PROGRAMMATION ORIENTÉE OBJETS
C++

UFR-SET/LMI2

Dr. Pape Abdoulaye BARRO

Présentation et objectif du cours

- **Organisation du travail (48h)**
  - Cours magistral (24h)
  - TD/TP (24h)
- **Evaluation**
  - Contrôles
  - Examen
- **Outils de travail**
  - Visual C++
  - Dev-C ++
- **Prérequis**
  - Quelques notions en C

Généralités

- le C++ est un langage à la fois procédural et orienté objet contrairement au C qui est seulement procédural.
  - Un langage est dit procédural s'il permet seulement la définition des données grâce à des variables, et des traitements grâce aux fonctions. il sépare données et traitements sur ces données.
  - Un langage est dit orienté objet lorsqu'il offre un mécanisme de classe rassemblant données et traitements.
- Le paradigme de programmation orientée objet implique une méthode différente pour concevoir et développer des applications.

Généralités

Historique

- Le C++ était initialement nommé C with Classes (C avec classes) par son développeur ***Bjarne Stroustrup***, *qui a eu l'idée,* **en 1979**, *d'améliorer le langage C pour sa thèse de doctorat*.
- **En 1984** le nom du langage passa de C with classes à celui de « **C++** ».
- Dès le départ, le langage ajoutait à C la notion de classe (avec l'encapsulation des données), de classe dérivée, de vérification des types renforcés (typage fort), d'argument par défaut, ...
- **En 1989**, c'est la sortie de la **version 2.0** de C++ avec l'ajout des nouvelles fonctionnalités, telle que l'héritage multiple, les classes abstraites, les fonctions membres statiques, les fonctions membres constantes, et les membres protégés.
- **Au cours de son évolution**, des fonctionnalités nécessaires ont été ajoutées:
  - les fonctions C traditionnelles telles que printf et scanf et autres, sont remplacées;
  - parmi les ajouts les plus importants, il y avait la Standard Template Library (*stl*).
- **En 2003**, une version de C++ corrigée est publiée, prenant en compte les erreurs remontés par les utilisateurs.

Généralités

Fonctionnalités et inclusion de fichier d'en-tête

- **On peut considérer que C++ « est du C » avec un ajout de fonctionnalités.**
  - Mais, Il serait utile d'avoir à l'esprit qu'un programmes syntaxiquement corrects en C peut ne pas l'être en C++.
- L'utilisation d'une bibliothèque peut se faire par l'intermédiaire de la directive ***#include*** (suivie du nom du fichier d'en-tête) comme en C.
  - Aussi, depuis le ***C++20***, le mot clé ***import*** peut servir à des fins similaires.
- **Dans ce cours, nous parlerons de Classe, d'Objet, de Méthode, d'Encapsulation, de Constructeur/Destructeur, de Surcharge, d'Héritage et de Polymorphisme.**

Rappel sur les bases (Parte I)

- **Les variables, les opérateurs et les opérations**
- Structures de contrôles
- Tableaux et pointeurs
- Fonctions et récursivité
- Quelques algorithmes de tri et de recherche
- Fichiers
- Structures
- Listes chainées, Piles, Files

Rappel

Les variables, les opérateurs et les opérations

- Une variable est un espace mémoire nommé, de taille fixée, prenant au cours du déroulement d'un programme, un nombre indéfini de valeurs différentes.

  - Un programme tourne généralement autours des variables;
  - Le changement de valeur se fait par l'opération d'affectation.
  - La variable diffère de la notion de constante qui, comme son nom l'indique, ne prend qu'une unique valeur au cours de l'exécution du programme.

