Eléments d'Informatique

Cours11 – Allocation dynamique, listes chaînées

Catherine Recanati

UNIVERSITÉ PARIS 13 NORD

Plan général

- **Représentation des nombres. Notion de variable.**
- **Programme. Expressions.**
- **Architecture des ordinateurs: langage machine, langage assembleur, AMIL.**
- **Systèmes d'exploitation : fichiers, processus, compilation.**
- **Instructions de contrôle: boucles et branchements.**
- **Programme. Définition de fonction. Appel fonctionnel.**
- **Tableaux de variables et fonctions d'arguments de type tableau.**
- **Sens d'un programme, pile d'exécution, compilation.**
- **Pointeurs et tableaux.**
- **Chaines de caractères, bibliothèque <string.h>.**
- **Allocation dynamique, liste chaînées.**
- Révisions.

Notes du partiel P1:

- Elles sont affichées.
- Si vous voulez voir votre copie, il n'y a actuellement qu'une seule possibilité :

demain (mardi) dans mon bureau B213

de 10h30 à 13h30,

de 14h30 à 17h15

Ensuite, je ne serai plus disponible avant début janvier

Programme du partiel P2:

Programme du premier partiel + les fonctions et les tableaux statiques d'entiers à une seule dimension (surtout les cours de CM5 à CM9 en sus de CM1, et CM2 principalement).

Il y aura à coup sûr plusieurs exercices avec des fonctions d'argument de type tableau (TD7, TD8 et TD9).

Ceux qui traiteront ces exercices en utilisant des pointeurs auront des points de bonus supplémentaires.

Programme détaillé du partiel P1:

- Codage d'un entier, d'un réel rationnel, d'un caractère. Variable, types simples de variable, identificateur de variable, valeur, adresse de variable et affectation.
- une certaine familiarité avec la notation BNF et, pouvoir reconnaître une expression, une définition ou déclaration de fonction et un appel de fonction.
- Vous devrez aussi maîtriser le sens des instructions de contrôle (en particulier des boucles) et la structure syntaxique d'un programme (bloc, main, déclaration/définition).

Programme du partiel P1 (suite):

- Compilation: comprendre dans les grandes lignes ce que fait le compilateur, et être capable de distinguer différentes sortes d'erreurs détectées par le compilateur selon les 5 étapes de la compilation (1.préprocesseur, 2.analyse lexicale, 3.analyse syntaxique, 4.analyse sémantique, 5.éditeur de liens).

=> Mais vous n'avez pas à connaitre dans les moindres détails les cours CM3, CM4 (même chose pour CM8). Ces cours sont destinés à vous aider à comprendre le fonctionnement du compilateur.

Nouveaux ajouts pour P2 :

- fonctions
- tableaux statiques à une dimension

(TD7, TD8 et TD9) + les TP correspondants.

Pour les pointeurs et les tableaux :

- pointeur comme adresse (cours CM2, etc. jusqu'à CM9)
- CM10 et CM11 ne sont pas à proprement parler au programme, au sens où vous n'êtes pas obligé de maitriser les pointeurs et le parcours de tableau avec des pointeurs. (les points du sujet accordés pour cette maitrise s'il y en a, seront des points bonus supplémentaires au barème).

Points de CC:

- Votre note de CC sera élaborée à partir des 3 QCC.
- Il y aura peut-être des points de présences accordés pour améliorer votre moyenne à ces 3 QCC.
- Il n'y aura pas de QCC cette semaine.
- aucun point de présence ne sera compté pour cette semaine pour améliorer votre note de CC, mais les feuilles de présences seront transmises comme il se doit à l'administration.

- Cours 11 – Allocation dynamique, listes chaînées

- Allocation dynamique
- le type complexe struct
- types complexes
- listes chaînées

void * malloc(size_t size);

La fonction `malloc` alloue *size* bytes de mémoire et retourne un pointeur sur l'espace alloué. On force alors la conversion de ce pointeur par le type des données souhaitées.

