Université Claude Bernard Lyon 1

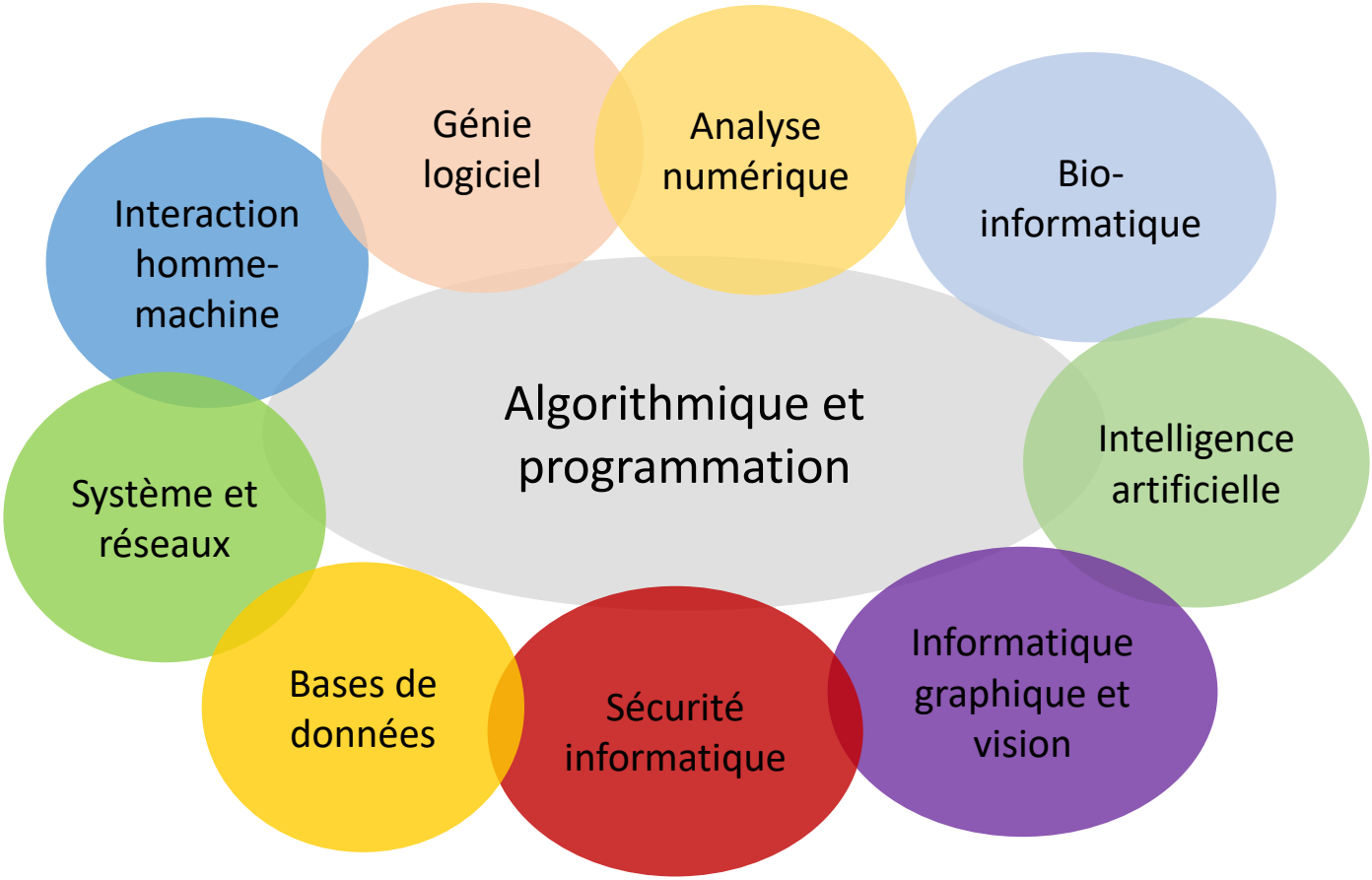
LIFAPSD – Algorithmique, Programmation et Structures de données

