

Chapitre 2

PROGRAMMES SIMPLES

POLYTECHNIQUE
MONTRÉAL

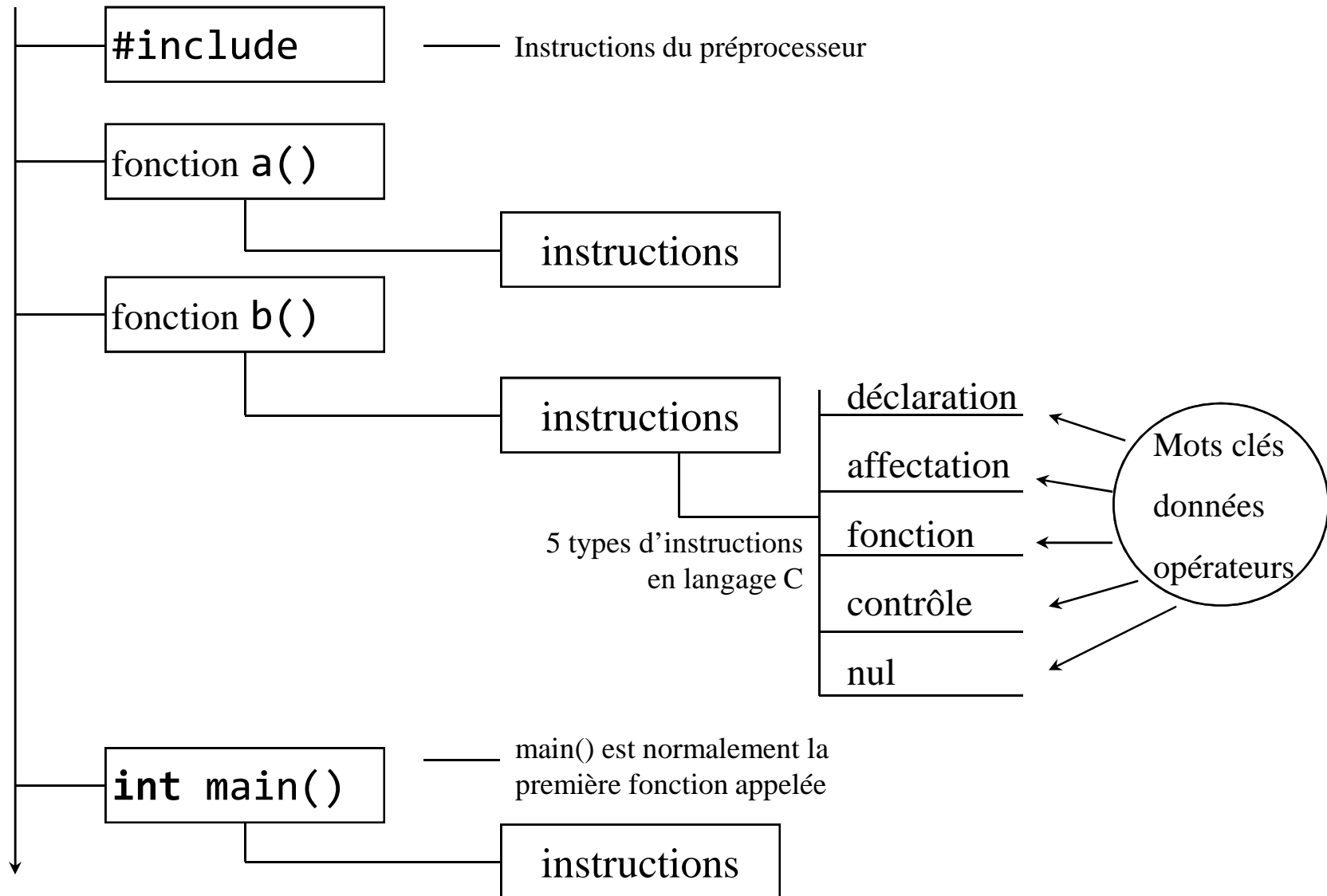
LE GÉNIE
EN PREMIÈRE CLASSE




```
//- EN-TETE DU PROGRAMME -  
/**  
 * Ce programme calcule et affiche la factorielle d'un nombre lu du clavier.  
 * \file    factorielle.cpp  
 * \author  Yves BOUDREAUULT  
 * \date    10 Septembre 2009  
 **/  
  
#include <iostream>    // Pour l'utilisation de cin et cout.  
using namespace std;  
  
int main()    // Fonction principale, début du programme.  
{  
    //- DECLARATION DES CONSTANTES -  
    const int NOMBREMAX = 8;  
  
    //- ENTRÉE DES DONNÉES -  
    cout << "Entrer un nombre inférieur à "  
         << NOMBREMAX << " : ";  
    int nombre;           // Variable de lecture d'un nombre entier.  
    cin >> nombre;        // Lecture d'un nombre.  
  
    //- TRAITEMENT -  
    int factorielle;      // Variable du résultat.  
    if (nombre == 0)      // Si le nombre lu est zéro.
```