Exemple:

```
typ *pt = (typ*) malloc(n*sizeof(typ))
...
free (pt);  /* libère l'espace alloué
               précédemment par malloc */
```

void * malloc(size_t size);

```
int main() {
   char* tab[]; // sa taille n'est pas fixée
   tab = (char*)malloc(10*sizeof(char));
//  la taille de tab est maintenant fixée
// à 10, et sa valeur (constante) aussi.
   tab[0] = 1;
        ...
   tab[9] = 567;
}
```

Types complexes

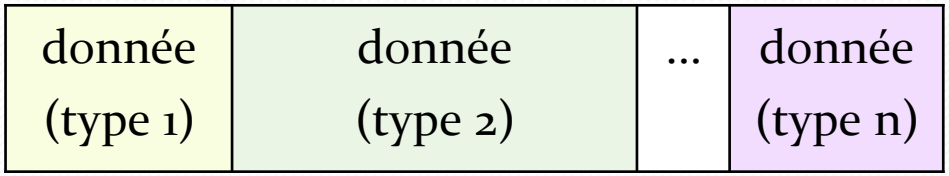
Le langage C propose deux types complexes permettant au programmeur d'utiliser d'autres types construits à partir des types primitifs :

- le type tableau qui peut contenir un certain nombre de données d'un même type
- le type structure qui peut contenir des données de types différents

Plan

- Un tableau est une variable composée de données de même type, stockées de manière contiguë en mémoire (les unes à la suite des autres).

- Une structure est une variable composée de données de types hétérogènes, stockées en mémoire les unes derrière les autres.



Le type struct

Déclaration d'un type de structure nommée toto.

```
struct toto {
        int val;
        char f;
        int tab[];
} ;
```

- sizeof (toto) renverra la place mémoire nécessaire à une variable de type « struct toto ».

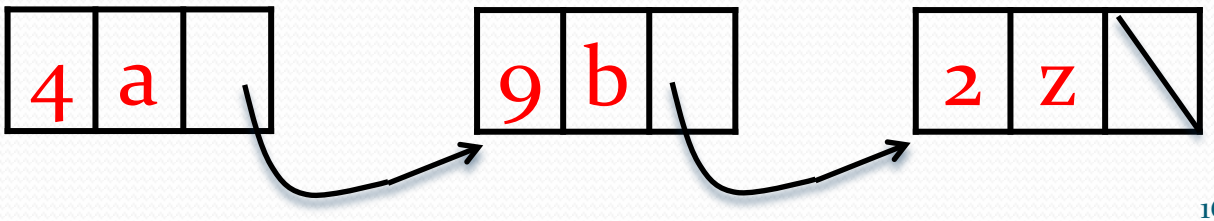
```
struct toto {
        int val;
        char f;
        int tab[];   /* ou int* tab;   */
} ;
```

On accèdera aux champs (de types divers) avec l'opérateur Point « . »

ex:
```
struct toto x; // declaration
/* la place pour une variable x
de type struct toto est réservée */
x.val= 8;    x.f = '\n';
x.tab[0]=0;
```

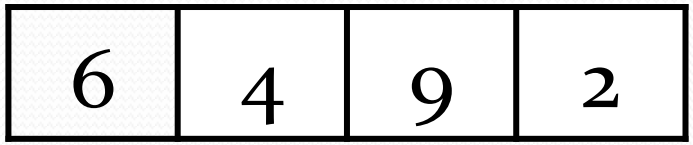
```
struct toto {
        int val;
        char f;
        struct toto* pt;
} ;
```

Grâce à une définition récursive et l'utilisation de pointeur, on va pouvoir créer des listes chainées d'éléments de type toto :