Nicolas Pronost

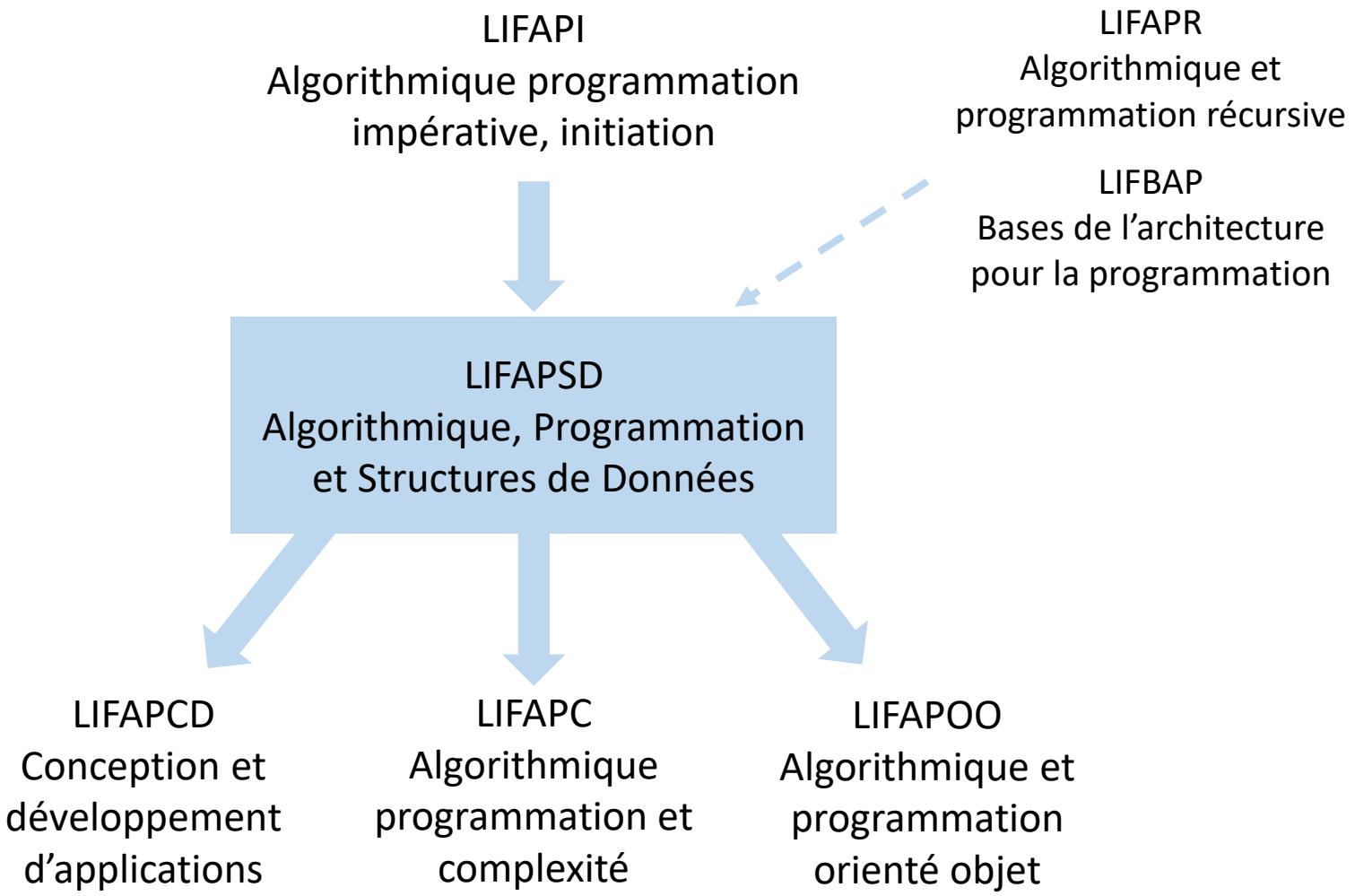
LIRIS

Présentation de l'UE

- L'algorithmique et la programmation sont au centre de toutes applications logicielles, tous domaines confondus



Place de l'UE dans le parcours LIFAP



Prérequis (surtout LIFAPI)

- Savoir écrire un algorithme simple en langage algorithmique
  - manipuler des variables de type booléen, entier, réel, caractère
  - manipuler des tableaux et des chaînes de caractères
  - connaître les structures de contrôle (tests, boucles, ...)
  - savoir découper un programme en fonctions et procédures
  - connaître les modes de passage des paramètres
  - être familier avec l'organisation de la mémoire
- Savoir implémenter tout ça en langage C/C++

Contenu de l'UE LIFAPSD

- Manipulation des principales structures de données utilisées en informatique
  - types primitifs: entier, réel, caractère, booléen, pointeur
  - types agrégés: tableau et structure
  - gestion des entrées-sorties: standard et fichier
  - structures dynamiques: tableau, liste, arbre, pile et file
- Les classes et objets
  - membre, spécificateur d'accès, surcharge d'opérateur
  - type de données abstrait et programmation modulaire
- Algorithmique
  - algorithmes de tri (sélection, insertion, fusion)
  - introduction à la notion de complexité algorithmique

Objectifs d'apprentissage

- A la fin de cette UE, vous serez capable
  - d'expliquer les forces et faiblesses des structures de données classiques
  - de construire une structure de donnée complexe à partir de structures de base
  - de prévoir l'évolution des données manipulées par un algorithme
  - de mesurer l'efficacité d'un algorithme simple
  - de concevoir un algorithme s'appuyant sur une structure de données appropriée
  - de mettre tout ça en œuvre en C++

