Programmation en C et Méthodes Numériques

Structures et Listes

T. Dietenbeck
thomas.dietenbeck@sorbonne-universite.fr

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Exemple

Parking de véhicules

On souhaite créer un système de gestion de véhicules dans un parking avec les contraintes suivantes. Pour les véhicules, on souhaite pouvoir

- consulter leur marque et année de fabrication
- calculer leur âge

Le nombre de places du parking sera limité. De plus, on souhaite pouvoir

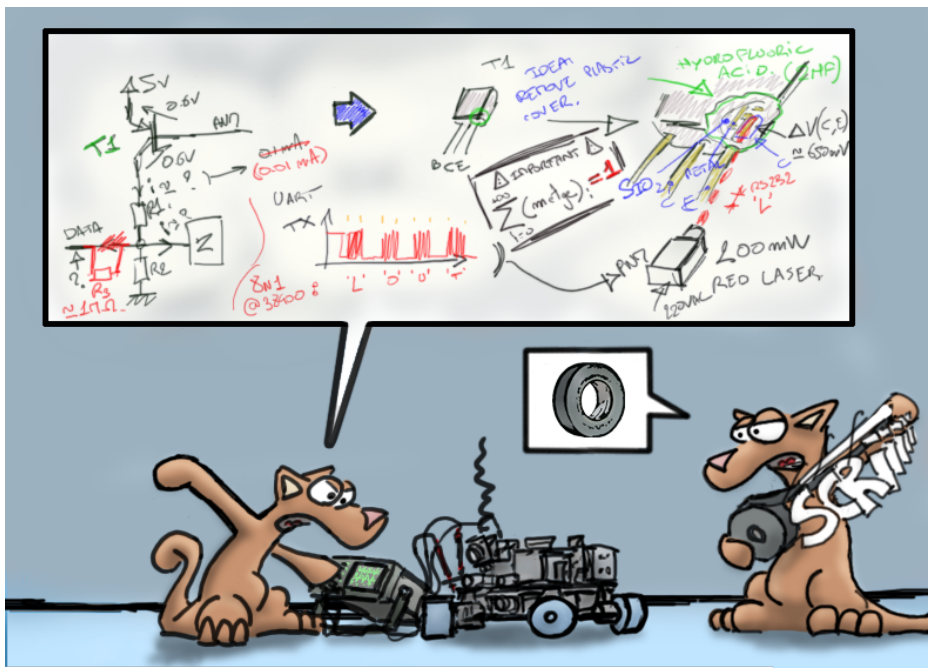
- ajouter ou retirer un véhicule (si cela est possible)
- afficher le nombre de véhicules présents dans le parking ainsi que leurs informations

Modélisation

- un entier N pour conserver le nombre maximum de véhicules
- un tableau de chaînes de caractères de taille N pour la marque
- un tableau d'entiers de taille N pour l'année de fabrication
- un entier pour conserver le nombre de véhicules présents

Limitations

- Pour les véhicules
  - Certaines variables sont liées implicitement
  - ex : la marque et l'année de fabrication d'un véhicule
- Pour le parking
  - Gros impact mémoire même si le parking est vide
  - Suppression d'un véhicule "au milieu" : faut-il laisser une case vide ou décaler tous les véhicules après ?



SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Structures

Principe

- Une structure vise à palier 2 problèmes liés aux tableaux et aux types primitifs
  - regrouper en un même endroit plusieurs données (*e.g.* la marque et l'année de fabrication d'une voiture)
  - permettre de stocker des variables de types différents (*e.g.* `int` et `char`).
- Contrairement aux types vus jusqu'à présent, il s'agit de types "personnels" qu'il va falloir définir.

[Alan Perlis]

"Un programme sans boucle et sans structure de donnée ne vaut pas la peine d'être écrit."

SCIENCES SORBONNE UNIVERSITÉ

Déclaration

Syntaxe en C

- Par convention, une structure est déclarée dans un fichier header (.h).
- **Mot-clé :** `struct`

```
struct NomStructure {
  type1 champ1;
  type2 champ2;
  ...
};     // Attention: le ; est obligatoire!
```

SCIENCES SORBONNE UNIVERSITÉ

Déclaration

Vocabulaire

Les données stockées dans une structure sont appelées des champs.

Remarques

- Le nom des champs répond aux mêmes critères que les noms de variables
- Deux champs d'une même structure ne peuvent avoir le même nom
- Les données peuvent être de n'importe quel type hormis le type de la structure dans laquelle elles se trouvent

SCIENCES SORBONNE UNIVERSITÉ

Exemple de déclaration

Les véhicules

```
struct Vehicule {
  int annee;
  char marque[30];
};
```

Remarque : `char marque[30]` déclare un tableau de 30 caractères (*i.e.* une chaîne de caractères)

Les personnes

```
struct Personne {
  char nom[20];
  char prenom[30];
  int annee;
  struct Vehicule voiture;  // Ok si struct Vehicule est declaree
};                          //        avant struct Personne
```

SCIENCES SORBONNE UNIVERSITÉ

Accès aux champs

Syntaxe

- L'accès aux champs d'une structure se fait avec l'opérateur "."

```
  // Declaration d'une variable de type NomStructure
struct NomStructure maStructure;
...
  // Acces a un champ de maStructure par "."
maStructure.champ1 = valeur;
```

SCIENCES SORBONNE UNIVERSITÉ

Accès aux champs

Exemple d'utilisation

```
struct Vehicule unVehicule;
unVehicule.annee = 2015;
unVehicule.marque = "Peugeot";
printf( "Vehicule %s de %d\n", unVehicule.marque, unVehicule.
    ↪ annee );

struct Personne unePersonne;
unePersonne.nom = "Norris";
unePersonne.prenom = "Chuck";
unePersonne.voiture = unVehicule;
```

SCIENCES SORBONNE UNIVERSITÉ

Structures et fonctions

Structures et fonctions

- Certaines fonctions peuvent avoir besoin d'une structure comme paramètre d'entrée ou de sortie (*e.g.* la fonction `calculerAge`)

Structure en entrée d'une fonction

- Comme pour tout autre paramètre, on donne le type et le nom du paramètre
- Remarque : on est obligé de rappeler le mot clé `struct` en plus du nom de la structure

```
type nomMethode( struct NomStructure maStructure, ... ) {
  // Utilisation usuelle de la structure
  maStructure.champ1 = valeur;
  ...
}
```

SCIENCES SORBONNE UNIVERSITÉ

Structures et fonctions

Structures et fonctions

- Certaines fonctions peuvent avoir besoin d'une structure comme paramètre d'entrée ou de sortie (*e.g.* la fonction `calculerAge`)

Structure comme résultat d'une fonction

- Comme pour tout autre résultat, on donne le type et on termine la fonction par le mot clé `return` et la structure résultat.
- Remarque : on est obligé de rappeler le mot clé `struct` en plus du nom de la structure

```
struct NomStructure nomMethode( ... ) {
  struct NomStructure maStructure;
  ...
  return maStructure;
}
```

SCIENCES SORBONNE UNIVERSITÉ

Modification d'une structure dans une fonction

Structure et pointeur

Comme pour les autres paramètres, si on souhaite modifier une structure dans une fonction, il faut passer son adresse (*i.e.* utiliser un pointeur)

```
// maStructure est un pointeur de structure NomStructure
type nomMethode( struct NomStructure *maStructure, ... ) {
  // Utilisation de la structure
  (*maStructure).champ1 = valeur;
  ...
}
```

Attention : en plus de précéder le nom de la structure d'une "*", on l'entoure aussi de "( )".