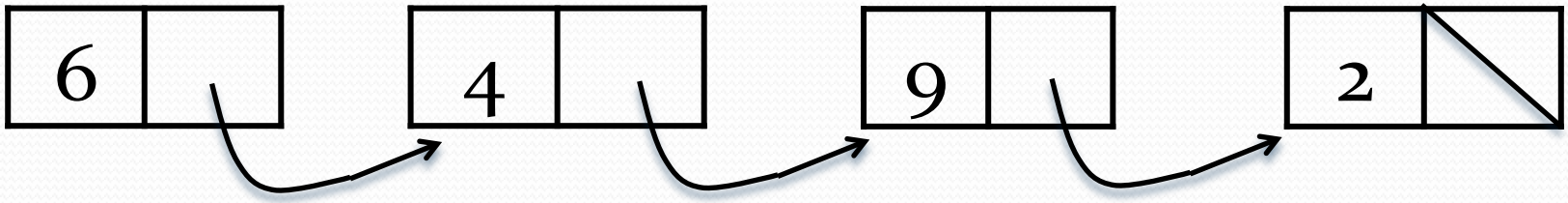


Comparaison des deux types de structures de données complexes : tableau et liste chaînée.

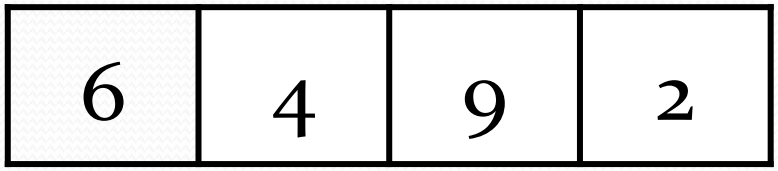
Un tableau



Une liste chaînée d'éléments (de type struct)

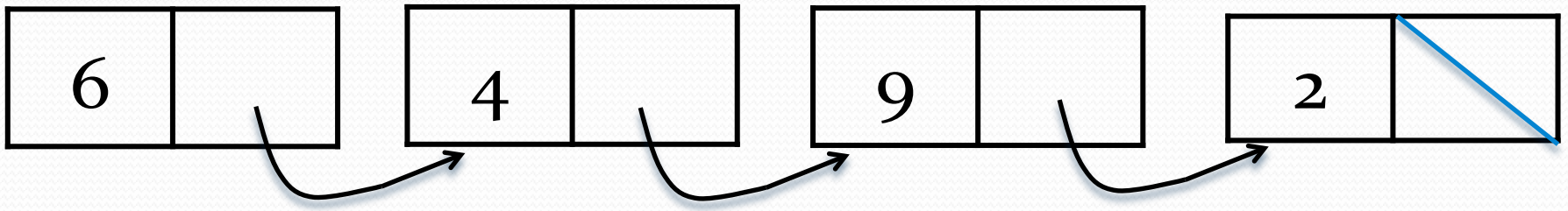


Un tableau, tab



- taille fixe (connue à la création)
- adresse du premier élément connue
- stockage continu: on peut accéder directement au i-ème élément tab[i]
- erreur d'exécution si on cherche à accéder à un élément inexistant (tab[j] avec j >= taille du tableau ou j < 0)
- supprimer ou ajouter un élément est difficile (réallocation et/ou recopie)

Une liste chaînée d'éléments



- taille inconnue (variable), limitée uniquement par la mémoire disponible
- impossible d'accéder directement à l'élément de rang i
- s'il n'y a pas d'élément suivant, l'adresse indiquée pour ce dernier sera NULL
- il est facile d'ajouter ou de supprimer un élément (sans avoir à recréer la liste)

Listes chaînées en C

Définition d'une structure appelée element. C'est une définition récursive (dite aussi auto-référentielle):

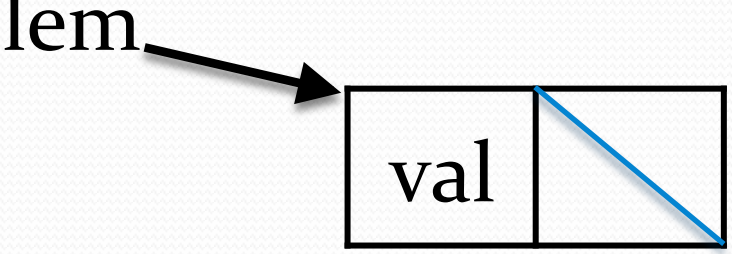
```
struct element {
        int valeur;
        struct element *suivant;
} ;
```

Une liste chaînée sera un pointeur sur une structure de ce type.

