Algorithmique

FLORIAN LETOMBE
letombe@cril.univ-artois.fr
Bureau 105F

Le Cours

- Unité d'enseignement 1.2
- Module 1.23 : Outils et méthodes informatiques pour le multimédia
- Volume horaire : 30 h (6 h Cours, 12 h TD, 12 h TP)
- Objectifs :
  - s'initier aux bases de l'algorithmique
  - savoir lire un algorithme
  - analyser des problèmes simples
- Pré-requis : aucun !!!
- Contenu :
  - histoire, variables, types de base
  - programmation structurée, structures de contrôle, trace d'un algorithme, fonctions
  - analyse et décomposition de problèmes

Précisions

- Intervenant Cours & TDs :
  - F. Letombe
- Intervenants TPs :
  - J. Hondermarck
  - F. Letombe

- Modalités de contrôle de connaissances :
  - Théorique :

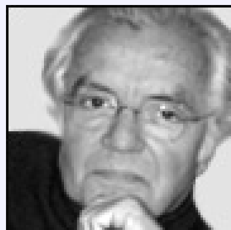
$$max(\frac{DS + Examen}{2}; Examen)$$

  - Pratique : 2 Examens TP
- Calcul de la moyenne :

$$Moyenne = \frac{2}{3} Théorique + \frac{1}{3} Pratique$$

- Préliminaires
  - Un peu d'histoire !
  - Algorithmique et programmation
- Structure séquentielle
  - Constantes
  - Variables
  - Types de base
  - Instructions élémentaires
  - Les types particuliers
- Structure conditionnelle
  - Généralités
  - Le Si
  - Succession de « if »
  - Les Cas
- Structure itérative
  - Généralités
  - Le Tant que
  - Le Pour
  - Le Répéter ... Tant que
  - Récapitulatif

Citation



« *Les préliminaires, c'est mettre le corps à l'ouvrage.* »
Jean-Loup Chiflet

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

Les origines

- Mathématicien persan Al Kwārizmī (780-850) auteur d'un ouvrage décrivant des méthodes de calcul algébrique
- latinisé au Moyen Âge en Algoritmi, puis en Algorisme (« procédure »)
- mot créé par lady Ada Lovelace, fille de lord Byron et assistante de Charles Babbage (1792-1871)
- Définition :

  « *diviser chacune des difficultés que j'examinerois,*
  en autant de parcelles qu'il se pourroit,
  et qu'il seroit requis pour les mieux résoudre. »
  [René Descartes, **Le Discours de la Méthode**]

- le principe des algorithmes était connu depuis l'Antiquité (algorithme d'Euclide), et Donald E. Knuth mentionne même leur usage par les Babyloniens

Donald Ervin Knuth (10/01/1938, Milwaukee, Wisconsin)

- Informaticien de renom
- Professeur émérite en informatique à l’Université de Stanford
- auteur de l’ouvrage « The Art of Computer Programming » (couramment appelé TAOCP), une des références dans le domaine de l’informatique, pour ne pas dire la bible (un mot cher à Knuth ...) des informaticiens
- crée l’analyse d’algorithmes
  - efficacité en temps
  - efficacité en mémoire
  - efficacité moyenne
  - efficacité dans le pire des cas

Et le siècle dernier ?

- années 1930 : A. Turing invente le modèle mathématique des machines de Turing
- années 39-45 : apparition des premiers calculateurs pour coder et décoder les messages
- depuis les années 1950 :
  - A. Turing invente le test de Turing
  - montée en puissance et miniaturisation des ordinateurs
  - développement de l'informatique, traitement automatique de l'information

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

Principe

données $\Longrightarrow$ machine $\Longrightarrow$ *résultats*

Pour que la machine donne les résultats attendus en fonction des données fournies, il faut lui indiquer les actions à exécuter

Principe

données $\Longrightarrow$ machine $\Longrightarrow$ *résultats*

Pour que la machine donne les résultats attendus en fonction des données fournies, il faut lui indiquer les actions à exécuter

Exemple

- machine : distributeur de boisson
- données : pièces fournies, code tapé
- résultat : la boisson désirée
- action : en fonction du code entré, la machine fournit la boisson demandée

Si je demande un café et qu'il est mauvais ou que j'ai une autre boisson, il ne faut pas s'en prendre à la machine mais à la personne qui l'a réglée !!!

Qu'est ce que l'algorithmique ?