Exemples

Exemples

```c
// Fonction utilisant une structure sans la modifier
int calculerAge( struct Vehicule monVehicule ) {
  // Utilisation usuelle de la structure
  return 2015 - monVehicule.annee;
}

// Fonction renvoyant une structure
struct Vehicule creerVehicule( int annee, char* marque ) {
  // Declaration d'une variable de type vehicule
  struct Vehicule monVehicule;
  // Utilisation usuelle de la structure
  monVehicule.annee = annee;
  monVehicule.marque = marque;
  // Renvoie du resultat
  return monVehicule;
}

// Fonction modifiant une structure
void setAnneeVehicule(int annee, struct Vehicule *monVehicule){
  (*monVehicule).annee = annee;
}
```

SCIENCES SORBONNE UNIVERSITÉ

Quelques raccourcis en C

Le mot clé typedef

- Le mot-clé `typedef` permet de définir un raccourci pour un type existant

```
typedef typeTresLongQuOnSouhaiteRaccourcir typePlusCourt;
```

- À la compilation, le mot `typePlusCourt` sera automatiquement remplacé par `typeTresLongQuOnSouhaiteRaccourcir`

Le mot clé typedef et les structures

- À l'aide du mot-clé `typedef`, on peut ainsi définir un raccourci pour un type structure afin d'éviter de répéter systématiquement `struct`

```
typedef struct NomStruct NStruct;
```

- **Remarque :** `typePlusCourt` peut même être le nom de la structure :

```
typedef struct NomStruct NomStruct;
```

SCIENCES SORBONNE UNIVERSITÉ

Quelques raccourcis en C

Exemples d'utilisation de typedef

```
// Pas de changement pour la declaration de la structure!
struct NomStruct{
   ...
};
// On definit le nouveau type
typedef struct NomStruct NomStruct;


// Utilisation dans une fonction
NomStruct uneMethode( NomStruct uneStructure ) {
   ...
}

// Utilisation dans le programme principal
int  main ( ) {
   NomStruct maStructure;
   ...
}
```

SCIENCES SORBONNE UNIVERSITÉ

Quelques raccourcis en C

Autres exemples d'utilisation de typedef

```
// On peut aussi commencer par declarer le nouveau type
typedef struct NomStruct NomStruct;
// Puis la structure (sans changement)
struct NomStruct{
   ...
};


// On peut faire les 2 en meme temps
typedef struct {
   ...
} NomStruct2;
```

SCIENCES SORBONNE UNIVERSITÉ

Quelques raccourcis

Déclaration et initialisation d'une structure

- On peut affecter des valeurs aux champs d'une structure lors de la déclaration de celle-ci.
- Les valeurs seront affectées aux champs dans l'ordre de passage
- Si il y a moins de valeurs que de champs, les champs restants auront une valeur inconnue

Exemple

```
// Plutot que d'ecrire
NomStruct maStruct;
maStruct.champ1 = valeur1;
maStruct.champ2 = valeur2;
...
maStruct.champN = valeurN;

// On peut ecrire
NomStruct maStruct = { valeur1, valeur2, ..., valeurN };
```

SCIENCES SORBONNE UNIVERSITÉ

Quelques raccourcis

Accès rapide aux champs

Lorsqu'on dispose de l'adresse d'une structure (*i.e.* un pointeur), la syntaxe `(*nom).champ` peut être abrégée par `nom->champ`.

```
void uneMethode( NomStruct *uneStructure ) {
  uneStructure->champ1 = valeur;
}
```

Question : En quoi cela abbrège t'il la syntaxe ?

Quelques raccourcis

Accès rapide aux champs

Lorsqu'on dispose de l'adresse d'une structure (*i.e.* un pointeur), la syntaxe `(*nom).champ` peut être abrégée par `nom->champ`.

```
void uneMethode( NomStruct *uneStructure ) {
  uneStructure->champ1 = valeur;
}
```

En quoi cela abbrège t'il la syntaxe ?

Soient les structures suivantes :

```
typedef struct{
  int a;
} StructA;
```

```
typedef struct{
  int b;
  StructA *sA;
} StructB;
```

```
typedef struct{
  int c;
  StructB *sB;
} StructC;
```

```
typedef struct{
  int d;
  StructC *sC;
} StructD;
```

Accès au champ a depuis une variable de type `structD` :

```
(*(*(*(*uneStructD).sC).sB).sA).a = 42;
uneStructD->sC->sB->sA->a = 42;          // Solution plus lisible
```

SCIENCES SORBONNE UNIVERSITÉ

Exemples

Polynôme

Déclarer une structure représentant un polynôme de degré n sous la forme :

$$p(x) = \sum_{i=0}^{n} a_i \prod_{j=0}^{i-1} (x - c_j).$$

On conservera les centres c_i et les coefficients a_i.

SCIENCES SORBONNE UNIVERSITÉ

Exemples

Polynôme

Déclarer une structure représentant un **polynôme** de **degré** n sous la forme :

$$p(x) = \sum_{i=0}^{n} a_i \prod_{j=0}^{i-1} (x - c_j).$$

On conservera **les centres** c_i et **les coefficients** a_i.

Analyse

- polynôme : structure
- degré : entier
- centres, coefficients : tableaux de réels

SCIENCES SORBONNE UNIVERSITÉ

Exemples

Polynôme

Déclarer une structure représentant un **polynôme** de **degré** n sous la forme :

$$p(x) = \sum_{i=0}^{n} a_i \prod_{j=0}^{i-1} (x - c_j).$$

On conservera **les centres** c_i et **les coefficients** a_i.

Analyse

- polynôme : structure
- degré : entier
- centres, coefficients : tableaux de réels

Code C

```
typedef struct {
  int degre;
  float* A;    // Coefficients
  float* C;    // Centres
} Polynome;
```

SCIENCES SORBONNE UNIVERSITÉ

Exemples

Systeme d'équations (ER1 2016/2017)

On souhaite résoudre un système linéaire $A\mathbf{x} = \mathbf{b}$, où A est une matrice carrée contenant les coefficients du système, $\mathbf{x}$ est le vecteur inconnu et $\mathbf{b}$ est le second membre.
Déclarer une structure `SystemLin` contenant les informations relatives au système.

SCIENCES SORBONNE UNIVERSITÉ

Exemples

Systeme d'équations (ER1 2016/2017)

On souhaite résoudre un **système linéaire** $A\mathbf{x} = \mathbf{b}$, où A est une matrice carrée contenant les **coefficients du système**, $\mathbf{x}$ est le vecteur inconnu et $\mathbf{b}$ est le **second membre**.
Déclarer une structure `SystemLin` contenant les informations relatives au système.

Analyse

- SystemLin : structure
- A : matrice de réels
- b : vecteur de réels
- n : nombre d'équations

SCIENCES SORBONNE UNIVERSITÉ

Exemples

Systeme d'équations (ER1 2016/2017)

On souhaite résoudre un **système linéaire** $A\mathbf{x} = \mathbf{b}$, où A est une matrice carrée contenant les **coefficients du système**, $\mathbf{x}$ est le vecteur inconnu et $\mathbf{b}$ est le **second membre**.
Déclarer une structure `SystemLin` contenant les informations relatives au système.

Analyse

- SystemLin : structure
- A : matrice de réels
- b : vecteur de réels
- n : nombre d'équations

Code C

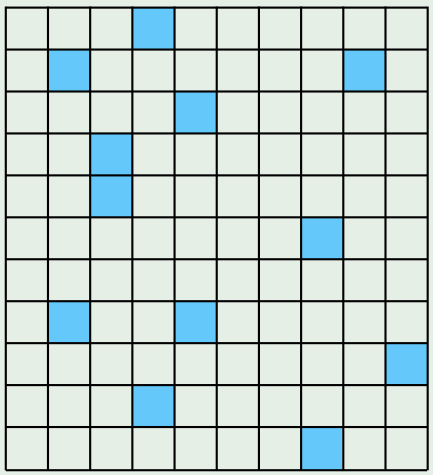
```
typedef struct {
  int n;        // Nombre d'equations et d'inconnues
  float** A;    // Matrice (carree, nxn) du systeme
  float* b;     // Second membre (vecteur 1xn)
} SystemLin;
```

SCIENCES SORBONNE UNIVERSITÉ

Exemples

Matrice sparse

Une matrice sparse est une matrice où seul un nombre limité de coefficients sont non nuls. Pour gagner en place, on ne conserve que ces coefficients.
Déclarer une structure permettant de gérer ce type de matrice.