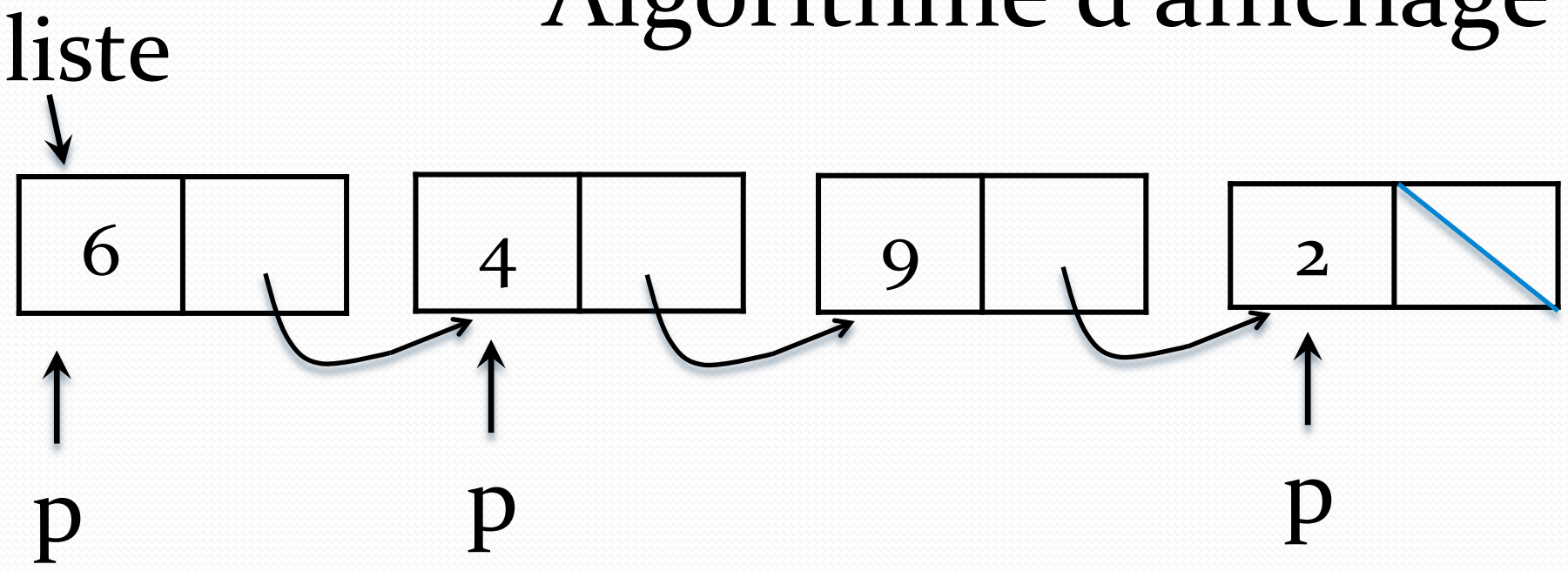
```
struct element {
        int valeur;
        struct element *suivant;
} ;
typedef struct element element;
typedef    element*      liste;

// trois déclarations équivalentes
struct element*    liste1 = NULL;
element*           liste2 = NULL;
liste              liste3 = NULL;
```

Fonction de création (d'une liste) d'un élément

```
element*  newElement (int val) {
    element*  nouvElem
              = malloc (sizeof (element));
    nouvElem->valeur = val;
    nouvElem->suivant = NULL;
    return nouvElem;
}
```