Définition *(Algorithme)*

« *Un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver en un nombre fini d'étapes à un résultat.* »

[Encyclopædia Universalis]

Exemple

- Recette de cuisine
- Mode d'emploi d'un appareil
- Indiquer son chemin à un touriste égaré
- Faire fonctionner un répondeur (cf. Muriel Robin)

Méthode

1. Comprendre le problème posé et indiquer les données fournies (input)
2. Indiquer les résultats que l'on souhaite obtenir (output)
3. Déterminer le processus qui permet d'obtenir les résultats à partir des données

Langage de description

Utilisation d'un langage appelé pseudo-code ou LDA (Langage de Description d'Algorithme)

Exemple

Transformation d'un prix en euros :

- donnée fournie : un prix en francs
- résultat attendu : un prix en euros
- méthode : on divise le prix en francs par 6,55957

Algorithme :

écrire(*"donner la somme à convertir :"*);
lire(*somme*);
SommeConvertie ← *somme / 6,55957*;
écrire(*"En euros, cela fait ", SommeConvertie*);

Qualités d’un algorithme

- Être clair, facile à comprendre (bien structuré et documenté)
- Être le plus général possible afin de gérer le plus de cas possibles (si possible, tous les cas)
- Être facile à utiliser (message à l’écran indiquant à l’utilisateur ce qu’il doit faire)
- Être bien conçu afin de limiter le temps des calculs et la place occupée en mémoire

Décomposer le problème

Pour réaliser un bon algorithme, on utilisera la méthode **d'analyse descendante** (top down) qui consiste à décomposer le problème à résoudre en plusieurs sous-problèmes plus simples que l'on traitera séparément, certains d'entre eux pouvant à nouveau être décomposés en sous-problèmes plus simples ...

Exemple

Faire un bon café

1. faire chauffer de l'eau
2. mettre le café
3. verser l'eau

Décomposer le problème (suite ...)

Chacun de ces problèmes peut être détaillé en une suite d'opérations plus simples

Exemple

1. Faire chauffer de l'eau :
   1.1 prendre une casserole
   1.2 mettre de l'eau dans la casserole
   1.3 allumer la plaque
   1.4 attendre l'ébullition
   1.5 éteindre la plaque

Décomposer le problème (suite et fin)

On peut encore détailler certaines opérations

Exemple

1.2 mettre de l'eau dans la casserole
- 1.2.1 mettre la casserole sous le robinet
- 1.2.2 ouvrir le robinet
- 1.2.3 remplir la casserole
- 1.2.4 fermer le robinet

Et cela jusqu'à obtenir des opérations simples compréhensibles par l'exécutant

De l'algorithme au programme

- Une fois l'algorithme écrit, il reste à le traduire dans le langage de programmation utilisé, langage compris par l'ordinateur
- Il y a deux types de langages :
  - Les langages *compilés* (Pascal, ADA, C, **C++**, ...) : le compilateur traduit le programme en langage machine et produit un exécutable
  - les langages *interprétés* (Lisp, Prolog, Scheme, ...) : un interpréteur exécute au fur et à mesure les différentes instructions

Exemple de la conversion en euros

Exemple

```
using namespace std;
#include <iostream>
#define TAUX 6.55957;

void main () {
    float somme, SommeConvertie;
    cout << “Donner la somme à convertir :” << endl;
    cin >> somme;
    SommeConvertie = somme/TAUX;
    cout << “En euros, cela fait “ << SommeConvertie;
}
```

Structures

Pour représenter un algorithme, on dispose des trois possibilités suivantes :

- la <u>structure séquentielle</u> qui indique que les instructions doivent être exécutées les une après les autres
- la <u>structure conditionnelle</u> qui indique quel ensemble d'instructions doit être exécuté selon les circonstances
- la <u>structure itérative</u> qui indique qu'un ensemble d'instructions doit être exécuté plusieurs fois

Citation

« *C'est l'histoire d'un type ...* »
Michel Colucci

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

Déclaration de constantes

- Il est possible de définir des constantes symboliques avec le mot clé **const** (ou avec un **#define**)
- Syntaxe :

const type nom = val;

Exemple

const double TAUX = 6.55957;
Équivalent à :
#define TAUX 6.55957

Il ne sera pas possible de modifier le *TAUX* dans le reste du programme (erreur à la compilation)

