé pour lui-même

Du point de vue du programme en cours d'exécution, la mémoire est organisée en 4 zones: code,var glob, tas, pile

Le code est exécuté en commençant par la fonction main()

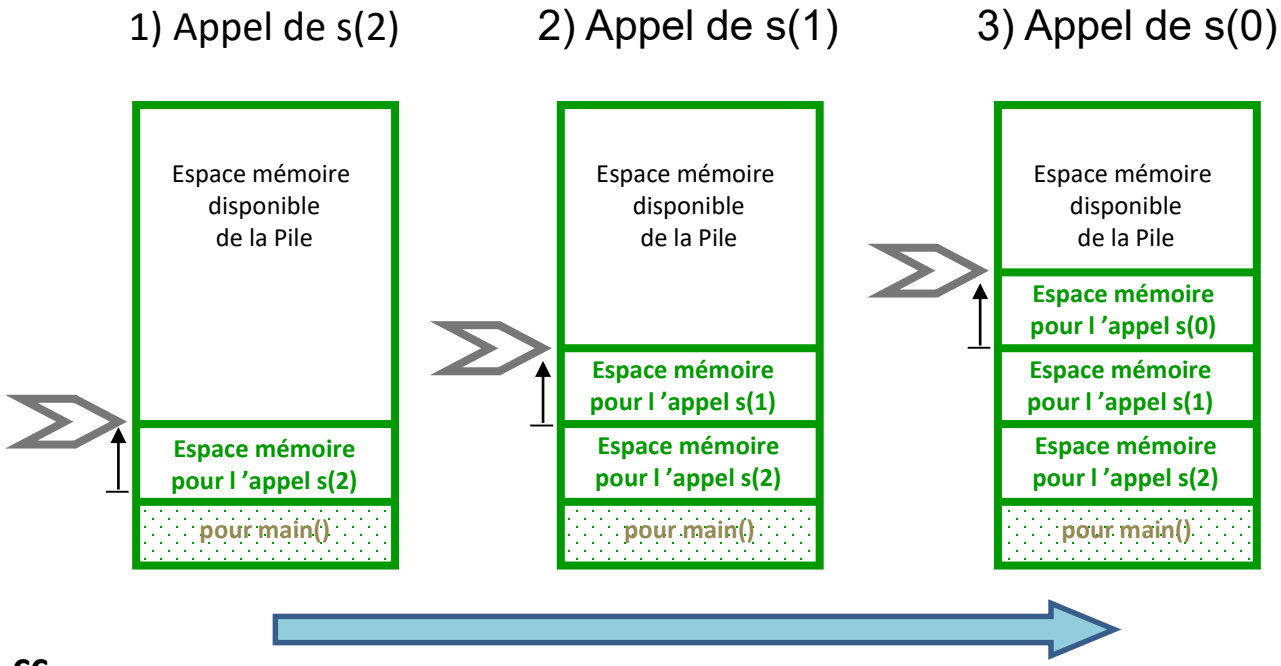
main() appelle s() qui s'appelle récursivement

Le **Pointeur de Pile** >> est mis à jour à chaque appel.

Un espace mémoire distinct existe pour n sur la pile pour chaque appel récursif.

```
 ...
main()
{
   cout << s(2)<< endl;
   ...
}

int s(int n)
{
   if (n <= 0)
      return 0;
   else
      return n + s(n-1);
}
```