Rappel

Les variables, les opérateurs et les opérations

- Dans la plupart des langages de programmation, avant de manipuler une variable, il faut préalablement déclarer son type. C'est à dire que la variable en question ne pourra changer de valeur que dans l'intervalle défini par le type qui lui est assigné.
- Le tableau ci-dessous, donne les noms des types et leur plage:

type	Min	Max
Signed char	-127	127
int	-32 767	32 767
long	-2 147 483 647	2 147 483 647
float	$-1x10^{37}$	$1x10^{37}$
double	$-1x10^{37}$	$1x10^{37}$

Rappel

Les variables, les opérateurs et les opérations

- type <*nom_variable*> permet de déclarer une variable. Mais le nommage des variables est régit par les règles suivantes :
  - elle commence par une lettre ;
  - les espaces sont interdits. On peut utiliser 'underscore' pour cela ;
  - on ne peut pas utiliser des accents ;
  - on peut utiliser des minuscules, des majuscules et des chiffres.
- Pour déclarer une constante, il faut utiliser le mot const devant le type et il est obligatoire de lui donner une valeur au moment de sa déclaration.
- On afficher le contenu d'une variable avec '***cout***' puis chevrons ouvrant '<<' (ex: cout<<variable;).
- Il est possible d'afficher la valeur de plusieurs variables dans un seul cout. Il vous suffit pour cela de les séparer par des chevrons ouvrants (ex: cout<<variable 1<<variable 2 <<…<<variable n;).
- On récupère les caractères saisis du tampon claver avec cin au moyen d'opérateur d'entée (chevrons fermants) '>>' (ex:cin>>variable;).

Rappel

Les variables, **les opérateurs** et les opérations

Un **opérateur** est un outil qui permet d'agir sur une variable ou d'effectuer des calculs. Il existe plusieurs types d'opérateurs:

- L'affectation qui confère une valeur à une variable ou à une constante, est représenté par le symbole '=' ;
- Les opérateurs arithmétiques qui permettent d'effectuer des opérations arithmétiques entre opérandes numériques :

+	addition
-	soustraction
*	multiplication
/	division
%	modulo

Rappel

Les variables, **les opérateurs** et **les opérations**

- Les opérateurs relationnels :

>	supérieur
<	inférieur
>=	Supérieur ou égal
=<	Inférieur ou égal
==	égal
!=	différent

- Les opérateurs logiques :
  - Opérateur unaire : « ! » (négation) ;
  - Opérateurs binaires : «&&» (conjonction), «| |» (disjonction).
- La concaténation : qui permet de créer une chaine de caractères à partir de deux chaînes de caractère en les mettant bout à bout. Il est représenté par le symbole « + ».
- Opérateur ternaire: Il permet l'affectations du type.
  - Syntaxe: Si condition est vraie alors variable vaut valeur, sinon variable vaut autre valeur.
  - Exemple: int a = (b > 0) ? 10 :20;

Rappel

Les variables, **les opérateurs** et **les opérations**

- Les opérateurs de manipulation de bit :
  - & : ET bit à bit
  - | : OU bit à bit
  - ^ : OU Exclusif bit à bit
  - << : Décalage à gauche
  - >> : Décalage à droite
  - ~ : Complément à un (bit à bit)
- La fonction sizeof est utilisée pour connaitre la taille en mémoire d'une variable passé en paramètre.
  - int a = 1; sizeof(a) donne 4; double a = 3,14; sizeof(a) donne 8.
- sur les entiers et les réels : addition, soustraction, multiplication, division, division entière, puissance, comparaisons, modulo ;
- sur les booléens : comparaisons, négation, conjonction, disjonction ;
- sur les caractères : comparaisons ;
- sur les chaînes de caractères : comparaisons, concaténation

Rappel

Les variables, **les opérateurs** et **les opérations**

Lors de l'évaluation d'une expression, la priorité de chaque opérateur permet de définir l'ordre d'exécution des différentes opérations. Pour changer la priorité d'exécution, on utilise les parenthèses.

- Ordre de priorité décroissante des opérateurs arithmétiques et de concaténation :
  - «*», «/» ;
  - « % »
  - «+» et «-» ;
  - «+» (concaténation).

- Ordre de priorité décroissante des opérateurs logiques :
  - « ! »
  - « && »
  - « || »

Rappel

Les variables, les opérateurs et les opérations

EXERCICES D'APPLICATIONS

- Application 1 :

Ecrire un algorithme/programme permettant de déclarer deux variables de type réel, de saisir les valeurs, de calculer et d'afficher leur somme, produit et moyenne.