Modalités d'évaluation

2 CC de TD (individuel, en salle de TD, env. 15 min)	20 % (10+10)
2 CC de TP (individuel, en salle de TP, env. 30 min)	40 % (20+20)
ECA (session 1, début janvier) (individuel, anonyme, en amphi, 1h30)	40 %
Rattrapage de l'ECA (session 2, fin juin) (individuel, anonyme, en amphi, 1h30)	40 %

Le rattrapage de session 2 ne remplace que la note de l'ECA de janvier, pas la note globale de l'UE, et le remplace dans tous les cas (même si note obtenue inférieure). Les autres notes (CC de TD et TP notés) sont conservées.

Une absence justifiée à un CC entraine sa neutralisation et injustifiée un zéro au CC. Une absence (justifiée ou injustifiée) à l'ECA entraine un zéro à l'épreuve.

Informations pratiques

- Site de l'UE

`http://licence-info.univ-lyon1.fr/LIFAPSD`

  - Informations générales sur l'UE
  - Transparents des CM
  - Ressources pour certains TP
    - Vous pouvez venir en TP avec vos ordinateurs portables
  - Diapositives des cours et polycopié TD/TP
    - Notes de cours dès le début de semestre
    - Pas besoin d'imprimer le polycopié vous-même, il vous sera distribué au premier TD
  - Corrigés des exercices de TD (au fur et à mesure)
  - Corrigés des exercices de TP (au fur et à mesure)
  - Annales d'examens (énoncés et corrigés)
  - Littérature (liens)

Chapitre 1
Introduction et rappels

Qu'est ce qu'un programme?

- Déclarations et définitions de données
  - Structurées en « structures de données »
  - Décrivent les données à manipuler

```
tab : tableau [1...15] de réels
i : entier
moy : réel
```

```
double tab [15];
int i;
float moy;
```

- Instructions
  - Structurées par l'algorithme
  - Décrivent les actions à effectuer sur les données

```
moy ← 0.0
Pour i allant de 1 à 15 par pas de 1 faire
   moy ← moy + tab[i]
Fin pour
moy ← moy / 15.0
```

```
moy = 0.0;
for (i=0;i<15;i++) {
   moy = moy + tab[i];
}
moy = moy / 15.0;
```

Différentes structures de données

Niveau langage (évolué)	Table Arbre Graphe Fichier Liste chaînée Pile File Tableau dynamique Tableau statique Structure/Classe
	Booléen Caractère Entier Réel Pointeur
Niveau machine	Mot Byte (octet) Bit

Relation entre données et instructions

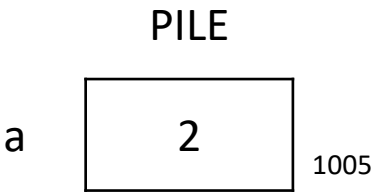
- Il faut utiliser une structuration des données qui rend l'algorithme (séquence d'instructions) le plus simple et le plus efficace possible

- Exemples avec des éléments de jeux
  - E/S jeu du pendu => caractères
  - Une grille (dame, échec, reversi) => tableau statique 2D
  - Le joueur => structure/classe
  - Les éléments à afficher à l'écran => tableau de taille dynamique
  - Les sauvegardes => fichier
  - Le déplacement des ennemis => graphe

Notion de variable

- Chaque variable possède
  - un nom
  - un type, et donc une taille en octets
  - une adresse en mémoire, qui ne change pas au cours de l'exécution
  - une valeur, qui elle peut changer au cours de l'exécution du programme
- Quand une nouvelle valeur est placée dans une variable (par affectation ou **cin**), elle remplace et détruit la valeur précédente

```
int a = 2;
```



Taille des types primitifs

Type	Taille supposée pour le modèle théorique utilisé en LIFAPSD
char, bool	1 octet
int, float	4 octets
double, @	8 octets

Taille des types primitifs

- Si vous êtes curieux pour votre machine

```
#include <iostream>

int main () {
   cout << "Taille d'un char:   " << sizeof(char) << endl;
   cout << "Taille d'un int:    " << sizeof(int) << endl;
   cout << "Taille d'un float:  " << sizeof(float) << endl;
   cout << "Taille d'un double: " << sizeof(double) << endl;
   cout << "Taille d'une @:     " << sizeof(void*) << endl;
   // etc.
   return 0;
}
```

Les opérateurs

- Affectation pour donner une valeur à une variable
  - =
  - ⚠ Ne pas confondre avec le signe égal mathématique. Cette confusion crée rarement une erreur de compilation
- Opérations arithmétiques classiques
  - + , - , / , * , % ...
- Affectation composée
  - += , -= ...
  - variable += valeur est équivalent à variable = variable + valeur
- Incrémentation et décrémentation
  - ++ et --
  - a++ ⟺ a += 1 ⟺ a = a+1
    - a++ retourne la valeur avant incrément
    - ++a retourne la valeur après incrément
- Autres opérateurs
  - [ ] pour effectuer un décalage dans l'accès mémoire
  - >> et << pour lire et écrire (clavier, écran)