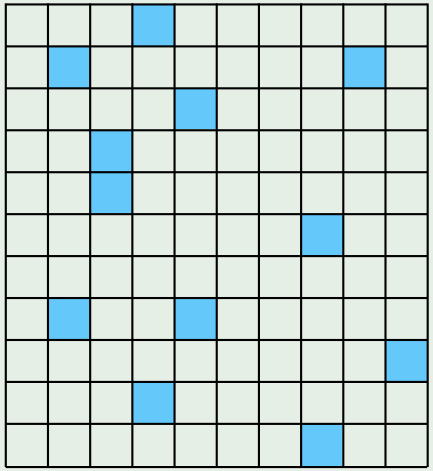


SCIENCES SORBONNE UNIVERSITÉ

Exemples

Matrice sparse

Une matrice sparse est une matrice où seul un nombre limité de coefficients sont non nuls. Pour gagner en place, on ne conserve que ces coefficients.
Déclarer une structure permettant de gérer ce type de matrice.



Code C

```c
typedef struct {
  int x, y;       // Coordonnees
  float a;        // Valeur
} Coefficient;

typedef struct {
  int dimX, dimY;  // Dimensions de la matrice
  // Coefficients non nuls
  int n;            // Nombre
  Coefficient *coef;
} MatriceSparse;
```

Exemples

Quelques erreurs

```c
struct {        // Oubli du typedef
  int unEntier;
} StructA;      // => StructA n'est pas un type

struct VehiculeB {
  int annee;
  char *marque;
};

void afficherVehiculeB( struct VehiculeB *v ) {
  // Utilisation de . au lieu de -> ou (* ).
  printf( "Vehicule fabrique en %d", v.annee );
  // Oubli des parentheses autour de *v
  printf( " de marque %s \n", *v.marque );
}

int main( ) {
  // Oubli de struct (car pas de typedef)
  VehiculeB v = { 2015, "Peugeot" };
  // Oubli de l'annee => type incompatible ( int <- String )
  struct VehiculeB v2 = { "Peugeot" };
  return 0;
}
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ

Parking de véhicules

On souhaite créer un système de gestion de véhicules dans un parking avec les contraintes suivantes. Pour les véhicules, on souhaite pouvoir

- consulter leur marque et année de fabrication
- calculer leur âge

Le nombre de places du parking sera limité. De plus, on souhaite pouvoir

- ajouter ou retirer un véhicule (si cela est possible)
- afficher le nombre de véhicules présents dans le parking ainsi que leurs informations

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ

Parking de véhicules

On souhaite créer un système de gestion de véhicules dans un parking avec les contraintes suivantes. Pour les véhicules, on souhaite pouvoir

- consulter leur marque et année de fabrication
- calculer leur âge

Le nombre de places du parking sera limité. De plus, on souhaite pouvoir

- ajouter ou retirer un véhicule (si cela est possible)
- afficher le nombre de véhicules présents dans le parking ainsi que leurs informations

Modélisation

- Une structure `Vehicule` contenant
  - un entier pour l'année de fabrication
  - un tableau de caractères pour la marque
- Une structure `Parking` contenant
  - un entier pour le nombre de places dans le parking
  - un tableau de `Vehicule` de dimension égale au nombre de places dans le parking
  - un entier pour le nombre de places occupées

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Véhicule

Header : vehicule.h

```
#ifndef _vehicule_h_  // Definition d'une bibliotheque
#define _vehicule_h_

// Inclusions des bibliotheques
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>   // Pour avoir l'annee courante

  /// Declaration d'un Vehicule
  typedef struct {
    int annee;
    char *marque;  // char* = chaine de caracteres
  } Vehicule;

  /// Prototypes des fonctions Vehicule
  Vehicule creerVehicule( int annee, char* marque );
  void afficherVehicule( Vehicule v );
  int calculerAge( Vehicule v );

#endif
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Véhicule

Implémentation : vehicule.c

```c
#include "vehicule.h"

Vehicule creerVehicule( int annee, char *marque ) {
  Vehicule v = { annee, marque };
  return v;
}

void afficherVehicule( Vehicule v ) {
  printf( "Vehicule %s de %d\n", v.marque, v.annee );
}

int calculerAge( Vehicule v ) {
  // Recuperation de la date du jour
  time_t timenow = time(NULL);
  struct tm *current = localtime(&timenow);
  int annee = current->tm_year + 1900;
  // Soustraction des annees
  return annee - v.annee;
}
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Parking

Header : parking.h

```
#ifndef _parking_h_  // Definition d'une bibliotheque
#define _parking_h_

// Inclusions des bibliotheques
#include "vehicule.h"

#define PLACES_MAX 4  // Nombre de places maximum

  /// Declaration d'un Parking avec tableau de Vehicule
  typedef struct {
    int nbVehicules;
    Vehicule *places;
  } Parking;

  /// Prototypes des fonctions Parking
  Parking creerParking( );
  void freeParking( Parking *p );
  void ajouterVehicule( Parking *p, Vehicule v );
  Vehicule retirerVehicule( Parking *p, int pos );
  void afficherParking( Parking p );

#endif
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Parking

Implémentation : parking.c

```
#include "parking.h"

  Parking creerParking( ) {
    Parking p = { 0 };    // 0 Vehicule dans le parking
    // Creation du tableau
    p.places = calloc( PLACES_MAX, sizeof( Vehicule ) );
    return p;
  }

  void freeParking( Parking *p ) {
    p->nbVehicules = 0;          // 0 Vehicule dans le parking
    free( p->places );           // Destruction du tableau
    p->places = NULL;            // Securite
  }

  void ajouterVehicule( Parking *p, Vehicule v ) {
    if( p->nbVehicules < PLACES_MAX ) { // Si il reste de la
      p->places[p->nbVehicules] = v;    // place dans le parking
      p->nbVehicules++;                 // => Ajout du vehicule
    } else
      printf( "Plus de places dans le parking\n" );
  }
  ...
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Parking

Implémentation : parking.c

```
...
Vehicule retirerVehicule( Parking *p, int pos ) {
  Vehicule res;
  if( pos < p->nbVehicules ) {
    res = p->places[pos];  // On recupere le vehicule (eg pour
        ↪ pouvoir le detruire)
    int i;          // On decale tous les vehicules
    for( i = pos; i < p->nbVehicules-1; i++ )
      p->places[i] = p->places[i+1];  // Attention a l'ordre
    p->nbVehicules--;
  }
  return res;
}

void afficherParkingTab( ParkingTab p ) {
  int i;
  printf( "%d Vehicules dans le parking\n", p.nbVehicules );
  for( i = 0; i < p.nbVehicules; i++ ) {
    printf( "Vehicule %d: ", (i+1) );
    afficherVehicule( p.places[i] );
  }
}
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Utilisation

Programme principal

```
#include "parking.h"
int main( ) {
  Vehicule v = creerVehicule( 2015, "Renault" );
  afficherVehicule( v );
  printf( "Age du vehicule: %d\n", calculerAge( v ) );

  Parking p = creerParking( );
  ajouterVehicule( &p, v );
  ajouterVehicule( &p, creerVehicule( 2014, "Peugeot" ) );
  ajouterVehicule( &p, creerVehicule( 2012, "Ford" ) );
  ajouterVehicule( &p, creerVehicule( 2010, "Ferrari" ) );
  ajouterVehicule( &p, creerVehicule( 2011, "Nissan" ) );

  afficherParking( p );
  retirerVehicule( &p, 2 );
  afficherParking( p );

  freeParking( &p );
  return 0;
}
```

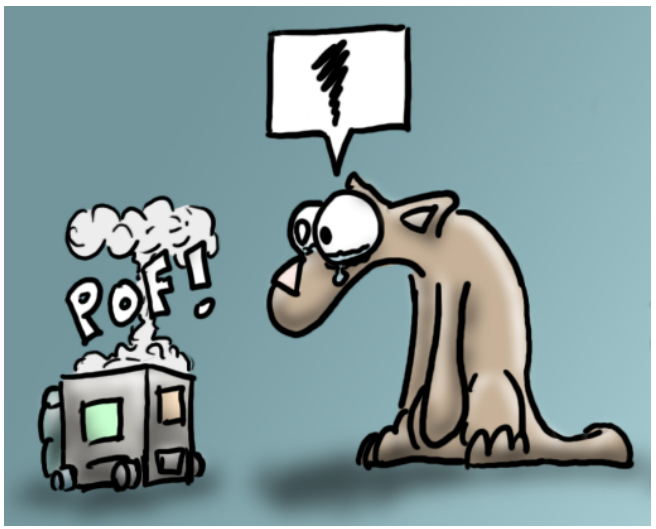
Qu'affiche le programme principal ?