Algorithme d'affichage



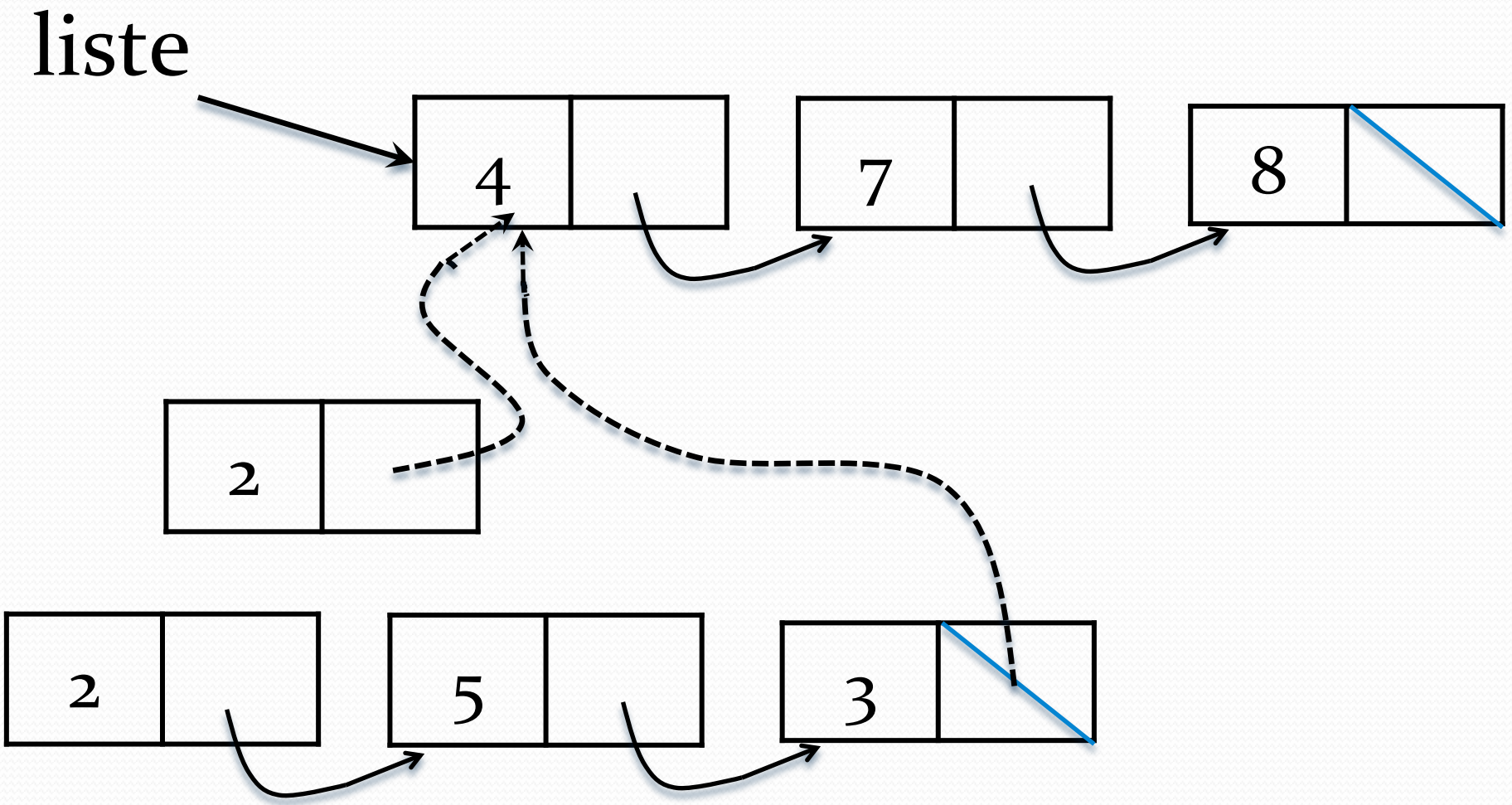

```
printf("(%d", p->valeur);          // ici: (6
while ( (p = p->suivant) != NULL)
      printf(", %d",  p->valeur); // puis: , 4
printf(")\n");
```

```
void affiche (liste list)    {
  element * p  = list;

  if (p == NULL) {          // on affiche "()"
                printf("()");
                return  ;
  }
  printf("(%d", p->valeur);          // "(" et 1er caract.
  while ( (p = p->suivant) != NULL)
          printf(", %d",  p->valeur); // les suivants...
  printf(")\n");                     // et pour terminer ")"
  return;     }
```