Code **somme_rec_n_entiers.cc**

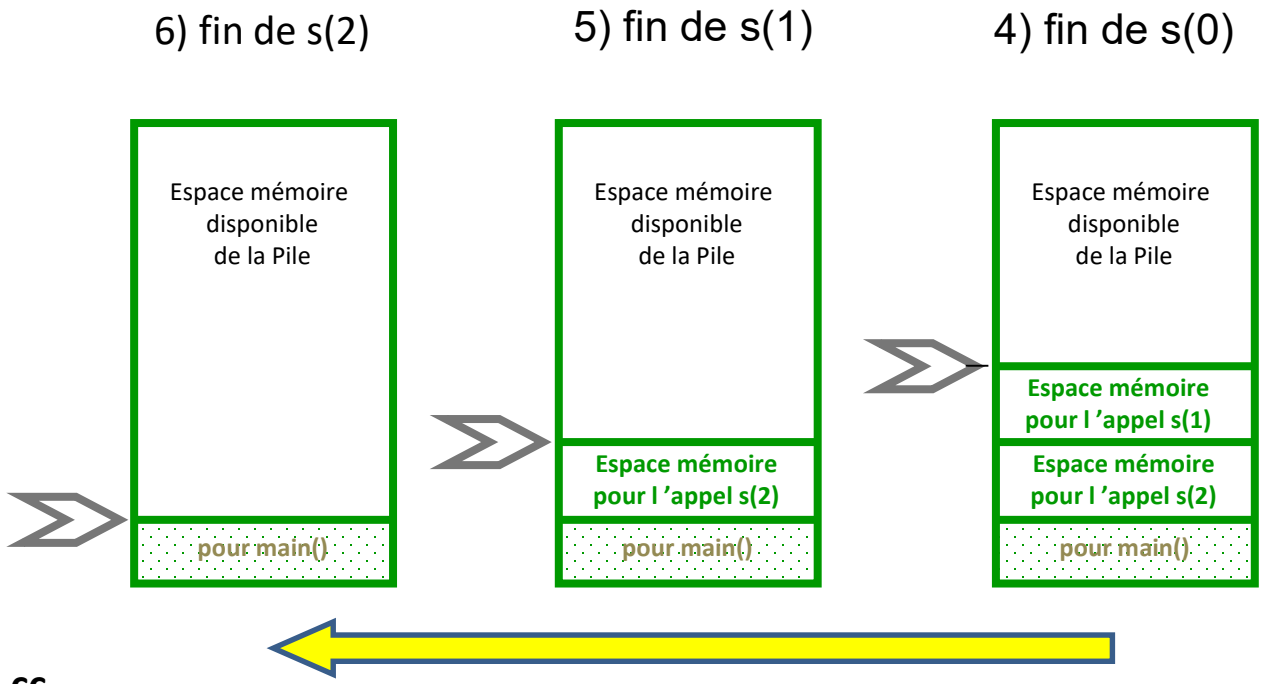
main() appelle s() qui s'appelle récursivement (2)

Le **Pointeur de Pile** >> est mis à jour à chaque appel.

Un espace mémoire distinct existe pour n sur la pile pour chaque appel récursif.

```
 ...
main()
{
   cout << s(2)<< endl;
   ...
}

int s(int n)
{
   if (n <= 0)
      return 0;
   else
      return n + s(n-1);
}
```



Code **somme_rec_n_entiers.cc**

Eléments complémentaires de la série6 pour le projet

a) Mesure de performances avec **time**

b) RappeL : boucle «système» qui effectue la **lecture** (slides semaine 4):

1) Tant qu'on n'a pas validé ce qui est tapé au clavier avec **Enter**, il n'y a rien à «lire» pour le programme
2) Dès le première validation avec **Enter**, ce qui est validé est mémorisé par les système (*buffer d'entrée*).
3) Ce qui est mémorisé par le système est extrait du *buffer d'entrée* et consommé par les appels successifs de **cin**
4) Par défaut **cin** ***filtre les séparateurs*** = les espaces, tabulation, Enter sont ignorés.
5) En cas d'échec de lecture d'une donnée le caractère fautif reste dans la mémoire système pour le prochain appel de cin (slides semaine 4)

Problème: comment faire pour lire les séparateurs avec **cin** ? (exercice «**codage de César**»)

➔ demander explicitement chaque caractère:

```
char c;
cin.get(c);
```

Eléments complémentaires de la série6 pour le projet

c) **Redirection** de l'entrée :

- La mise au point d'un projet demande beaucoup de tests
- Chaque **test** correspond à un scénario d'exécution pour lequel on connait le résultat
- Le projet effectue des manipulations sur un ensemble de données alphanumériques
- L'entrée manuelle des données des tests consomme un temps précieux

- La **redirection de l'entrée** permet de diriger le contenu d'un fichier texte vers l'entrée par défaut
  - AUCUNE ligne de code à changer
  - AUCUNE donnée à entrer au clavier

- **Méthode** :
  - Utiliser les fichiers de test fournis et créer les siens avec geany
  - Effectuer la redirection depuis le terminal en mode ligne de commande

Méthode de conception d'une fonction en 5 étapes

1) Le QUOI : que doit faire la fonction ? Quel est son but ? **prototype**
   - ➔ Détermine son **nom**
   - ➔ Ne PAS se soucier du COMMENT à ce stade

2) Avec quelle matière première ? Quels paramètres ?
   - ➔ choisir un **nom** qui exprime sa nature/ son utilité

3) Pour chaque paramètre: doit-il être modifié ou pas ? Si pas de modif :
   - ➔ privilégier le passage par valeur pour paramètres scalaires
   - ➔ privilégier le passage par référence constante pour tableaux

4) La fonction est-elle utilisée dans une expression ? Ex: `y = f(x);`
   - ➔ Renvoie-t-elle une valeur?
   - ➔ Si oui, de quel type ? Si non, elle est de type `void`. **appel**

5) Le COMMENT : on commence seulement l'analyse fine de la fonction quand son prototype est clairement défini = son contrat avec le monde extérieur.

definition