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

Déclaration de variables

- Il est nécessaire de préciser le type de données qui sera contenu dans une variable (pour connaître la place occupée en mémoire)
- On déclarera donc, avant toute utilisation d'une variable, les noms des variables utilisées dans le programme précédés du type de données

Exemple

int *heure, minute;*
double *durée, seconde;*

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

Les types élémentaires

- **int** : les entiers (au minimum codés sur 16 bits, permettant de stocker des valeurs $\leq$ 32767)
  Prennent leurs valeurs dans $\mathbb{N}$
- **double** ou **float** : les réels
  Prennent leurs valeurs dans $\mathbb{R}$
  Un réel peut être donné sous :
  - la *forme usuelle* avec le point
  - la *notation scientifique* **aEb**, où **a** est la mantisse et **b** est l'exposant (e.g. 247 = 2.47E2 = 0.247E+3 = 2470E-1 = ...)
- **char** : les caractères ('a', 'b', ..., 'A', ..., ':', ...)
- **bool** : les booléens (vrai ou faux)

Opérations sur les entiers

- addition : opérateur +, opérandes et résultat entiers
- soustraction : opérateur -, opérandes et résultat entiers
- multiplication : opérateur *, opérandes et résultat entiers
- division entière : opérateur /, opérandes et résultat entiers
- reste de la division entière : opérateur %, opérandes et résultat entiers
- division : plus tard ... , résultat réel

Exemple

12/3 = 4
12%3 = 0
7/3 = 2

Opérations sur les entiers (suite et fin)

Opérations de comparaison :

- égal : opérateur $==$
- différent : opérateur $! =$
- supérieur : opérateur $>$
- inférieur : opérateur $<$
- supérieur ou égal : opérateur $>=$
- inférieur ou égal : opérateur $<=$

Priorités :

- *, /, % : $2^{ème}$ priorité
- +, - : $3^{ème}$ priorité
- $=, \neq, <, >, <=, >=$: $4^{ème}$ priorité
- Pour les opérateurs de même priorité, on a une priorité de gauche à droite

Opérations sur les réels

- addition : opérateur +, opérandes et résultat réels
- soustraction : opérateur -, opérandes et résultat réels
- multiplication : opérateur *, opérandes et résultat réels
- division : opérateur /, opérandes et résultat réels

Opérations de comparaison :

Les mêmes que pour les entiers

Priorités :

Les mêmes que pour les entiers

Opérations sur les caractères

- Un caractère peut appartenir au domaine des chiffes de '0'à '9', des lettres de 'A' à 'Z' (majuscules ou minuscules) et des caractères spéciaux ('+', '-', ',', ';', '.', ...)
- Un caractère sera toujours noté entre des apostrophes
- Le caractère blanc (espace) s'écrit ' ', le caractère apostrophe '\'', ...

Opérations de comparaison :

Les mêmes que pour les entiers ; l'ordre utilisé pour comparer les caractères sera l'ordre des caractères du codage ASCII (American Standard Code for Information Interchange)

Opérations sur les booléens

- Un booléen peut prendre deux valeurs :

  true (1) ou false (0)

- Cette valeur peut être le résultat d'une comparaison entre deux variables du même type

Exemple

```
continuer = 'A' < 'Z'; // mis à vrai
n = 3;
continuer = n >= 5; // mis à faux
```

Opérations sur les booléens (suite ...)

- Opérateur **et** (&&)

a	b	$a \&\& b$
0	0	0
0	1	0
1	0	0
1	1	1

- Opérateur **ou** (||)

a	b	$a \mid\mid b$
0	0	0
0	1	1
1	0	1
1	1	1

- Opérateur **non** (!)

a	$!a$
0	1
1	0

Opérations sur les booléens (suite et fin)

Opérations de comparaison :

Les mêmes que pour les entiers ; l'ordre sera le suivant :

false < true (ou 0 < 1)

Priorités :

- non : 1$^{\text{ère}}$ priorité
- et : 2$^{\text{ème}}$ priorité
- ou : 3$^{\text{ème}}$ priorité
- ==, !=, <, <=, >, >= : 4$^{\text{ème}}$ priorité
- Pour les opérateurs de même priorité, on a une priorité de gauche à droite