voir la suite dans le fichier II_factorielle.cpp

Structure d'un programme



Identificateur

 Identificateur: nom d'une entité.

 Nom d'une variable, constante, fonction...


 Composé de:

 A-Z a-z 0-9 (caractères alphanumériques)

 _ (soulignement)

 ne débutant pas par un nombre.

 C++11 supporte les caractères accentués.

 Les lettres minuscules et majuscules sont différenciées.
Ainsi TP1, Tp1, tP1, tp1 sont quatre identificateurs distincts.

Les déclarations

- ❑ constantes `const int MAX = 15;`
- ❑ variables `int age = 10;`
- ❑ définitions de types `typedef double Reel;`
`struct Point { Reel x,y; };`
- ❑ prototypes de sous-programmes `int sommer(int,int);`
- ❑ macros `#define IFDEBUG(x) x`

Instruction simple

Chaque ligne, terminée par un point-virgule :

```
int nbJours = 30;
```

```
cout << "Donner deux valeurs entières ";
```

```
int nombre1, nombre2;
```

```
cin >> nombre1 >> nombre2;
```

```
cout << nbJours << nombre1 << nombre2;
```

II_instruction_simple.cpp

Instruction composée


Bloc d'instructions simples entre accolades.

Les variables d'un bloc n'existent plus après le bloc.

```
{
    int numero = 12345;
    string alliage = "aluminium";
    double enStock = 3.45e12;
    cout << numero << alliage << enStock;
}
// numero, alliage et enStock n'existent plus.
{
    cout << "Donner votre numéro de code";
    int numCode;
    cin >> numCode;
    cout << numCode;
}
// numCode n'existe plus.
```

II_instruction_composee.cpp

VARIABLE

 Il s'agit d'une donnée située à un emplacement mémoire dont la valeur peut être modifiée.

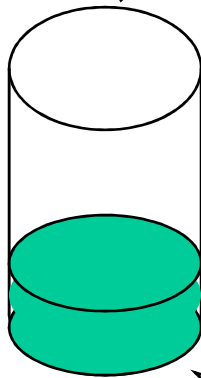
TYPE

 Le type correspond à la sorte de variable;

 Le type précise pour la variable:

- l'espace mémoire occupé;
- la représentation mémoire: la façon dont elle est mémorisée;
- les opérations admissibles.

Une variable peut être comparée à ce verre. On peut ajouter et retirer du liquide selon sa bonne volonté.



Analogie:

Verre	Variable
- Contenant	- Adresse mémoire
- Contenu	- Valeur

Le type indique :

- la grandeur du verre,
- le liquide admis,
- la façon que ce liquide doit être manipulé.