- Application 2 :

Ecrire un algorithme/programme qui permet de permuter les valeurs de A et B sans utiliser de variable auxiliaire.

- Application 3 :

Ecrire un algorithme/programme permettant de déclarer trois variables A, B, C de type réel, d'initialiser leurs valeurs et ensuite d'effectuer la permutation circulaire des trois variables.

- Application 4 :

Ecrire un algorithme/programme qui permet de saisir les paramètres d'une équation du second degré et de calculer son discriminant delta.

- Application 5 :

Ecrire un algorithme/programme qui à partir de la valeur saisie du côté d'un carré donné, permet de calculer son périmètre et sa surface et affiche les résultats à l'écran.

Rappel sur les bases

- Les variables, les opérateurs et les opérations
- **Structures de contrôles**
- Tableaux et pointeurs
- Fonctions et récursivité
- Quelques algorithmes de tri et de recherche
- Fichiers
- Structures
- Listes chainées, Piles, Files

Rappel

Structures conditionnelles et les structures itératives

Un ordinateur exécute un programme de manière séquentielle. Pour lui doter de l'intelligence relative afin d'être capable d'effectuer des choix ou des boucles sur une bloc d'instructions et de casser cette linéarité, il va falloir utiliser les structures de contrôle.

- Parmi les structures de contrôle nous avons :
  - LES STRUCTURES CONDITIONNELLES
  - LES STRUCTURES ITERATIVES

Rappel

Structures conditionnelles et les structures itératives

- if, syntaxes:

```
if(condition) {
        /*instructions*/
}
```

  - Condition étant une expression booléenne
  - Exemple:
    - …..
    - if(jour != 7) {
    - cout <<''Je vais à l'école''<<endl;
    - }
    - ….

Rappel

Structures conditionnelles et les structures itératives

- if…else, syntaxes:

```
if(condition) {
        /*instructions*/
} else {
        /*instructions*/
}
```

  - Exemple:

```
…..
if(jour != 7 && greve==0) {
   cout <<''Je vais à l'école''<<endl;
} else {
  cout <<''Il n'y a pas école''<<endl;
}
….
```

Rappel

Structures conditionnelles et les structures itératives

- if imbriquées, syntaxes:

```
if(condition1) {
        /*instructions*/
} else if(condition 2){
        /*instructions*/
} else if(condition 3){
        /*instructions*/
}
…
else if(condition n){
        /*instructions*/
} else {
        /*instructions*/
}
```

- Exemple:

```
…
if(jour==1) {
   cout <<''Lundi''<<endl;
} else if(jour==2){
   cout <<''Mardi''<<endl;
} else if(jour==3){
   cout <<''Mercredi''<<endl;
}
…
else if(jour==6){
   cout <<''Samedi''<<endl;
} else {
   cout <<''Dimanche''<<endl;
}
…
```

Rappel

Structures conditionnelles et les structures itératives

- Structure à choix multiple, syntaxes:

```
switch (expression)
{
        case valeur1:
          {instruction 1};
        break;
        case valeur2:
          {instruction 2};
        break;
                  …
        default:
          {suite_instruction} ;
}
```

- Exemple:

```
…
switch(jour)
    {
   case 1:
     cout <<''Lundi''<<endl;
    break;
   case 2:
     cout <<''Mardi''<<endl;
    break;
   case 3:
     cout <<''Mercredi''<<endl;
    break;
   …
   case 6:
     cout <<''Samedi''<<endl;
    break;
    default:
     cout <<''Dimanche''<<endl;
   }
…
```

Rappel

Structures conditionnelles et les structures itératives

EXERCICES D'APPLICATIONS

- Application 6 :

Écrivez un programme qui calcule les solutions réelles d'une équation du second degré $ax^2+bx+c = 0$ en discutant la formule:

- Utilisez une variable d'aide d pour la valeur du discriminant b^2 - 4*a*c et décidez à l'aide de d, si l'équation a une, deux ou aucune solution réelle. Utilisez des variables du type entier pour a, b et c. Affichez les résultats et les messages nécessaires sur l'écran.