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ

Remarques

- Cette modélisation permet de résoudre le problème de lien entre variables
- En revanche, les problèmes suivants persistent :
  - Encombrement mémoire du parking (tableau de 50 `Vehicule` même si une seule place est occupée
  - Suppression d'un élément



SCIENCES SORBONNE UNIVERSITÉ

Plan

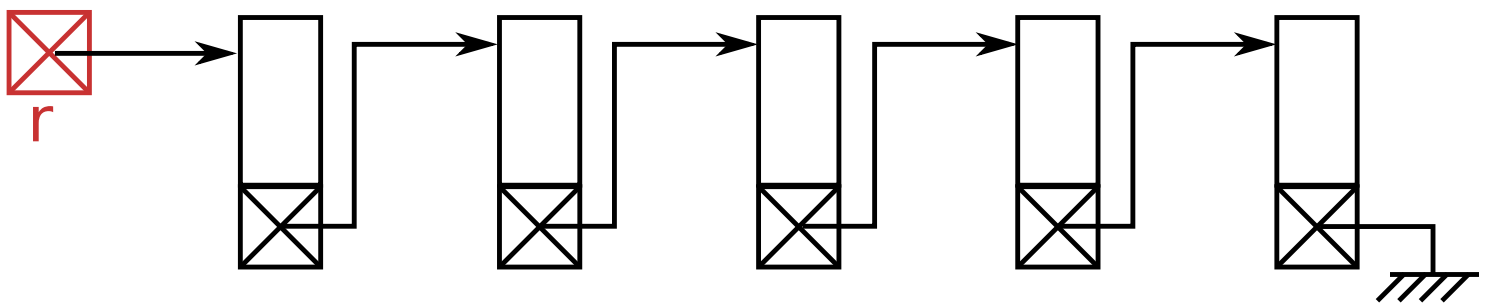
SCIENCES SORBONNE UNIVERSITÉ

Liste chainée

Définition

Une liste chainée est une suite d'éléments telle que :

- le premier élément (appelée tête ou racine) est connu par son adresse (pointeur)
- chaque élément connaît son successeur (pointeur)



SCIENCES SORBONNE UNIVERSITÉ

Manipulation de listes

Fonctions usuelles

- Création de la liste
- Destruction de la liste (suppression de tous les éléments)
- Ajout d'un élément (au moins 1 des 3)
  - au début
  - au milieu
  - à la fin
- Suppression d'un élément (au moins 1 des 3)
  - au début
  - au milieu
  - à la fin
- Parcours de la liste
  - accès à l'élément suivant
  - test de fin

SCIENCES SORBONNE UNIVERSITÉ

Manipulation de listes

Remarques

- Le code de ces fonctions est le même quelle que soit la liste, seul le nom du contenu change !
- Beaucoup de langages proposent des implémentations génériques des listes.

Problème

- En langage C, on peut implémenter
  - soit une liste spécifique à un type particulier (collection homogène) $\Rightarrow$ nécessite de réimplémenter les listes pour tous les types dont on a besoin
  - soit une liste générique pouvant contenir tous les types en même temps (collection hétérogène) $\Rightarrow$ risque d'erreurs lors de l'application d'une fonction

SCIENCES SORBONNE UNIVERSITÉ

Manipulation de listes

Problème

- En langage C, on peut implémenter
  - soit une liste spécifique à un type particulier (collection homogène) ⇒ nécessite de réimplémenter les listes pour tous les types dont on a besoin
  - soit une liste générique pouvant contenir tous les types en même temps (collection hétérogène) ⇒ risque d'erreurs lors de l'application d'une fonction

Solution retenue (pour l'UE 3E103)

- Une bibliothèque contenant l'implémentation d'une liste générique vous sera fournie (**fichiers** `listeSC.h` **et** `listeSC.c`)
- On supposera lors de manipulations de la liste que la collection est homogène
- On réimplémentera quelques fonctions selon le contenu de la liste

SCIENCES SORBONNE UNIVERSITÉ

Liste générique : Pour vous rassurer

listeSC.h listeSC.c

```c
#ifndef _listeSC_h_
#define _listeSC_h_

    // Bibliotheques "standards"
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    // Bibliotheques personnelles (a completer si besoin)


    /// Partie specifique a implementer (par les etudiants)
    /** Suppression d'une donnee dans la liste
      * @param d l'adresse de la donnee a supprimer
      */
    void freeData( void *d );

    /** Affichage d'une donnee
      * @param d l'adresse de la donnee a afficher
      */
    void afficherData( void *d );



    /// Partie generique deja implementee (par les enseignants)
    // Definition de la liste
        // Definition des types
    typedef struct Maille Maille;   // Element de la liste
    typedef Maille* pMaille;        // Pointeur vers une maille
        // Declaration de la structure Maille
    struct Maille {
        void *data;         // La donnee
        pMaille next;       // Le pointeur vers la maille suivante
    };

    typedef struct {
        int length;
        int dataSize;
        pMaille root;       // Tete de liste
        pMaille last;       // Fin de liste
        pMaille current;    // Element courant
    } ListeSC;

    // Constructeur et Destructeur
    /** Creation d'une liste
      * @return une liste prete a l'emploi
      */
```

listeSC.h

SCIENCES SORBONNE UNIVERSITÉ

Liste générique : Pour vous rassurer

listeSC.h listeSC.c

```c
#include "listeSC.h"

/// Partie specifique a implementer (par les etudiants)
/** Suppression d'une donnee dans la liste
  * @param d l'adresse de la donnee a supprimer
  */
void freeData( void *d ) {
}

/** Affichage d'un point
  * @param d la donnee a afficher
  */
void afficherData( void *d ) {
}


/// Partie generique deja implementee (par les enseignants)
// Constructeur et Destructeur
/** Creation d'une liste
  * @return une liste prete a l'emploi
  */
ListeSC* creerListe( int dataSize ) {
    // On cree la tete de liste
    ListeSC* liste = malloc( sizeof(ListeSC) );
    // On l'initialise
    liste->length = 0;   // longueur nulle
    liste->dataSize = dataSize;   // taille d'un element
    liste->root = NULL;  // aucun element
    liste->last = NULL;  // aucun element
    liste->current = NULL;  // aucun element
    // On renvoie la tete de liste
    return liste;
}

/** Suppression d'une liste
  * @param liste la liste a supprimer
  */
void freeListe( ListeSC* liste ) {
    // Tant que la liste n'est pas vide
    while( liste->root != NULL )
        suppr( liste, 0 );  // On supprime le premier element

    // On libere l'espace memoire occupe par la tete de liste
    free( liste );
    liste = NULL;
}
```

listeSC.c

listeSC.h listeSC.c

```c
// Methodes de parcours de liste
/** Methode indiquant si l'element courant a un successeur
  * @param liste la liste parcourue
  * @return 1 si le point courant a un successeur, 0 sinon
  */
int hasNext( ListeSC* liste ) {
    // Si l'element n'est pas le dernier de la liste
    return( liste->current != NULL );
}

/** Acces au successeur d'un element d'une liste
  * @param liste la liste parcourue
  */
void getNext( ListeSC* liste ) {
    liste->current = liste->current->next;
}


// Methodes d'ajout / de suppression d'element de la liste
/** Ajout d'un element dans une liste
  * @param liste la liste ou l'on veut ajouter un element
  * @param d l'element a ajouter
  * @param typeAjout type d'ajout (0: au debut, 1: au milieu, 2: en fin de liste)
  */
void ajout( ListeSC* liste, void *d, int typeAjout ) {
    if( (typeAjout < 0) || (typeAjout > 2) ) {  // Mauvaise valeur
        printf( "Erreur: typeAjout doit etre compris entre 0 et 2\n" );
    } else {
        // On declare et alloue en memoire une variable de type Maille
        pMaille tmpLP = malloc( sizeof( Maille ) );
        tmpLP->data = malloc( liste->dataSize );    // On alloue de la memoire pour les donnees
        memcpy( tmpLP->data, d, liste->dataSize );  // On les copie
        // Ajout dans la liste et chainage
        if( liste->length == 0 ) {  // Liste vide ?
            tmpLP->next = NULL;     // Pas de successeur
            liste->root = tmpLP;    // On definit la tete de liste,
            liste->current = tmpLP; // l'element courant
            liste->last = tmpLP;    // et la fin de liste
        } else {
            switch( typeAjout ) {
                case 0:             // Ajout en tete de liste
                    tmpLP->next = liste->root;  // Chainage
                    liste->root = tmpLP;        // Ajout
                    break;
                case 1:             // Ajout en milieu de liste
                    tmpLP->next = liste->current->next;     // Chainage
```

listeSC.c (suite)