Les opérateurs relationnels et logiques

Opération	Description
`!x`	Retourne faux si x est vrai et vice-versa
`x < y`	Retourne vrai si x est strictement plus petit que y
`x > y`	Retourne vrai si x est strictement plus grand que y
`x <= y`	Retourne vrai si x est plus petit ou égal à y
`x >= y`	Retourne vrai si x est plus grand ou égal à y
`x == y`	Retourne vrai si x est égal à y
`x != y`	Retourne vrai si x est différent de y
`x && y`	Retourne vrai si x et y sont tous les deux vrai
`x || y`	Retourne vrai si au moins l'un des deux est vrai
`x ^ y`	Retourne vrai si un et un seul des deux est vrai

Structures de contrôle

- Structure conditionnelle
  - l'instruction si : **if-else**
- Structure itérative
  - la boucle tant-que : **while**
  - la boucle faire tant-que : **do-while**
  - la boucle pour : **for**
- Structure de saut
  - l'instruction **break**
  - l'instruction **continue**
  - l'instruction **goto**
- Structure sélective
  - l'instruction cas : **switch**

La visibilité (portée) des variables

- Les variables sont accessibles dans le bloc d'instructions dans lequel elles sont définies
- Variable globale
  - déclarée en dehors de tout bloc d'instructions
  - utilisable dans n'importe quel bloc d'instruction du programme
    - y compris dans un autre fichier source (si déclarée **extern**)
  - stockée dans le segment « variables globales » de l'espace d'adressage
  - à éviter: risque de modification non désirée ou non prévue
- Variable locale
  - déclarée à l'intérieur d'un bloc d'instructions (ex. dans une fonction ou une boucle)
  - inutilisable dans un autre bloc
  - masque, le cas échéant, une variable globale de même nom
  - stockée dans le segment « pile » de l'espace d'adressage

Les entrées/sorties standards

- En utilisant la bibliothèque C++ iostream

```
#include <iostream>
using namespace std;
```

- Afficher sur la sortie standard (l'écran)

```
cout << "Bienvenue " << nomDuJoueur << endl;
```

- Lire depuis l'entrée standard (le clavier)

```
int AgeDuJoueur;
cout << "Veuillez entrer votre age: ";
cin >> AgeDuJoueur;
```

Les fonctions

- Une fonction est un groupe d'instructions qui est exécutée quand elle est appelée depuis une autre instruction du programme

```
type nom_fonction ([parametre1, parametre2, ...]) {
  // bloc d'instructions
}
```

  - le type indique le type de la valeur de retour de la fonction
  - le nom sert à identifier la fonction
  - les paramètres formels (type suivi d'un identificateur) servent de variables locales à l'intérieur de la fonction

Fonctions vides et procédures

- Des fonctions sans paramètres et/ou sans type de retour

```
int UneFonctionSansParam (void) {
  int b = 1;
  return b;
}
```

```
void UneProcedureAvecParam (int a) {
  int b = a + 1;
}
```

```
void UneProcedureSansParam () {
  int b = 1;
}
```

Fonction modificatrice

- Les paramètres formels sont des COPIES des paramètres effectifs
  - Les modifications des paramètres formels dans la fonction n'ont pas d'effet sur les paramètres effectifs
  - Si une modification est souhaitée, il faut utiliser une référence au paramètre effectif

```
void PrecEtSuiv (int x, int & prec, int & suiv) {
   prec = x-1;
   suiv = x+1;
}

int main () {
   int x = 100; int y = 15; int z = 8;
   PrecEtSuiv(x,y,z);
   cout << "Précédent=" << y << ", Suivant= " << z;
   return 0;
}
```

Passage de paramètres : rappel

Norme algorithmique	Mise en œuvre en C++	Utilisation
Mode donnée	*type* x	Petits objets
	const *type* & x	Gros objets
Mode donnée-résultat ou résultat	*type* & x	Tous objets

- `type x` recopie le paramètre effectif
  - uniquement pour les objets prenant peu de mémoire (types primitifs)
- **`const`** `type & x` recopie l'adresse du paramètre effectif (8 octets quelque soit le paramètre)
  - interdit la modification du paramètre effectif
- `type & x` recopie l'adresse du paramètre effectif (8 octets quelque soit le paramètre)
  - autorise la modification du paramètre effectif

Notion de tableau

- Utilité: éviter d'avoir à définir 10 variables différentes pour stocker 10 notes, par exemple

```
n1, n2, n3, n4, n5, n6, n7, n8, n9, n10 : réels
notes : tableau [1...10] de réels
```