- Application 7 :

Écrivez un programme qui permet de calculer la superficie d'un cercle, d'un rectangle ou d'un triangle. L'utilisateur saisira "C", "R" ou "T" selon la superficie de la figure qu'il souhaite calculer, ensuite il saisira les dimensions.

Selon le choix de l'utilisateur, le programme doit pouvoir lui demander de saisir les dimensions appropriées.

Afficher ensuite à l'écran selon son choix la superficie demandée

Rappel

Structures conditionnelles et **les structures itératives**

- Supposons qu'on veut afficher tous les nombres entiers comprises entre 9 et 999. il va falloir faire:

```
cout<<''9'';
cout<<''10'';
 …
cout<<''999'';
```

Une tache répétitive fastidieuse. D'où la nécessité de trouver une solution alternative.

- Une itération consiste en la répétition d'un bloc d'instructions jusqu'à ce qu'une certaine condition soit vérifiée.

Il en existe 2 sortes:

- Le nombre d'itérations est connu d'avance
- Le nombre d'itération dépend du résultat précédemment obtenue.

Rappel

Structures conditionnelles et **les structures itératives**

- for, syntaxe:

```
for (initialisation ; condition ; incrémentation){
        /*instructions*/
 }
```

- Exemple:
  - …..
  - int compteur;
  - for (compteur = 9; compteur <1000 ; compteur++)
  - {
  - cout << compteur << endl;
  - }
  - ….

Rappel

Structures conditionnelles et **les structures itératives**

- while, syntaxe:

```
while(condition ){
        /*instructions*/
 }
```

- Exemple:

```
…..
int result(0), i(1), n;
cout << ''Entrez un entier naturel ?'' << endl;
cin >> n;
while(i<=n)
{
   result = result + i;
   i = i+1;
}
cout << ''Somme ='' << result << endl;
….
```

Rappel

Structures conditionnelles et **les structures itératives**

- do…while, syntaxe:

```
do {
        /*instructions*/
 } while(condition_de_reprise );
```

- le contenu de la boucle sera toujours lu au moins une fois.
- Exemple:

```
…..
int nombre(0);
do
{
    cout << ''veuillez entrer un entier ?'' << endl;
    cin >> nombre;
} while (nombre< 0);
….
```

Rappel

Structures conditionnelles et les structures itératives

EXERCICES D'APPLICATIONS

- Application 8:

Écrivez un programme qui calcule les solutions réelles d'une équation du second degré ax2+bx+c = 0 en discutant la formule:

- Utilisez une variable d'aide d pour la valeur du discriminant b2 - 4*a*c et décidez à l'aide de d, si l'équation a une, deux ou aucune solution réelle. Utilisez des variables du type entier pour a, b et c. **On suppose que les valeurs saisies sont non nulles**. Affichez les résultats et les messages nécessaires sur l'écran

- Application 9:

Ecrire un programme qui permet de faire les opérations suivantes :

- Ecrire un programme qui affiche la somme des n premiers entiers naturels. La valeur de n est saisie au clavier lors de l'exécution.
- Ecrire un programme qui affiche la somme des entiers compris entre les entiers d et f. Les valeurs de d et f sont saisies au clavier lors de l'exécution.
- Ecrire un programme qui affiche la somme des valeurs absolues des entiers compris entre les entiers relatifs d et f. Les valeurs de d et f sont saisies au clavier lors de l'exécution.