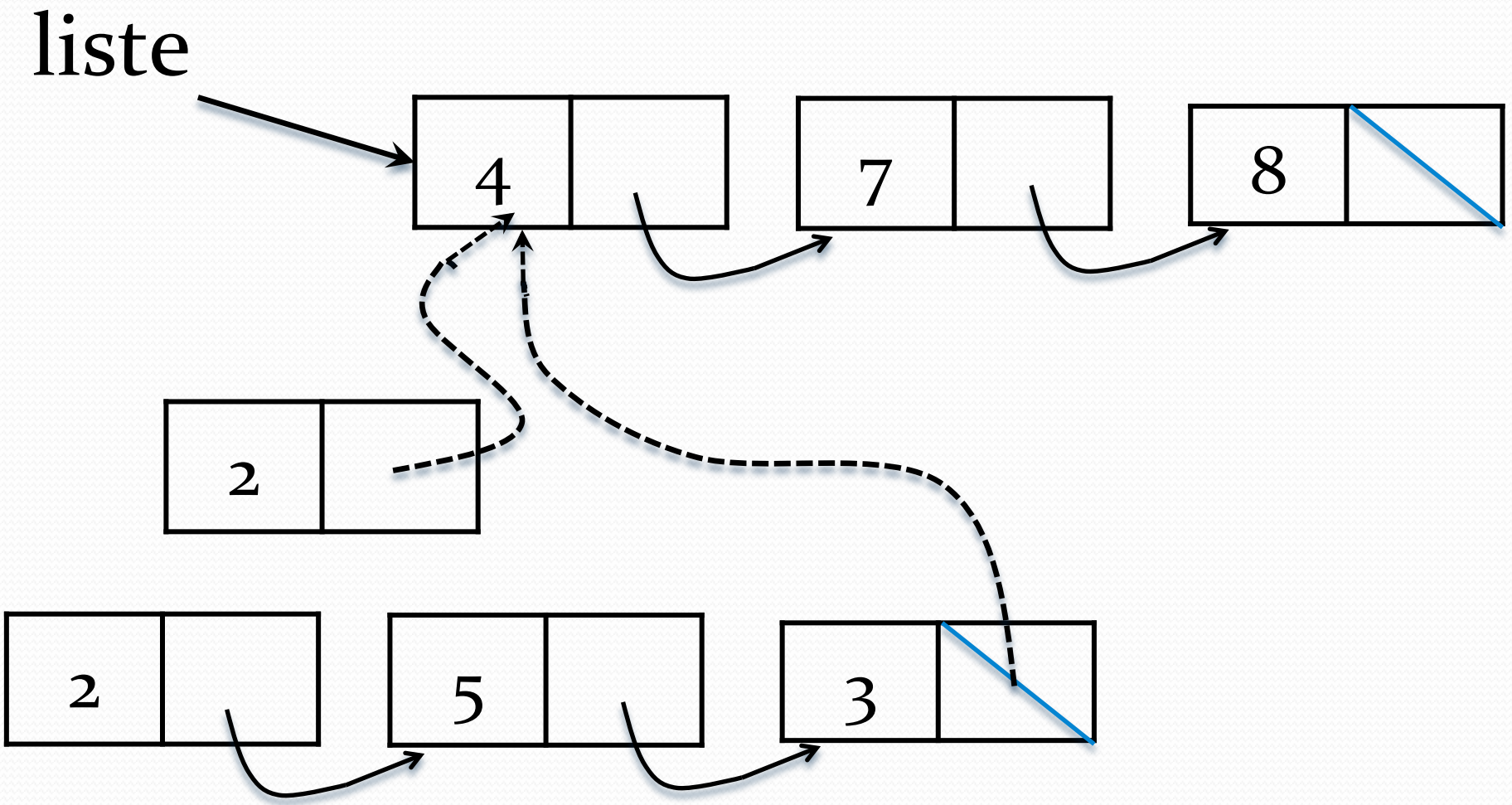
Suite du programme ...