- Préliminaires
  - Un peu d'histoire !
  - Algorithmique et programmation
- Structure séquentielle
  - Constantes
  - Variables
  - Types de base
  - Instructions élémentaires
  - Les types particuliers
- Structure conditionnelle
  - Généralités
  - Le Si
  - Succession de « if »
  - Les Cas
- Structure itérative
  - Généralités
  - Le Tant que
  - Le Pour
  - Le Répéter ... Tant que
  - Récapitulatif

Lecture/Écriture

Ajouter en tête du fichier source

using namespace std;

#include <iostream>

1. écriture (output)
   - **cout** $<<$ $expr_1$ $<<$ $expr_2$ $<<$ $\ldots$ $<<$ $expr_n$;
   - cette instruction affiche $expr_1$, puis $expr_2$, ..., puis $expr_n$
   - afficher un saut de ligne se fait au moyen de *cout* $<<$ *endl*
   - *cout* désigne le flot de sortie standard
   - on utilise le même opérateur pour afficher des caractères, des entiers, des réels ou des chaînes de caractères

Exemple

cout $<<$ *“En euros, cela fait “* $<<$ *SommeConvertie* $<<$ **endl***;*

Lecture/Écriture (suite et fin)

2 lecture (input)

- **cin** $>>$ *var*$_1$ $>>$ *var*$_2$ $>>$ $\ldots$ $>>$ *var*$_n$ *;*
- cette instruction lit (au clavier) des valeurs et les affecte à *var*$_1$, puis à *var*$_2$, ..., puis à *var*$_n$
- *cin* est le flot d'entrée standard et $>>$ est un opérateur similaire à $<<$

Exemple

cin $>>$ *somme;*

Instructions d'affectation

- Syntaxe :

variable = expression

Exemple

n = 2 + 4 ;
a = a - 5 ;

- Une variable est représentée par un identificateur
- Un identificateur est un nom qui commence par une lettre suivie d'un nombre quelconque de lettres, de chiffres ou du caractère _ (souligné)
- Une expression est une formule mathématique
- Toute variable, avant d'être utilisée doit être initialisée

Commentaires

- Pour commenter une ligne ou une fin de ligne :
  // un commentaire
- Pour commenter une ensemble de lignes :
  / commentaires */*

Exemple

```
// Ceci est un commentaire
x = 10 ; // ceci est aussi un commentaire
y = x + 8 ; /* ceci est encore et
                toujours un commentaire */
```

Instructions d’incrémentation/de décrémentation

Définition *(Incrémenter/Décrémenter)*

Incrémenter (resp. décrémenter) une variable revient à augmenter (resp. diminuer) la valeur de cette variable d’une quantité constante 1

Forme générale :

<*variable*>++;
++<*variable*>;
<*variable*>−−;
−−<*variable*>;

Instructions d'incrémentation/de décrémentation (suite et fin)

Remarques :

- L'expression $i++$ (resp. $++i$) incrémente la variable i et retourne son ancienne (resp. sa nouvelle) valeur
- Lorsqu'elles sont utilisées en dehors d'une expression, $i++$ et $++i$ sont donc équivalentes
- La décrémentation fonctionne selon le même modèle

Exemple

$i = 5; j = 8;$	// i vaut 5 et j vaut 8
$i++;$	// i vaut 6
$i--;$	// i vaut de nouveau 5
$i = j-- + 2;$	// i vaut 10 et j vaut 7
$j = ++i - 3;$	// i vaut 11 et j vaut de nouveau 8

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

Les chaînes de caractères

- La classe **string** en C++
- Pour utiliser ce type, il faut placer en tête du fichier source

#include *<string>*

- Syntaxe :

string *t;* définit une variable de type string
string *s(N, c);* définit une variable *s* de longueur *N* (un entier) où chaque élément est initialisé avec le caractère *c*

Exemple

string mot1 = “bonjour”; définit une variable *mot1* contenant la chaîne *bonjour*
string mot2(10, ’a’) définit une variable *mot2* comprenant 10 occurrences de la lettre *a*

Opérations sur les chaînes de caractères

- *s.size()* représente la longueur de la chaîne *s*

Exemple

s = “bonjour”;
s.size() vaut donc 7

- *s[i]* désigne le $i^{ème}$ caractère de la chaîne *s*, avec $i = 0, 1, \ldots, s.size() - 1$

Exemple

s[3] vaut ’j’

- l’opérateur + : opérateur de concaténation

Exemple

t = “ le monde”;
s+t vaut *bonjour le monde*

Les tableaux à une dimension

- Il est très souvent utile de stocker en mémoire un certain nombre de valeurs de même type
- On utilise alors la notion de tableau (ou vecteur) qui permet de garder en mémoire ces valeurs

Exemple

- On veut stocker en mémoire pour chacune des journées de la semaine le nombre de calories pris lors d'un repas
- On définira le tableau suivant :
  vector<int> tableau(7);
- La case 1 contiendra le nombre de calories du lundi, la case 2 celui du mardi, etc
- On pourra alors réaliser différentes opérations, sans avoir à ressaisir toutes les valeurs (étude comparative entre les jours, calcul du max, du min, du total sur la semaine, ...)