Rappel sur les bases

- Les variables, les opérateurs et les opérations
- Structures de contrôles
- **Tableaux et pointeurs**
- Fonctions et récursivité
- Quelques algorithmes de tri et de recherche
- Fichiers
- Structures
- Listes chainées, Piles, Files

Rappel

Tableaux, tableaux dynamiques (vector/string) et pointeurs

- Un tableau est une liste d'éléments ayant le même type et désignés sous le même nom et accessibles par indices.
  - Il est déclaré comme suit:
    - type nom [taille] ;
    - Exemple:
      - double notes[10] ;
    - En lecture, nous avons:
      - cin>> notes[i]; // i=0,1,2,…,9
    - En écriture, nous avons:
      - cout<<notes[i]<<endl; // i=0,1,2,…,9

Rappel

Tableaux, tableaux dynamiques (vector/string) et pointeurs

- On peut avoir un tableau bidirectionnel(ou multidimensionnel). Dans ce cas, il est déclaré comme suit:
  - type nom [ligne][colonne] ;
  - Exemple:
    - double matrice[2][3] ;
  - En lecture, nous avons:
    - cin>> matrice[1][2]; // i=0,1 et j=0,1,2
  - En écriture, nous avons:
    - cout<< matrice[i][j] <<endl; // i=0,1 et j=0,1,2

Rappel

Tableaux, tableaux dynamiques (vector/string) et pointeurs

- On peut attribuer un nom à un type, dans ce cas, on utilise la déclaration typedef:
- Sa forme générale est: **typedef** <déclaration>
  - C'est très pratique pour nommer certains types de tableaux.
  - Exemple:
    - **typedef** int Matrice[2][3]; // définit un type Matrice
  - On peut l'utiliser pour déclarer ensuite, son équivalent :
    - Matrice M;

Rappel

Tableaux, **tableaux dynamiques (vector/string)** et pointeurs

Un tableau dynamique est un tableau dont la taille peut varier.

- Avec vector, la syntaxe est la suivante: **vector**<TYPE> nom(TAILLE);
  // Il va falloir inclure la bibliothèque <vector>
- Quelques fonctions utiles:
  - push_back (): ajout à la fin du vecteur
  - pop_back (): retire de la fin du vecteur
  - size() : retourne le nombre d'éléments du vecteur
  - erase() : supprime un éléments ou un intervalle d'un vecteur et déplace les éléments suivants.

Rappel

Tableaux, **tableaux dynamiques (vector/string)** et pointeurs

- **Exemples**:
  - Créer un tableau vide: vector <double> tab;
  - Créer un tableau de 10 éléments: vector <int> tab(10);
  - Créer un tableau de 10 éléments initialisés à 0:
    - vector <int> tab(10, 0);
    - tab.push_back(3); // ajout du 11ème case au tableau de valeur 3;
    - tab.pop_back(); // suppression de la dernière case du tableau;
    - const int taille(tab.size()); // variable contenant la taille du tableau;
    - tab.erase(tab.begin()+5); // suppression du 6iéme élément;
    - tab.erase(tab.begin(), tab.begin()+5); // suppression des 5 premiers éléments;

Rappel

Tableaux, **tableaux dynamiques (vector/string)** et pointeurs

Il est également possible de créer des tableaux multidimensionnels de taille variable en utilisant les vector.

- Syntaxe pour 2D: **vector**<**vector**<TYPE>> nom; // nous avons plus tôt un tableau de ligne.

  Exemples:

  - vector<vector<int>> mat;
  - mat.push_back(vector(3)); //ajout d'une ligne de 3 cases;
  - mat[0].push_back(4); //ajout d'une case contenant 4 à la 1ière ligne du tableau;
  - mat[0][2] = 6; // change la valeur de la cellule (0, 2) du tableau;

Rappel

Tableaux, **tableaux dynamiques (vector/string)** et pointeurs

Une chaine de caractère est en réalité un tableau de caractères. Ce qui veut dire qu'il a beaucoup de point communs avec les vector.