- Tableau = ensemble ordonné d'éléments contigus en mémoire et de même type
- Chaque élément est repéré par son indice = sa position dans le tableau

1	2	3	4	5	6	7	8	9	10
15.5	12.5	10.0	8.5	17.0	14.5	16.0	13.0	12.5	10.5

- Accès à la valeur d'un élément par l'opérateur [ ]

```
notes[1] ← 16.0
afficher(notes[3])
```

Tableau : mise en œuvre en C++

- Déclaration d'un tableau statique

```
type nomTableau [taille];
```

 Les indices vont de 0 à taille-1

 taille est un entier positif

indice	0	1	2	3	4	5	6	7	8	9
position	1	2	3	4	5	6	7	8	9	10
valeur	15.5	12.5	10.0	8.5	17.0	14.5	16.0	13.0	12.5	10.5

- Aucune opération globale possible sur les tableaux
  - tab1 = tab2 ne fonctionne pas! (recopie l'adresse du tableau)
  - il faut faire la copie élément par élément, avec une boucle

Tableau : mise en œuvre en C++

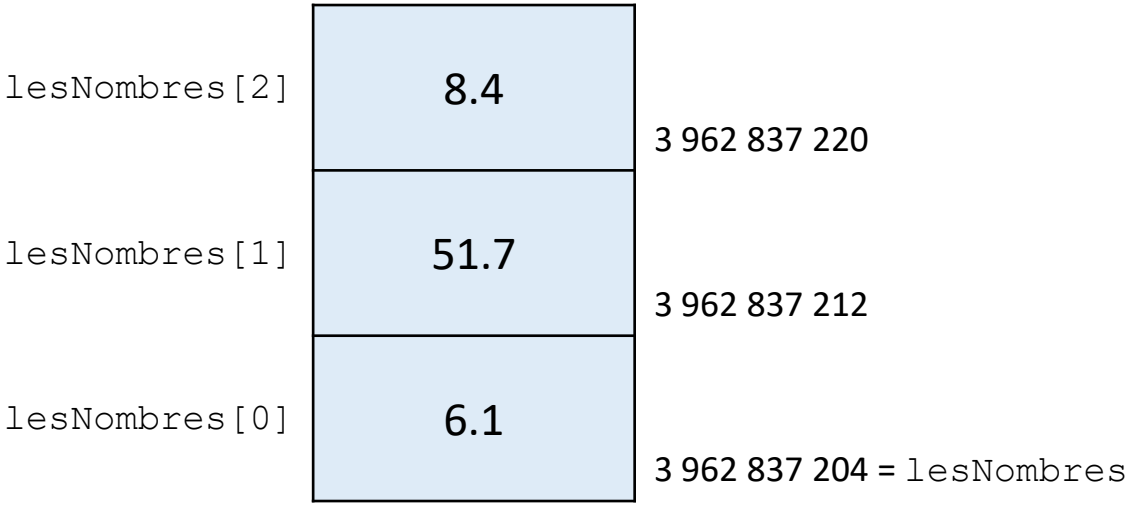
- Dans l'expression constante de la taille du tableau, tous les opérateurs arithmétiques sont autorisés, ainsi que l'opérateur **sizeof**
- La taille peut être omise lorsque le compilateur peut déduire sa valeur

`int tab[4];`	OK
`int tab[3*8+2];`	OK
`#define TAILLE 10` `int tab[TAILLE + 5];`	OK
`int tab[8*sizeof(float)];`	OK
`int tab [] = {2,3,5,7,11};`	OK
`int n = tailleTab();` `int tab[n];`	OK en C++98
`int tab[3.14159];`	PAS ENTIER
`int tab[-2];`	PAS POSITIF

Interprétation d'un tableau

- Une variable tableau ne désigne pas le tableau en entier, mais l'adresse mémoire du premier élément du tableau

```
double lesNombres [] = {6.1, 51.7, 8.4};
```



- On accède aux autres éléments en ajoutant la bonne quantité à l'adresse de base (c.-à-d. à `lesNombres`)

Arithmétique des pointeurs

- Quelle est cette « bonne quantité » ?
  - Si p est une variable contenant l'adresse 3 925 632 140, que vaut p + 1 ?
- Cela dépend du type des éléments stockés à partir de p!
  - **int** p [] => p + 1 == 3 925 632 144 (p + 1 int)
  - **double** p [] => p + 1 == 3 925 632 148 (p + 1 double)
  - **char** p [] => p + 1 == 3 925 632 141 (p + 1 char)
- p + i désigne l'adresse située à i***sizeof**(type) octets plus loin
- Conséquence
  - tab + i = adresse de tab[i]
  - tab[i] = valeur dans la case i

En C++, la mise en œuvre de l'opérateur [ ] repose sur l'arithmétique des pointeurs