Les tableaux à une dimension (suite et fin)

- La classe **vector** en C++
- Pour utiliser ce type, il faut placer en tête du fichier source

  #include *<vector>*
- Il est possible de définir des « tableaux simples » mais c'est une autre histoire ...

- Syntaxe :
  - **vector<**_char_**>** *t(100, 'a');* définit un tableau *t* de *100* caractères dont chaque case est initialisée à *a*
  - **vector<**_char_**>** *t(50);* définit un tableau *t* de longueur *50* initialisé à *0*
  - **vector<**_double_**>** *t;* définit un tableau de réels, vide

Opérations sur les tableaux à une dimension

- La fonction *size()*
- Le $i^{ème}$ élément (*[i]*)
- La copie d'un tableau dans un autre

Exemple

vector<*int*> *t1(50, 10);*
vector<*int*> *t2 = t1;*
Le tableau t2 contiendra alors 50 cases initialisées à 10

- Il est possible de créer des tableaux à dimension multiple (deux, trois, etc) mais encore une fois, c'est une autre histoire ...
- Nous reviendrons sur les tableaux un peu plus tard

Citation

« *L'estime de soi ne se conjugue pas au conditionnel.* »
Bill Watterson
Calvin et Hobbes - Complètement surbookés !

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

Une moyenne en algorithmique

Exemple

On souhaite écrire un algorithme qui calcule la moyenne de contrôle en algorithmique pour un étudiant en utilisant la formule

$$max(\frac{DS + Examen}{2}; \frac{Examen}{2})$$

L'algorithme est le suivant :

- demander la note de DS
- demander la note d'examen
- calculer la moyenne des deux notes
- si cette moyenne est supérieure à la note d'examen, afficher cette moyenne
- sinon, afficher la note d'examen

L'algorithme traduit en C++

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

void main () {
    double ds, exam, moyenne;
    cout << "Donner la note de DS :" << endl;
    cin >> ds;
    cout << "Donner la note d'examen :" << endl;
    cin >> exam;
    moyenne = ds+exam/2;
    if (moyenne > exam)
      cout << "Note : " << moyenne << endl;
    else cout << "Note : " << exam << endl;
}
```

Forme générale

if *(<expression>)*
<séquence n°1 d'instructions>
else *<séquence n°2 d'instructions>*

- Si l'expression est vraie, la séquence n°1 d'instructions sera exécutée et la séquence n°2 d'instructions ignorée
- Si l'expression est fausse, la séquence n°2 d'instructions sera exécutée et la séquence n°1 d'instructions ignorée

Attention à l'indentation !!!

Forme générale (suite et fin)

- L'expression booléenne : on privilégie les parenthèses afin d'éviter les confusions entre les priorités des opérateurs
- Une séquence d'instruction(s) :
  - Si elle n'est composée que d'une instruction unique, alors inutile de créer un nouveau bloc d'instructions
  - Si elle contient plusieurs instructions, alors il est nécessaire d'encadrer cette séquence de { et }

Exemple

```
if (a && b) {
    instruction_1;
    ...
    instruction_n;
}
else instruction_else;
```

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

L'instruction **if**

Dans certains cas, lorsque l'expression est fausse, aucune instruction n'est à exécuter. On utilise alors la forme suivante :

if *(<expression>)*
<séquence d'instructions>

Exemple

Dans le programme précédent, on peut remplacer la structure conditionnelle par :

```
if (moyenne > exam)
    exam = moyenne;
cout << "Note : " << exam << endl;
```

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

Imbrications

Il est possible d'imbriquer des structures conditionnelles entre elles (ne pas oublier d'indenter pour une bonne lisibilité de l'algorithme)

Exemple