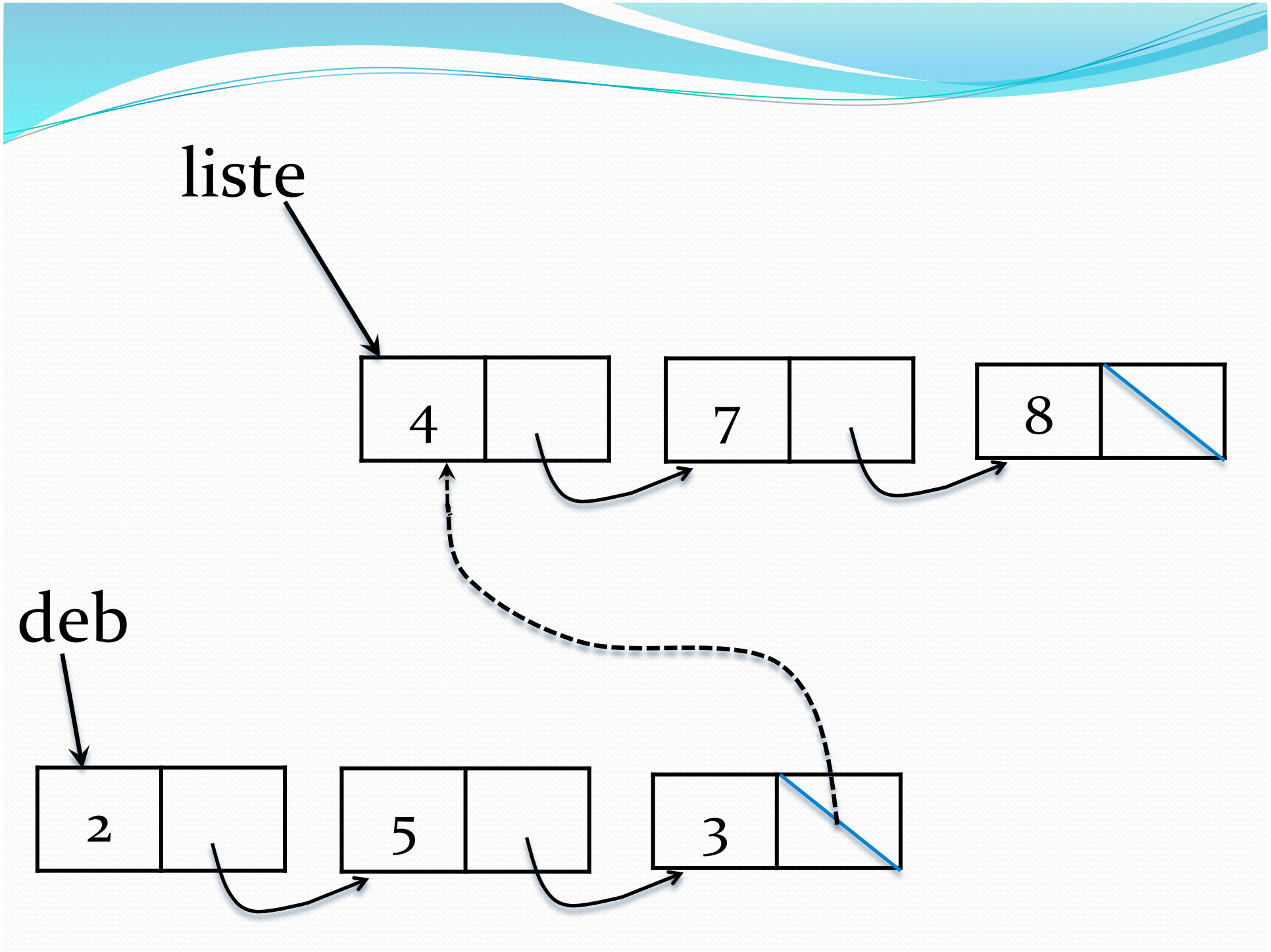
- insertion d'un élément en tête
- insertion d’un élément en queue
- ...
- recherche d'un élément
- suppression d’un élément

Ajouter un élément en tête de liste



Ajouter un élément en tête de liste




liste
4
7
8
deb
2
5
3

Plan

Merci pour votre attention !

Des questions ?