SCIENCES SORBONNE UNIVERSITÉ

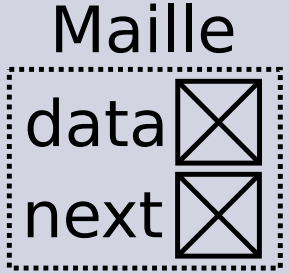
Liste générique : Implémentation existante

Maille

Un élément de la liste (ou maille) est une structure composée

- de la donnée stockée
- d'un pointeur vers la maille suivante

Remarque : Comme on n'a qu'un pointeur vers le suivant, on parle de **liste simplement chaînée**



Problèmes

Pour pouvoir être générique,

- la donnée doit pouvoir contenir n'importe quel type
- il faut connaître l'espace mémoire qu'elle occupe

SCIENCES SORBONNE UNIVERSITÉ

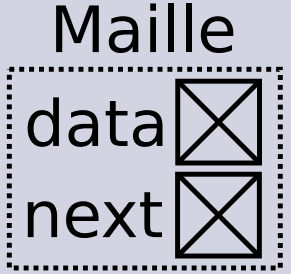
Liste générique : Implémentation existante

Maille

Un élément de la liste (ou maille) est une structure composée

- de la donnée stockée
- d'un pointeur vers la maille suivante

Remarque : Comme on n'a qu'un pointeur vers le suivant, on parle de **liste simplement chaînée**



Problèmes

Pour pouvoir être générique,

- la donnée doit pouvoir contenir n'importe quel type
- il faut connaître l'espace mémoire qu'elle occupe

Solution

- On utilise le type le plus générique : `void`
- Plutôt que de manipuler le contenu de la donnée (dont la taille varie selon la donnée), on utilise son adresse (*i.e.* un pointeur) dont la taille est fixe

SCIENCES SORBONNE UNIVERSITÉ

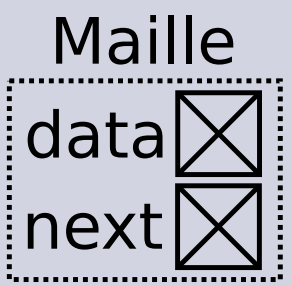
Liste générique : Implémentation existante

Maille

Un élément de la liste (ou maille) est une structure composée

- de la donnée stockée
- d'un pointeur vers la maille suivante

Remarque : Comme on n'a qu'un pointeur vers le suivant, on parle de **liste simplement chaînée**



Déclaration (dans `listeSC.h`)

```
// Definition des types
typedef struct Maille Maille;  // Element de la liste
typedef Maille* pMaille;       // Pointeur vers une maille
// Declaration de la structure Maille
struct Maille {
  void *data;                  // Pointeur vers la donnee
  pMaille next;          // Pointeur vers la maille suivante
};
```

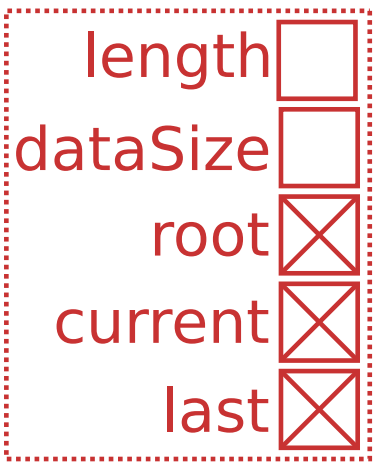
SCIENCES SORBONNE UNIVERSITÉ

Liste générique : Implémentation existante

Tête de liste

On définit une structure supplémentaire qui contient des "méta-données" de la liste (nombre d'éléments, début, fin, position courante, *etc.*)

```
typedef struct {
  int length;          // Nombre d'elements dans la liste
  int dataSize;        // Taille d'une donnee
  pMaille root;        // Tete de liste
  pMaille current;     // Element courant
  pMaille last;        // Fin de liste
} ListeSC;
```



SCIENCES SORBONNE UNIVERSITÉ

Liste générique : Implémentation existante

Constructeur et destructeur

```
ListeSC* creerListe( int dataSize );  // Creation d'une liste
void freeListe( ListeSC *liste );     // Destruction d'une liste
```

Parcours de liste

```
// Methode indiquant si l'element courant a un successeur
int hasNext( ListeSC *liste );
// Acces au successeur d'un element d'une liste
void getNext( ListeSC *liste );
```

- `hasNext` renvoie 1 si `liste->current` est différent de `liste->last` et 0 sinon
- `getNext` fait avancer `liste->current` d'une maille

Affichage de liste

```
void afficherListe( ListeSC *liste );
```

SCIENCES SORBONNE UNIVERSITÉ

Liste générique : Implémentation existante

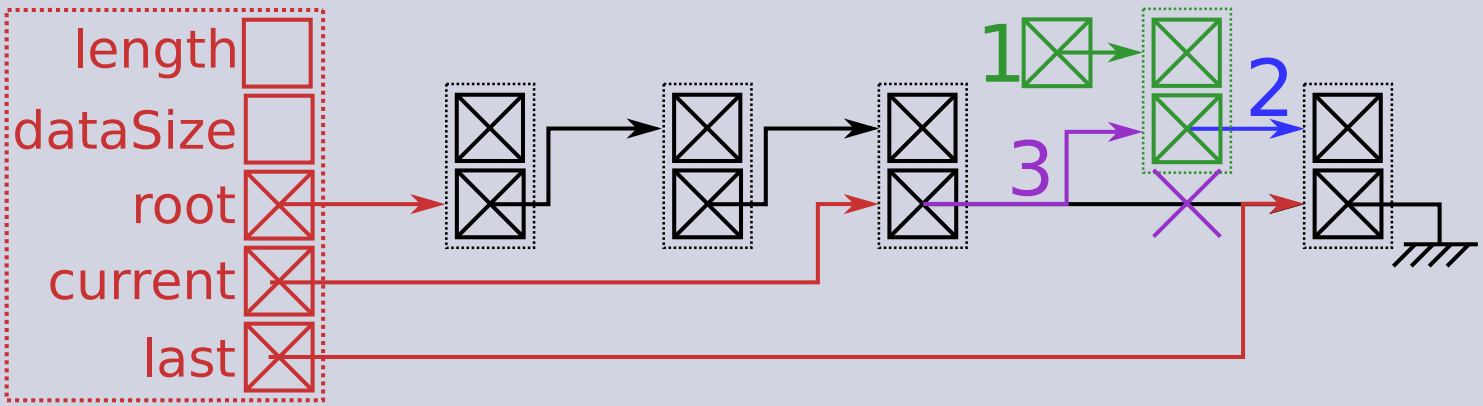
Ajout de maille

```
void ajout( ListeSC *liste, void *d, int typeAjout );
```

- `typeAjout` définit où se fait l'ajout :
  - 0 : au début de la liste
  - 1 : au milieu de la liste après l'élément courant
  - 2 : à la fin de la liste

Principe

1. On créé la nouvelle maille
2. On garantit le chaînage : `next` pointe vers le suivant
3. On ajoute dans la liste : l'élément courant (ou la tête de liste) pointe vers le nouvel élément (et la longueur de la liste est mise à jour)