```
if (note < 10)
  cout << "Vous êtes recalé." << endl;
else if (note < 12)
        cout << "Vous êtes reçu" << endl;
      else if (note < 14)
              cout << "Bravo, mention AB << endl;
            else if (note < 16)
                    cout << "Super, mention B << endl;
                  else cout << "Excellent, mention TB << endl;
```

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

L'instruction **switch**

- Dans l'exemple précédent, le choix ne se limite pas à une alternative
- Lorsque l'on a un choix multiple sur une variable de <u>**type énuméré**</u> (entier, caractère), on peut le traduire en C++ par :

switch *(<expression>) {*
    case *<expr const 1> : <séquence d'instructions 1>*
    case *<expr const 2> : <séquence d'instructions 2>*
    ...
    case *<expr const n> : <séquence d'instructions n>*
    default *: <séquence d'instructions n+1>*
}

L'instruction **switch** (suite ...)

Interprétation :

- <expression> est évaluée et fournit une valeur *v*
- Si *v* est une des valeurs <expr const 1>,. . . ,<expr const n>, alors l'exécution se poursuit par les instructions du cas correspondant et ce jusqu'à la prochaine instruction **break;** ou la dernière instruction de la séquence d'instructions n+1
- Si aucune des <expr const i> n'est vraie, la séquence d'instructions n+1 qui suit le **default** est exécutée

Remarques :

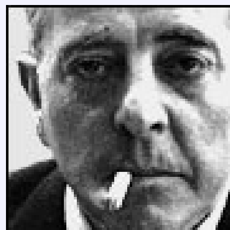
- <expr const i> doit être une constante de type entier ou caractère, et <expr const i> $\neq$ <expr const j>, $\forall i \neq j$
- la ligne **default ...** n'est pas obligatoire

L'instruction **switch** (suite et fin)

Exemple

```
char c;
cout << "Avez-vous au moins 10 (o ou n) ?" << endl;
cin >> c;
switch (c) {
case 'o' : cout << "Vous êtes reçu :)" << endl;
           break;
case 'n' : cout << "Vous êtes recalé :(" << endl;
           break;
default : cout << "Revoyez votre alphabet !!!" << endl;
}
```

Citation



« *Si quelqu'un vous dit : "Je me tue à vous le répéter", laissez-le mourir.* »

Jacques Prévert

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- **Généralités**
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

- Il arrive très souvent qu'une séquence d'instructions soit à répéter un certain nombre de fois
- Deux solutions :
  - Écrire cette séquence autant de fois que nécessaire, à condition de connaître ce nombre de fois (bon courage !!!)
  - Utiliser une structure itérative, valable même si on ne connaît pas ce nombre de tours à effectuer

Exemple

On désire calculer le *pgcd* (plus grand commun diviseur) de deux entiers naturels en utilisant l'algorithme d'Euclide

a = 1272 et b = 408

1272 = 408*3+48

408 = 48*8+24

48 = 24*2+0

Alors $pgcd(1272, 408) = pgcd(408, 48) = pgcd(48, 24) = 24$

Algorithme d'Euclide

Exemple

```
int n1, n2, c1, c2, r;
cout << "Donner les entiers n1 et n2" << endl;
cin >> n1 >> n2;
c1 = n1;
c2 = n2;
while (c2 != 0) {
    r = c1 % c2;
    c1 = c2;
    c2 = r;
}
cout<<"pgcd("<<n1<<","<<n2<<")="<<c1<< endl;
```

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

L'instruction **while**

Forme générale :

while (*<expression logique>*)
<séquence d'instructions>

1. L'expression logique est évaluée
   Si elle est à vrai :
   2. La séquence d'instructions est réalisée
   3. On retourne au point 1

   Si elle est à faux, on exécute l'instruction qui suit la séquence d'instructions

Remarques :

- Vérifier que la valeur de l'expression logique à l'entrée dans la boucle est connue
- Vérifier que la valeur de vérité de l'expression logique peut varier durant la boucle afin d'éviter les boucles infinies

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- **Le Pour**
- Le Répéter ... Tant que
- Récapitulatif

L'instruction **for**

- Il arrive que l'on connaisse le nombre exact d'exécutions de la boucle
- Dans ce cas, on peut utiliser une boucle **for**

Exemple

On désire calculer la puissance $n^{ème}$ d'un réel x (i.e. x^n)