- Pour pouvoir utiliser la classe standard, il faut rajouter la bibliothèque <string>;
  - Elle embarque toutes les opérations de base sur les chaînes:
    - **Déclaration**: string s1; string s2="Hello";
    - **Saisie et Affichage**: cin>>s1; cout<<s2;
    - **Concaténation**: string s3=s1+s2;
      - s3.size(); // pour connaitre le nombre de lettres;
      - s3.push_back("!"); // pour ajouter des lettres à la fin;
      - s3.at(i); // pour récupérer le i-ème caractère;
      - getline(cin, s4); // pour saisir une chaîne de caractères en utilisant le passage à la ligne comme séparateur (notre chaîne de caractères peut alors comporter des espaces);

Rappel

Tableaux, **tableaux dynamiques (vector/string)** et pointeurs

Exemple:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string prenom("Masamba");
    cout << "Je suis" << prenom << "et toi ?" <<endl;
    prenom[2] = 'd';
    prenom[3] = 'e';
    cout << "moi c'est" << prenom << "!" << endl;
    return 0;
}
```

Rappel

Tableaux, tableaux dynamiques (vector/string) et pointeurs

EXERCICES D'APPLICATIONS

- Application 10:

Ecrivez un programme qui lit au clavier une suite de nombres entiers positifs ou nuls et qui les affiche dans l'ordre inverse de leur lecture. La frappe d'un nombre négatif indique la fin de la série. Nous avons des raisons de penser qu'il n'y aura pas plus de 100 nombres.

- Application 11 :

On considère un tableau tab de N entiers. Ecrire un programme permettant:

a) de compter le nombre d'éléments nuls de tab
b) de chercher la position et la valeur du premier élément non nul de tab
c) de remplacer les éléments positifs par leur carré

- Application 12 :

Ecrire un programme qui permet de saisir des nombres entiers dans un tableau à deux dimensions TAB[10][20] et de calculer les totaux par ligne et par colonne dans des tableaux TOTLIG[10] et TOTCOL[20].

- Application 13 :

Ecrire un programme qui permet de chercher une valeur x dans un tableau à deux dimensions t[m][n]. Le programme doit aussi afficher les indices ligne et colonne si x a été trouvé.

Rappel

Tableaux, tableaux dynamiques (vector/string) et **pointeurs**

Un pointeur est une variable spéciale qui peut contenir l'adresse d'une autre variable.

- Si un pointeur P contient l'adresse d'une variable N, on dit que *'P pointe sur N'*.
- Les pointeurs et les noms de variables ont presque le même rôle (à exception prés):
  - Ils donnent accès à un espace mémoire.
  - Un pointeur peut 'pointer' sur différentes adresses tant disque le nom d'une variable reste toujours lié à la même adresse.

Rappel

Tableaux, tableaux dynamiques (vector/string) et **pointeurs**

- **Déclaration**
  - Syntaxe: <type> *<nom>
  - **Exemple**: int *p;
- **Affectation**
  - Syntaxe: <pointeur> = <&(variable)>
  - **Exemple**:
    - Int n=10;
    - Int *p(0);
    - p=&n;
- **Manipulation**
  - cout << "entrer une valeur";
  - cin >> *p; // écrire dans la case mémoire pointée par p
  - cout << "La valeur est : " << *p<< endl;
  - cout << "L'adresse est : " << p<< endl;

Rappel

Tableaux, tableaux dynamiques (vector/string) et **pointeurs**

- Pour demander manuellement une case mémoire, on utilise l'opérateur malloc qui signifie « Memory ALLOCation ».
- malloc est une fonction ne retournant aucune valeur (void)(on n'en reviendra plus tard) :
  - void* malloc(size_t nombreOctetsNecessaires);
    - **Exemple**:
      - int* p= NULL;
      - p = malloc(sizeof(int));
- On peut libérer la ressource après usage via l'opérateur *free*
- *Free également est une fonction ne revoyant aucune valeur.*
  - *void free(void* p);*
    - **Exemple**: *free(*p);

Rappel

Tableaux, tableaux dynamiques (vector/string) et **pointeurs**