SCIENCES SORBONNE UNIVERSITÉ

Liste générique : Implémentation existante

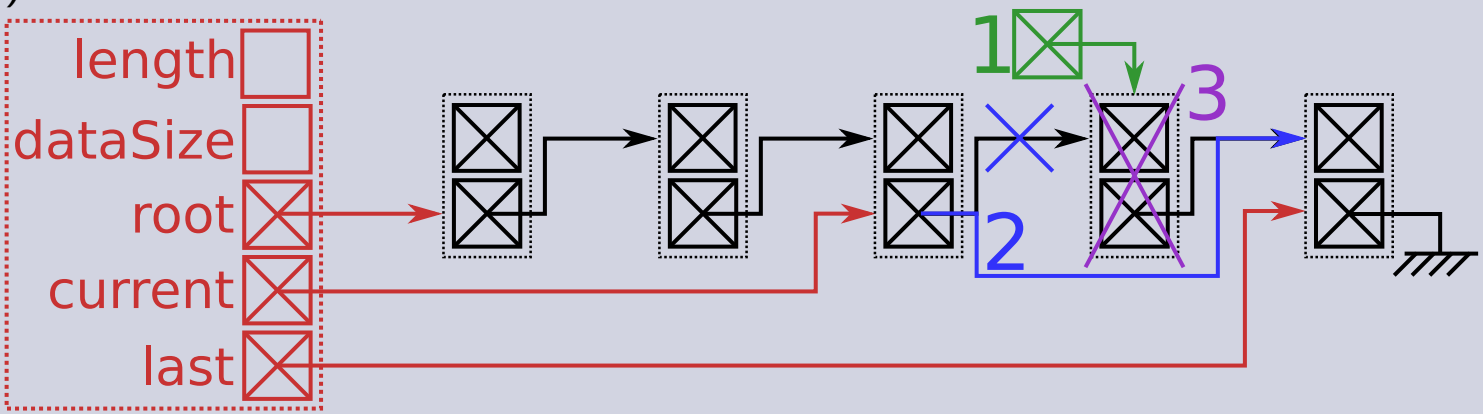
Suppression de maille

```
void suppr( ListeSC *liste, int typeSuppr );
```

- typeSuppr définit où se fait la suppression :
  - 0 : au début de la liste
  - 1 : au milieu de la liste après l'élément courant
  - 2 : à la fin de la liste

Principe

1. On récupère la maille à supprimer
2. On garantit le chaînage : l'élément courant (ou la tête de liste) pointe vers le suivant
3. On supprime de la liste : la maille est détruite (et la longueur de la liste est mise à jour)



SCIENCES SORBONNE UNIVERSITÉ

Liste générique : Ce que vous devez implémenter

Destruction des données (dans `listeSC.c`)

```
void freeData( void *d );
```

- si les données ne sont pas obtenues par allocation dynamique, on ne fait rien dans cette fonction
- sinon, on appelle le destructeur correspondant

Affichage des données (dans `listeSC.c`)

```
void afficherData( void *d );
```

- On convertit `d` depuis `void` vers le type correct
- On appelle la fonction d'affichage

SCIENCES SORBONNE UNIVERSITÉ

Quelques listes particulières : Liste doublement chaînée

Définition

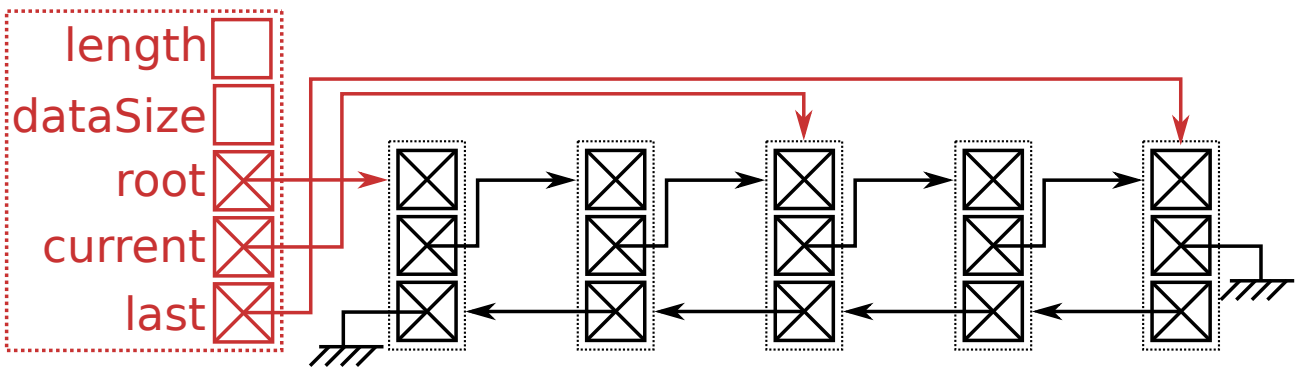
- Une liste doublement chaînée est une liste où chaque élément connaît à la fois son successeur et son prédecesseur. On peut donc la parcourir **dans les 2 sens !**

Fonctionnalités supplémentaires : parcours depuis la fin

- `int hasPrev( ListeDC* liste )` : Est on arrivé au début ? (*i.e.* `liste->current = liste->root` ?)
- `void getPrev( ListeDC* liste )` : reculer l'élément courant d'un pas

Fonctionnalités modifiées : ajout et suppression

- Mise à jour du prédécesseur **et du successeur** de l'élement ajouté / supprimé



SCIENCES SORBONNE UNIVERSITÉ

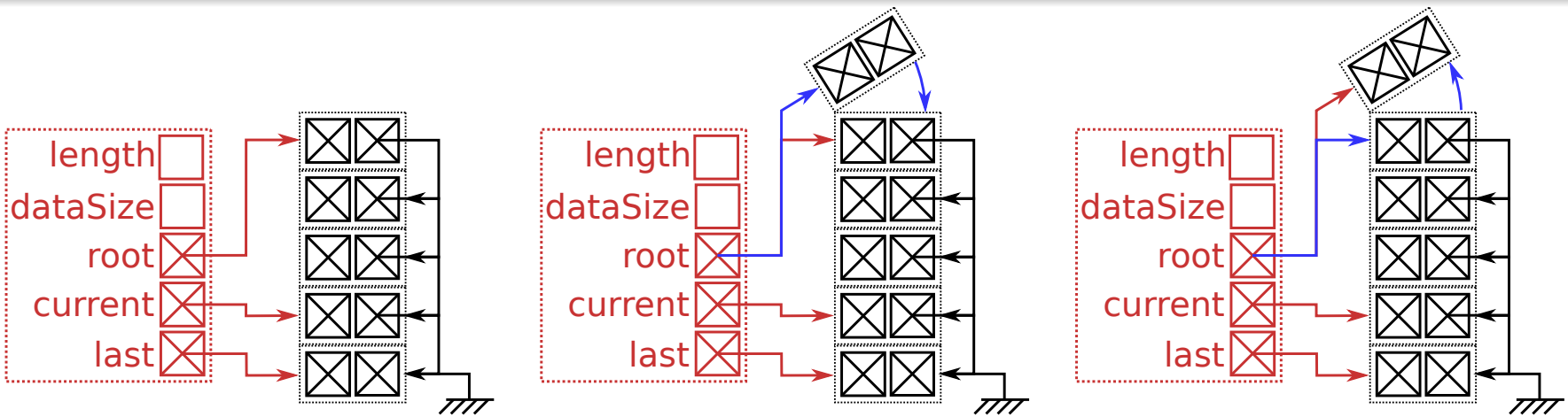
Quelques listes particulières : Pile (LIFO)

Définition

- Une pile est une liste où l'on ajoute et supprime toujours le premier élément de la liste (*e.g.* pile d'assiettes).
- 1 seule fonction d'ajout (en tête) et de suppression (en tête)

Remarque

- Une pile est parfois appelée LIFO (Last In First Out)