```
cout << "Donner la valeur de x et celle de n :" << endl;
cin >> x >> n;      // x et n ont été déclarées au préalable
puiss = 1;          // initialisation de puiss
for (i = 0 ; i < n ; i++)
    puiss = puiss*x;

cout << x << " puissance " << n
     << " est " << puiss << endl;
```

L'instruction **for** (suite ...)

Forme générale :

for *(<expression 1> ; <expression 2> ; <expression 3>)*

<séquence d'instructions>

1. L'expression 1 est exécutée
2. La valeur de l'expression 2 est évaluée

   Si elle est à vrai :

   3. La séquence d'instructions est exécutée
   4. L'expression 3 est exécutée
   5. On retourne au point 2

   Si elle est à faux, on exécute l'instruction qui suit la séquence d'instructions

L'instruction **for** (suite et fin)

Remarques :

- Les expressions 1, 2 et 3 sont généralement dépendantes de la même variable appelée *variable de contrôle*
- On peut utiliser la valeur de la variable de contrôle dans la séquence d'instructions
- Il est **INTERDIT** de modifier la valeur de la variable de contrôle dans la séquence d'instructions
- Schématiquement, on peut décrire une boucle **for** ainsi :

```
<expression 1>;
while (<expression 2>) {
    <séquence d'instructions>;
    <expression 3>;
}
```

Préliminaires
- Un peu d'histoire !
- Algorithmique et programmation

Structure séquentielle
- Constantes
- Variables
- Types de base
- Instructions élémentaires
- Les types particuliers

Structure conditionnelle
- Généralités
- Le Si
- Succession de « if »
- Les Cas

Structure itérative
- Généralités
- Le Tant que
- Le Pour
- Le Répéter ... Tant que
- Récapitulatif

L'instruction **do**

- Il arrive que l'on soit certain que la boucle s'exécutera au moins une fois
- Dans ce cas, on privilégie l'emploi de la boucle **do ... while**

Forme générale :

do
    <séquence d'instructions>
while*(<expression logique>)*

1. La séquence d'instructions est réalisée
2. L'expression logique est évaluée

    Si elle est à vrai, on retourne au point 1
    Si elle est à faux, on exécute l'instruction qui suit le **while**

L'instruction **do** (suite et fin)

Remarques :

- La condition de sortie de la boucle **do** fonctionne exactement comme celle du **while**
- Certains langages ont compliqué cette condition de sortie de la boucle en faisant correspondre l'expression logique avec la sortie de la boucle (le **repeat ... until**) alors qu'elle correspond ici à la continuation de la boucle

Exemple

Soit c une variable de type caractère

```
do
    cout << "Voulez-vous continuer (o ou n) ?" << endl;
    cin >> c;
while ((c != 'o') || (c != 'n'))
```

Préliminaires
Un peu d'histoire !
Algorithmique et programmation
Structure séquentielle
Constantes
Variables
Types de base
Instructions élémentaires
Les types particuliers
Structure conditionnelle
Généralités
Le Si
Succession de « if »
Les Cas
Structure itérative
Généralités
Le Tant que
Le Pour
Le Répéter ... Tant que
Récapitulatif

Les boucles

- Lorsque vous avez une boucle à implémenter, trois cas se présentent à vous :
  1. Vous connaissez le nombre de tours dans la boucle :

     utilisez le **for**
  2. Vous savez qu'au moins un tour va être effectué :

     utilisez le **do ... while**
  3. Dans tous les autres cas :

     utilisez le **while**

- Cependant, il est toujours possible de substituer un type de boucle par un autre

Attention à l'indentation !!!

Les boucles

- Lorsque vous avez une boucle à implémenter, trois cas se présentent à vous :
  1. Vous connaissez le nombre de tours dans la boucle :
     utilisez le **for**
  2. Vous savez qu'au moins un tour va être effectué :
     utilisez le **do ... while**
  3. Dans tous les autres cas :
     utilisez le **while**
- Cependant, il est toujours possible de substituer un type de boucle par un autre

Attention à l'indentation !!!

Les boucles

- Lorsque vous avez une boucle à implémenter, trois cas se présentent à vous :
  1. Vous connaissez le nombre de tours dans la boucle :
     utilisez le **for**
  2. Vous savez qu'au moins un tour va être effectué :
     utilisez le **do ... while**
  3. Dans tous les autres cas :
     utilisez le **while**
- Cependant, il est toujours possible de substituer un type de boucle par un autre

Attention à l'indentation !!!