Tableau en paramètre

```
void procedure (int tab[10]);
void procedure (int tab[]);
void procedure (int * tab);
```

- Les 3 écritures sont strictement équivalentes
- Elles reviennent à passer à `procedure` l'adresse du 1er élément du tableau
- La taille du tableau n'est pas transmise (même dans la première écriture)
- Le contenu du tableau n'est pas copié, seule son adresse l'est, donc manipuler tab manipule le tableau original (passage en mode résultat ou donnée-résultat)
- Pour un passage en mode donnée: `void procedure (const int * tab);`
  - Toujours pas de copie locale, mais on s'engage à ne pas modifier le contenu du tableau original

Tableau en retour de fonction

- On ne peut pas stocker le tableau dans la pile si on veut le renvoyer (uniquement local à la fonction, sera détruit en sortant)
- Il faut le stocker soi-même sur le tas: allocation dynamique de mémoire
- Attention au risque de « fuite mémoire » si l'on oublie de libérer le tableau

Voir cours suivant sur l'allocation dynamique de mémoire

Tableau à 2 dimensions - algorithmique

	1	2	3	4	5	6	7	8	9	10
1	15.5	12.5	10.0	8.5	17.0	14.5	16.0	13.0	12.5	10.5
2	6.0	18.5	19.0	7.5	13.5	13.0	11.0	13.0	15.5	9.5
3	10.5	10.0	12.0	15.5	11.5	9.0	8.5	14.0	18.5	17.0

- Déclaration `nom : tableau [1...L][1...C] de type`
  - ex. `notes : tableau [1...3][1...10] de réels`
- Lignes numérotées de 1 à L et colonnes de 1 à C
- Accès à un élément

```
notes[2][9] ← 16.0
```