Ajout en tête Suppression en tête

SCIENCES SORBONNE UNIVERSITÉ

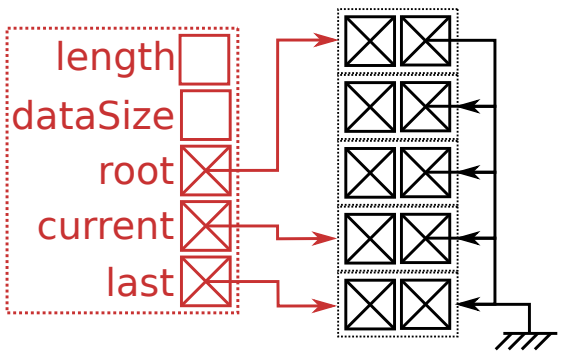
Quelques listes particulières : File (FIFO)

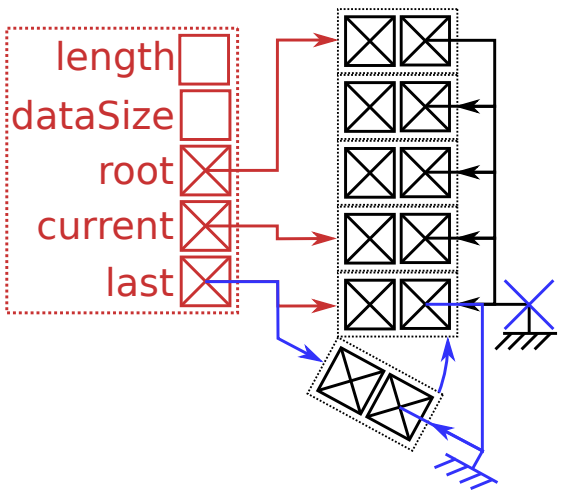
Définition

- Une file est une liste où l'on ajoute toujours à la fin et supprime toujours au début (*e.g.* file d'attente).
- 1 seule fonction d'ajout (à la fin) et de suppression (en tête)

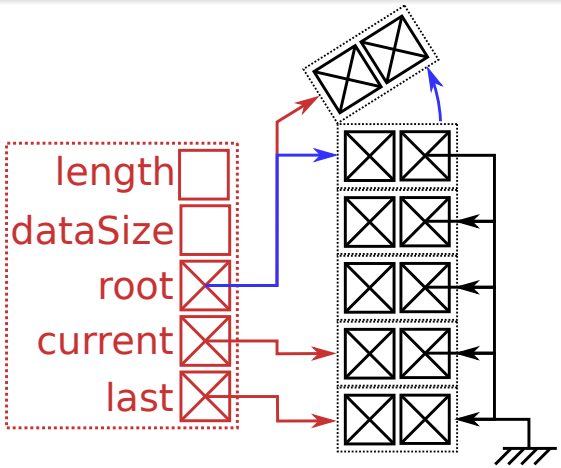
Remarque

- Une file est parfois appelée FIFO (First In First Out)





Ajout à la fin



Suppression en tête

SCIENCES SORBONNE UNIVERSITÉ

Complexité des opérations

Opération	Tableau	LSC	LDC	Pile	File
Insertion en tête	$\mathcal{O}(1)$ ou $\mathcal{O}(n)^{1,2}$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	N/A
Insertion au milieu	$\mathcal{O}(1)$ ou $\mathcal{O}(n)^{1,2}$	$\mathcal{O}(n)^3$	$\mathcal{O}(n)^3$	N/A	N/A
Insertion à la fin	$\mathcal{O}(1)^2$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	N/A	$\mathcal{O}(1)$
Suppression en tête	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Suppression au milieu	$\mathcal{O}(n)$	$\mathcal{O}(n)^3$	$\mathcal{O}(n)^3$	N/A	N/A
Suppression à la fin	$\mathcal{O}(1)$	$\mathcal{O}(n)^3$	$\mathcal{O}(1)$	N/A	N/A
Parcours	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Recherche d'un élément	$\mathcal{O}(1)$ ou $\mathcal{O}(n)^4$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$

LSC : liste simplement chaînée ; LDC : liste doublement chaînée

[1] $\mathcal{O}(1)$ si on remplace l'élément et $\mathcal{O}(n)$ si on décale tout le tableau

[2] à condition d'avoir la place dans le tableau !

[3] Nécessite de parcourir toute la liste pour arriver à l'endroit où ajouter / supprimer.

[4] $\mathcal{O}(1)$ si on connait l'index de l'élément recherché, $\mathcal{O}(n)$ sinon.

SCIENCES SORBONNE UNIVERSITÉ

Exemples de déclaration

Exemple 1 : Liste d'entiers

Dans listeSC.c

```
void freeData( void *d ) {  // Pas d'allocation dynamique
}                           // => Cette fonction ne fait rien
void afficherData( void *d ) {
  int* pEntier = (int*)d;  // Conversion de d en pointeur entier
  printf( "%d\n", *pEntier );  // On affiche la valeur pointee
}
```

Utilisation dans un programme principal

```
#include "listeSC.h"
int main( ) {
  // sizeof(int) pour passer la taille d'un entier
  ListeSC *listeEntier = creerListe(sizeof(int));  // Declaration
  // Ajout d'elements
  ajout( listeEntier, 42, 0 );       // Ajout au debut
  ajout( listeEntier, 142857, 2 );   // Ajout a la fin
  // On se place sur le 1er element de la chaine
  listeEntier->current = listeEntier->root;
  ajout( listeEntier, 1337, 1 );     // Ajout au milieu

  afficherListe( listeEntier );      // Affichage
  freeListe( listeEntier );          // Destruction
  return 0;
}
```

SCIENCES SORBONNE UNIVERSITÉ

Exemples de déclaration

Exemple 2 : Liste de Personne

Soit un fichier Personne.h contenant :

```
typedef struct {
  char nom[30], prenom[30];  // 2 tableaux statiques de
} Personne;                  // caracteres

void afficherPersonne( Personne pers );
```

Pour gérer une liste de Personne, on ajoute dans le fichier listeSC.c

```
void freeData( void *d ) {  // Pas d'allocation dynamique
}                           // => Cette fonction ne fait rien

void afficherData( void *d ) {
  // Conversion de d en pointeur vers une Personne
  Personne* pPers = (Personne*)d;
  // On appelle la fonction d'affichage d'une Personne
  afficherPersonne( *pPers );
}
```

SCIENCES SORBONNE UNIVERSITÉ

Exemples de déclaration

Exemple 3 : Liste d'Etudiant

Soit un fichier Etudiant.h contenant :

```
typedef struct {
  int numEtudiant;  // Numero d'etudiant
  float* notes;     // Moyenne de chaque UE
} Etudiant;

void freeEtudiant( Etudiant *etu );
void afficherEtudiant( Etudiant etu );
```

Pour gérer une liste d'Etudiant, on ajoute dans le fichier listeSC.c

```
void freeData( void *d ) {
  // Allocation dynamique des notes => On doit liberer la memoire
    // Conversion de d en pointeur vers un Etudiant
  Etudiant* pEtu = (Etudiant*)d;
    // On appelle la fonction de liberation d'un Etudiant
  freeEtudiant( pEtu );
}

void afficherData( void *d ) {
  afficherEtudiant( (Etudiant*)d ); // Conversion de d puis
      ↪ affichage
}
```

SCIENCES SORBONNE UNIVERSITÉ

Exemples de parcours

Boucle for

Dans une liste d'Etudiant, on cherche le major de promo

```
Etudiant* findMajor( ListeSC *liste )
  Etudiant* major = NULL;  // Resultat
  float moyenneMajor = -1;
  // Parcours de la liste
  for(liste->current=liste->root;hasNext(liste);getNext(liste)) {
    // 1/ On recupere l'element courant
    Etudiant etu = ((Etudiant*)liste->current->data);
    // 2/ On traite les donnees
      // Calcul de la moyenne
    int i;
    float moyenne = 0;
    for( i = 0; i < NB_UE; i++ )
      moyenne += etu.notes[i];
    moyenne /= NB_UE;
      // Comparaison avec le major actuel
    if( moyenneEtu > moyenneMajor ) {
      major = etu;
      moyenneMajor = moyenneEtu;
    }
  }
  return major;
}
```