L'allocation dynamique d'un tableau est un mécanisme très utile. Elle permet de demander à créer un tableau ayant exactement la taille nécessaire (pas plus, ni moins).

- Si on veut créer un tableau de n élément de type int (par exemple), on fera appel à malloc.
  - **Exemple**:
    - int *t = NULL;
    - t = (int *)malloc(n*sizeof(int));
- Autres fonctions
  - calloc: identique à malloc mais avec initialisation des cases réservées à 0.
    - void* calloc(size_t taille, size_t nombreOctetsNecessaires);
  - realloc : permet d'agrandir une zone mémoire déjà réservée
    - void* realloc(void* tableau, size_t nombreOctetsNecessaires);
    - **Exemple**:
      - t = (int *) calloc (taille, sizeof(int));
      - taille = taille+10;
      - t =(int *) realloc(t, taille*sizeof(int));

Rappel

Tableaux, tableaux dynamiques (vector/string) et **pointeurs**

- En C++, comme avec malloc et calloc, il est possible de demander manuellement une case mémoire, en utilisant l'opérateur new.
- La syntaxe est la suivante: <pointeur> = new type
  - **Exemple**:
    - int *p(0);
    - p = new int;
- On peut accéder à la case et modifier sa valeur
  - **Exemple**: *p = 10;
- On peut libérer la ressource après usage via l'opérateur *delete*
  - **Exemple**: *delete* p;

Rappel

Tableaux, tableaux dynamiques (vector/string) et **pointeurs**

On peut donc créer un tableau dynamique avec operateur new[taille]. L'utilisation de delete[] permettra alors de détruire un tableau précédemment alloué.

- **Pour le cas d'un tableau unidimensionnel, voici, ci-dessous, une illustration**:
  - int i, taille;
  - …
  - cout << " Entrez la taille du tableau: ";
  - cin >> taille;
  - int *t;
  - t = new int[taille];
  - …
  - delete[] t;

Rappel

Tableaux, tableaux dynamiques (vector/string) et **pointeurs**

- **Pour le cas d'un tableau à deux dimensions, voici, ci-dessous, une illustration**:
  - `int **t;`
  - `int nColonnes;`
  - `int nLignes;`
  - `…`
  - `t = new int* [nLignes];`
  - `for (int i=0; i < nLignes; i++)`
    `t[i] = new int[nColonnes];`
  - `…`
  - `delete[] t;`

Rappel

Tableaux, tableaux dynamiques (vector/string) et **pointeurs**

EXERCICES D'APPLICATIONS

- Application 12 :

Ecrivez un programme déclarant une variable i de type int et une variable p de type pointeur sur int. Affichez les dix premiers nombres entiers en :

  - n'incrémentant que *p
  - n'affichant que i

- Application 13 :

Écrire un programme qui lit un entier n au clavier, alloue un tableau de n entiers initialisés à 0, remplir le tableau par des valeurs saisies au claviers et affiche le tableau.

- Application 14 :

Ecrire un programme qui place dans un tableau T les N premiers nombres impairs, puis qui affiche le tableau. Vous accèderez à l'élément d'indice i de t avec l'expression *(t + i).

- Application 15 :

Ecrivez un programme qui demande à l'utilisateur de saisir un nombre n et qui crée une matrice T de dimensions n*n avec un tableau de n tableaux de chacun n éléments. Nous noterons tij=0 j-ème élément du i-ème tableau. Vous initialiserez T de la sorte : pour tous i, j, tij=1 si i=j (les éléments de la diagonale) et tij=0 si i≠j (les autres éléments). Puis vous afficherez T.

- Application 16 :

Écrire un programme allouant dynamiquement un emplacement pour un tableau d'entiers, dont la taille est fournie par l'utilisateur. Utiliser ce tableau pour y placer des nombres entiers lus également au clavier. Créer ensuite dynamiquement un nouveau tableau destiné à recevoir les carrés des nombres contenus dans le premier. Supprimer le premier tableau, afficher les valeurs du second et supprimer le tout.

- Application 17 :

Écrire un programme qui demande à l'utilisateur de lui fournir un nombre entier entre 1 et 7 et qui affiche le nom du jour de la semaine ayant le numéro indiqué (lundi pour 1, mardi pour 2, ... dimanche pour 7).