- Parcours de toute la matrice à l'aide de 2 boucles imbriquées

Tableau à 2 dimensions – C++

- Déclaration d'un tableau 2D statique

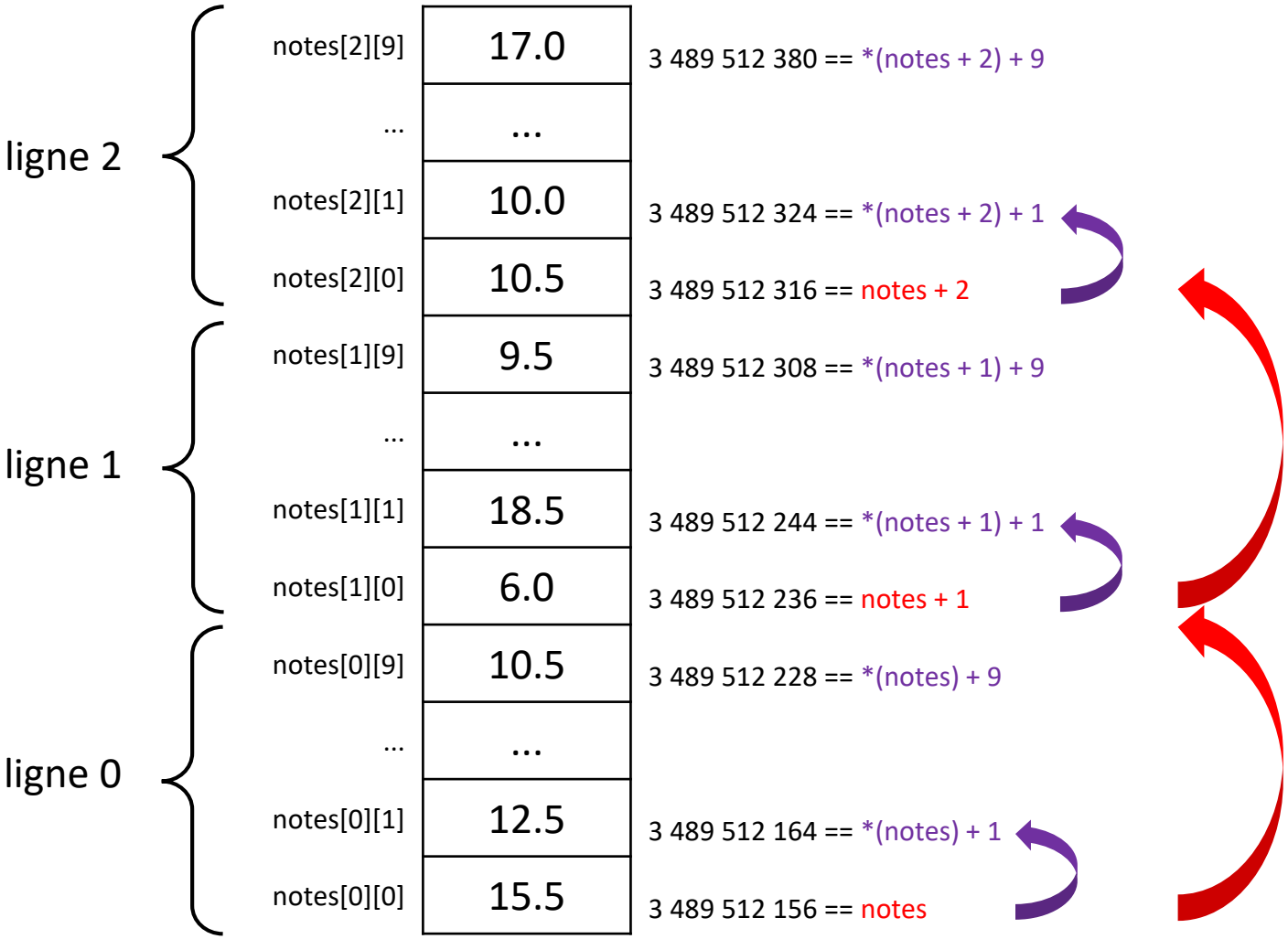
```
type nom [L][C];
```

  - L et C doivent être des entiers positifs
  - ex. `double notes [3][10];`
- Lignes indicées de 0 à L-1 et colonnes de 0 à C-1
- Accès à un élément

```
notes [1][8] = 16.0;
```

- Parcours de toute la matrice à l'aide de 2 boucles imbriquées

Tableau à 2 dimensions - mémoire



Chaîne de caractères

- Il y a deux représentations possibles
  - A la C, avec un tableau 1D d'éléments de type caractère

```
char question [] = "Quel est votre nom?";
```

  - A la C++, avec la classe string (tableau dynamique)

```
#include <string>
string question ("Quel est votre nom?");
```

- Par exemple, le tableau `char texte [22];` est un tableau qui peut stocker jusqu'à 22 caractères
  - Le tableau n'a pas besoin d'être entièrement remplis
  - La séquence de caractères se termine par le caractère nul '\0' (backslash zéro)

L	I	F	A	P	S	D	\0												
L	I	F	A	P	S	D		C	'	E	S	T		S	U	P	E	R	!	\0	

Chaîne de caractères

- La représentation en tableau utilise le même formalisme que les autres tableaux
  - Initialisation avec une séquence définie de caractères

```
char invite [] = {'B', 'o', 'n', 'j', 'o', 'u', 'r', '\0'};
```

  - Le caractère de terminaison est ajouté automatiquement dans une constante littérale (texte entre guillemets)

```
char invite [] = "Bonjour";
```

  - Impossible de déclarer le tableau puis affecter la chaîne de caractères (copie des adresses des tableaux)
  - Possibilité de changer chaque élément individuellement

```
invite[0] = 'H';
invite[1] = 'e';
invite[2] = 'l';
invite[3] = 'l';
invite[4] = 'o';
// invite vaut maintenant
```

H	e	l	l	o	u	r	\0

Chaîne de caractères

- Les deux représentations coexistent et sont convertibles

```
char question1 [] = "Quel est votre nom?";
string question2 ("Où habitez-vous?");
char reponse1 [80];        // tableau statique (taille connue)
string reponse2;           // tableau dynamique (taille inconnue)
cout << question1;
cin >> reponse1;
cout << question2;
cin >> reponse2;
cout << "Bonjour " << reponse1;
cout << "de " << reponse2;
```

```
char texteEnC [] = "LIFAPSD c'est super!";
string texteEnCPP (texteEnC);       // conversion tableau en string
const char * ptrTexteEnC = texteEnCPP.c_str(); // string en tableau
```

-  Attention à ne pas confondre a, 'a' et "a"

Chaîne de caractères

- Quelques fonctionnalités de la classe string
  - Affectation entre chaînes (copie du contenu de la chaîne)

```
string str1 = "Texte";
string str2 = str1;
```

  - Taille de la chaîne en octets ( peut être différente du nombre de caractères, suivant l'encodage)

```
string str = "Texte";
int taille = (int) str.size();
```

  - Opérateur [] pour accéder aux caractères individuellement
  - Opérateur + pour ajouter en fin de chaîne

```
string chemin = "~/LIFAPSD";
string repTP, fichier;
cout << "Entrez le repertoire: "; cin >> repTP;
cout << "Entrez le fichier: "; cin >> fichier;
chemin = chemin + "/" + repTP + "/" + fichier + ".cpp";
```

Chaîne de caractères

- Quelques fonctionnalités de la classe string
  - Rechercher (sous chaîne, caractère, depuis début, depuis fin...)

```
string path = "~/LIFAPSD/TP2/main.cpp";
int posPoint = (int) path.find('.');
```

  - Comparaison de chaînes

```
string str1 ("LIFAPSD c'est super!");
string str2 ("LIFAPSD c'est genial!");
int comp = str1.compare(str2);
```

    - 0 si égal, négatif ou positif suivant l'ordre alphabétique

  - Beaucoup d'autres pour itérer, vider, tester si vide, ajouter, supprimer, remplacer, copier, convertir en valeur numérique, etc.