Exemples de parcours

Boucle while

Dans une liste d'Etudiant, on cherche la position d'un numéro particulier

```
Etudiant* findEtudiant( ListeSC *liste, int numEtu )
  // On positionne l'element courant en tete de liste
  liste->current = liste->root;
  // Parcours de la liste
  Etudiant* res = NULL;  // Etudiant trouve?
  while( hasNext( liste ) && (res == NULL) ) {  // Fin de liste?
      ↪ ou Trouve?
    // 1/ On recupere l'element courant
    Etudiant etu = ((Etudiant*)liste->current->data);
    // 2/ On traite les donnees
    if( etu->numEtudiant == numEtu )
      res = etu;
    // 3/ On avance d'un pas
    getNext( liste );
  }
  return res;
}
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Modélisation

Parking de véhicules

On souhaite créer un système de gestion de véhicules dans un parking avec les contraintes suivantes. Pour les véhicules, on souhaite pouvoir

- consulter leur marque et année de fabrication
- calculer leur âge

Le nombre de places du parking sera limité. De plus, on souhaite pouvoir

- ajouter ou retirer un véhicule (si cela est possible)
- afficher le nombre de véhicules présents dans le parking ainsi que leurs informations

Modélisation (avant)

- Une structure `Vehicule` contenant
  - un entier pour l'année de fabrication
  - un tableau de caractères pour la marque
- Une structure `Parking` contenant
  - un tableau de `Vehicule` de dimension égale au nombre de places dans le parking
  - un entier pour le nombre de places occupées

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Modélisation

Parking de véhicules

On souhaite créer un système de gestion de véhicules dans un parking avec les contraintes suivantes. Pour les véhicules, on souhaite pouvoir

- consulter leur marque et année de fabrication
- calculer leur âge

Le nombre de places du parking sera limité. De plus, on souhaite pouvoir

- ajouter ou retirer un véhicule (si cela est possible)
- afficher le nombre de véhicules présents dans le parking ainsi que leurs informations

Modélisation (après)

- Une structure `Vehicule` contenant
  - un entier pour l'année de fabrication
  - un tableau de caractères pour la marque
- Une structure `Parking` contenant
  - une liste de `Vehicule`

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Modélisation

Modélisation

- Une structure `Vehicule` contenant
  - un entier pour l'année de fabrication
  - un tableau de caractères pour la marque
- Une structure `Parking` contenant
  - une liste de `Vehicule`

Remarques

- Les fichiers `vehicule.h` et `vehicule.c` sont inchangés
- On n'a pas besoin d'un entier indiquant le nombre de places occupées dans le parking, car cela correspond à la longueur de la liste

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Liste

Header listeSC.h

```
#ifndef _listeSC_h_
#define _listeSC_h_

// Bibliotheques "standards"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Bibliotheques personnelles (a completer si besoin)
#include "vehicule.h"

  /// Partie specifique a implementer (par les etudiants)
  /** Suppression d'une donnee dans la liste
    * @param d l'adresse de la donnee a supprimer
    */
  void freeData( void *d );

  /** Affichage d'une donnee
    * @param d l'adresse de la donnee a afficher
    */
  void afficherData( void *d );

  ...
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Liste

Implémentation `listeSC.c`

```
#include "listeSC.h"

  /// Partie specifique a implementer (par les etudiants)
  /** Suppression d'une donnee dans la liste
    * @param d l'adresse de la donnee a supprimer
    */
  void freeData( void *d ) {
    // Comme d est un Vehicule sans allocation dynamique,
    // on ne fait rien dans cette fonction
  }

  /** Affichage d'un point
    * @param d la donnee a afficher
    */
  void afficherData( void *d ) {
    Vehicule* v = (Vehicule*)d; // Cast de d depuis void vers
        ↪ Vehicule
    afficherVehicule( *v );
  }

...
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Parking

Header : parking.h

```
#ifndef _parking_h_  // Definition d'une bibliotheque
#define _parking_h_

// Inclusions des bibliotheques
#include "listeSC.h"
#include "vehicule.h"

#define PLACES_MAX 4  // Nombre de places maximum

  /// Declaration d'unParking avec liste de Vehicule
  typedef struct {     // Pas besoin du nombre de places,
    ListeSC *places;   // c'est la longueur de la liste
  } Parking;

  /// Prototypes des fonctions Parking
  Parking creerParking( );
  void freeParking( Parking *p );
  void ajouterVehicule( Parking *p, Vehicule v );
  void retirerVehicule( Parking *p, int type );
  void afficherParking( Parking p );

#endif
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Parking

Implementation : parking.c

```
#include "parking.h"

  Parking creerParking( ) {
    Parking p;  // Parking vide
    p.places = creerListe( sizeof( Vehicule ) );
    return p;   // On renvoie le parking
  }

  void freeParking( Parking* p ) {
    freeListe( p->places );
  }

  void ajouterVehicule( Parking *p, Vehicule v ) {
    if( p->places->length < PLACES_MAX )
      ajout( p->places, &v, 2 );  // Ajout a la fin
    else
      printf( "Plus de places dans le parking\n" );
  }

  ...
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple de départ : Parking

Implementation : parking.c

```
...

void retirerVehiculeListe( Parking *p, int type ) {
  // L'appel de suppr detruit les donnees
  // => rien a renvoyer
  suppr( p->places, type );
}

void afficherParking( Parking p ) {
  printf( "%d Vehicules dans le parking\n", p.places->length );
  afficherListe( p.places );
}
```

SCIENCES SORBONNE UNIVERSITÉ

Exemple : Utilisation

Programme principal

```
#include "parking.h"
int main( ) {
  Vehicule v = creerVehicule( 2015, "Renault" );
  afficherVehicule( v );
  printf( "Age du vehicule: %d\n", calculerAge( v ) );

  Parking pL = creerParking( );
  ajouterVehicule( &pL, v );
  ajouterVehicule( &pL, creerVehicule( 2014, "Peugeot" ) );
  ajouterVehicule( &pL, creerVehicule( 2012, "Ford" ) );
  ajouterVehicule( &pL, creerVehicule( 2010, "Ferrari" ) );
  ajouterVehicule( &pL, creerVehicule( 2011, "Nissan" ) );

  afficherParking( pL );
  pL.places->current = pL.places->root->next;
  retirerVehicule( &pL, 1 );
  afficherParking( pL );

  freeParking( &pL );
  return 0;
}
```

Qu'affiche le programme principal ?

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Exercice : Enoncé

Liste d'entiers

- On considère une liste simplement chaînée contenant des entiers dont la déclaration est :

```
typedef struct {
  ListeSC *liste;
} ListeEntier;
```

Questions

- Écrire une fonction qui cherche la première occurence d'un entier n. On renverra la position ou -1 si l'entier n'est pas présent.
- Écrire une fonction permettant de supprimer un élément de la liste sur 2 (en gardant le premier élément).

SCIENCES SORBONNE UNIVERSITÉ

Exercice : Enoncé

File et filtrage AR

- On s'intéresse à un filtre auto-régressif dont la sortie est donnée par :

$$s(n) = e(n) + \sum_{i=1}^{5} a_i s(n-i)$$

où e est le signal d'entrée et s le signal de sortie.

- Pour stocker les 5 échantillons du signal de sortie nécessaire au calcul de $s(n)$, on utilise une file dont la déclaration est :

```
typedef struct {
  ListeSC *echantillons;
} Signal;
```

Questions

- Écrire une fonction qui calcule la valeur de $s(n)$ connaissant $e(n)$, les 5 dernières valeurs de s et les coefficients du filtres (stockés dans un tableau)
- Écrire le programme principal qui lit les valeurs de $e(n)$ dans un fichier, calcule la valeur de $s(n)$ et l'écrit dans un autre fichier.

SCIENCES SORBONNE UNIVERSITÉ

Exercice

Expression parenthésées (CodinGame)

- On considère une expression mathématique dont on souhaite savoir si elle est bien parenthésées ou non.
- Une expression est bien parenthésée si les parenthèses (), les crochets [] et les accolades {} sont correctement appairés.
- Des caractères autres que (,), [,], {, } peuvent être présents dans l'expression
- Il n'y a pas d'espace dans l'expression

Exemples :

- L'expression $([\{\}][()])$ est bien appairée
- L'expression $([\{\})]$ n'est pas appairée

Question

- Écrire un programme permettant de résoudre ce problème