Notion de structure

- Nouveau type de donnée pour ajouter aux types primitifs
- Agrégat d'information de types éventuellement différents (donc différent d'un tableau)
- Les différentes informations s'appellent les champs
- Chaque champ est de type primitif, un tableau ou un autre type défini par l'utilisateur (ex. une autre structure)
- Manipulation des champs avec l'opérateur . (point)

```
Structure Date
   jour : entier
   mois : entier
   annee : entier
Fin structure

Structure Etudiant
   numero : entier
   nom : chaine de caractères
   prenom : chaine de caractères
   datenaiss : Date
   notes : tableau de 10 reels
Fin structure

Variables
   unEtudiant : Etudiant
Debut
   unEtudiant.numero ← 164987
   saisir(unEtudiant.nom)
   saisir(unEtudiant.prenom)
   saisir(unEtudiant.datenaiss.jour)
   ...
Fin
```

Structure et sous-programme

- Un sous-programme peut renvoyer une structure

```
Fonction saisirEtudiant (num : entier) : Etudiant
Parametre en mode donnée: num
Variables locales: etu : Etudiant
Debut
   etu.numero ← num
   afficher("Saisir le nom:")
   saisir(etu.nom)
   afficher("Saisir le prenom:")
   saisir(etu.prenom)
   afficher("Saisir le jour de naissance:")
   saisir(etu.datenaiss.jour)
   ...
   retourner etu
Fin saisirEtudiant
```

Structure et sous-programme

- Un sous-programme peut prendre une structure en paramètre

```
Fonction saisirEtudiant (num : entier) : Etudiant
Parametre en mode donnée: num
Variables locales: etu : Etudiant
Debut
   etu.numero ← num
   afficher("Saisir le nom:")
   saisir(etu.nom)
   afficher("Saisir le prenom:")
   saisir(etu.prenom)
   afficher("Saisir la date de naissance:")
   saisirDate(etu.datenaiss)
   retourner etu
Fin saisirEtudiant
```

```
Procédure saisirDate (d : Date)
Parametre en mode donnée-résultat: d
Debut
   afficher("Saisir le jour:")
   saisir(d.jour)
   afficher("Saisir le mois:")
   saisir(d.mois)
   afficher("Saisir l'année:")
   saisir(d.annee)
Fin saisirDate
```

Structure et tableau

- Une structure peut contenir des champs tableaux
- On peut aussi faire des tableaux de structures

```
Variables
   lesEtudiants : tableau [1..1000] de Etudiant
   i : entier
Debut
   Pour i allant de 1 à 1000 par pas de 1 faire
      lesEtudiants[i] ← saisirEtudiant(10264+i)
   Fin pour
   ...
Fin
```

10265	10266	10267	10268	…
"Dupont"	"Dupont"	"Durand"	"Dupuis"	…
"Ernest"	"Arnold"	"Aurelie"	"Coralie"	…
5	28	14	21	…
12	10	1	9	…
1990	1990	1991	1989	…

Comparaison de structures

- Tout comme la lecture et l'écriture ne peuvent pas se faire automatiquement (format inconnu), la comparaison entre structures doit être spécifiée
- En donnant une fonction de comparaison (on verra plus tard un autre moyen plus standard)

```
Fonction estInférieur(etu1 : Etudiant, etu2: Etudiant) : booléen
```

  - De même on peut avoir besoin de: estEgal, estSuperieur, estSuperieurOuEgal, estInferieurOuEgal, estDifferent, etc.
  - Pour notre exemple, on peut décider qu'un étudiant est « plus petit » qu'un autre quand son numéro est plus petit, mais aussi on pourrait dire lorsque son nom vient avant dans l'ordre alphabétique ou lorsqu'il est plus jeune,...
  - Le programme ne peut pas décider pour vous!

Structure en C++

- Déclaration et définition avec le mot clé **struct**
  - Ne pas oublier le point-virgule à la fin (instruction)

```
struct nom_structure {
   type_champ1 champ1;
   type_champ2 champ2; ...
} [nom_variables];
```

- Accès aux champs avec l'opérateur . (point)

```
struct Date {
   int jour, mois, annee;
};

struct Etudiant {
   int numero;
   string nom, prenom;
   Date datenaiss;
   double notes [10];
};
```

```
Etudiant saisirEtudiant (int num) {
   Etudiant etu;
   etu.numero = num;
   cout << "Saisir le nom:";
   cin >> etu.nom;
   cout << "Saisir le prenom:";
   cin >> etu.prenom;
   cout << "Saisir la date de naissance:";
   saisirDate(etu.datenaiss);
   return etu;
}
```