Types de base les plus utilisés

Type	Description	Syntaxe	Valeurs*	Taille*
int	nombre entier	123	± 2 milliards	4 octets
unsigned	nombre entier ≥ 0 ; utilisé pour les tailles	123U sizeof(x)	0 à 4 milliards	4 octets
double	nombre réel	123.0 6.02e23	$\pm 10^{\pm 308}$ ~16 chiffres signif.	8 octets
char	caractère ; élément de base pour les textes	'A' (entre apostrophes)	0-9 A-Z a-z ! " # \$ & ...	1 octet
bool	booléen (vrai ou faux) ; pour les conditions	true false	true false	1 octet

Types les plus utilisés pour les textes

tableau de char	chaîne de caractères style C	"bonjour"
string	chaîne de caractères style C++	string("bonjour")

Note*: Les valeurs limites (expliquées au chapitre 8) et tailles peuvent varier selon l'ordinateur ou le compilateur.

LES ENTIERS

Opérateurs arithmétiques

$+$, $-$, $*$, $/$, $\%$

Opérateurs $/$ et $\%$ $(a/b)*b + a\%b$ est égal à a

Opérateur	Opération	Expression	Résultat
$/$	Quotient (division)	$11 / 4$	2
	(l'arrondi est vers zéro)	$-11 / 4$	-2 (en C++11)
$\%$	Reste (« modulo »)	$11 \% 4$	3
	(même signe que le numérateur)	$-11 \% 4$	-3 (en C++11)

Opérateurs relationnels

$==$, $<$, $>$, $<=$, $>=$, $!=$

LES RÉELS

Opérateurs arithmétiques

$+$, $-$, $*$, $/$

✋ La division entre des opérandes réels donne un résultat réel

➔ $5.0 / 2.0$ ou $5 / 2.0$ ou $5.0 / 2$ donne 2.5

Opérateurs relationnels

$==$, $<$, $>$, $<=$, $>=$, $!=$

(attention que l'égalité est sensible aux imprécisions de calculs,
par exemple $1.0 / 49 * 49 \neq 1.0$)

Égalité entre deux réels

```
double valeurEsperée = 13.567;

cout << "Donner une valeur réelle: ";
double valeurLue;
cin >> valeurLue;

// Valeur réelle égale à 3 chiffres après la virgule:
// (boolalpha pour afficher true ou false plutôt que 1 ou 0)
cout << boolalpha << (fabs(valeurEsperée - valeurLue) < 0.0005) << endl;
```

II_reel_approximation.cpp

Fonctions prédéfinies

- Rôle : exécuter une action spécifique
 - Inclues dans des bibliothèques (« librairies ») de fonctions
- ```
#include <cmath> // bibliothèque mathématique
#include <string> // bibliothèque pour les chaînes de caractères
```

***type du résultat nomfonction (liste et type des paramètres)***

(***type du résultat*** est ***void*** si aucun résultat)

- Lors de l'exécution de la fonction, les paramètres sont remplacés par les arguments (valeurs données aux paramètres)

# Fonctions prédéfinies

- Signature ou prototype de la fonction

*double* pow(*double* x, *double* y) //  $x^y$  (*x* exposant *y*)

Type de retour

2 paramètres de  
type double

- Appel de la fonction

```
double base = 4.2, exposant = 2.1;
double resultat = pow(base, exposant);
// ou
// resultat = pow(4.2, 2.1);
```

arguments

# Appel de fonctions

Alternative 1: fonction globale

**retour = nomFonction (paramètres)**

*double* cos(*double* x)    cos(3.14)  $\approx$  -1.0

*int* isalpha(*int* x)    isalpha('d') = true

Alternative 2: fonction membre

**retour = variableObjet.nomFonction(paramètres)**

*chaine.size()*

string chaine = "bonjour les amis";

cout << chaine.size();

// affiche 16

Alternative 3: fonction membre modifiant la variable objet

**variableObjet.nomFonction(paramètres)**

*chaine.insert(pos, ajout)*

string chaine = "la lo lu";

chaine.insert(3, "le li ");

// chaine contient "la le li lo lu"



# Fonctions sur les entiers et réels

⇒ Définies dans le fichier **cmath**

| Fonction                                           | Description                                        | Exemple                          |
|----------------------------------------------------|----------------------------------------------------|----------------------------------|
| <i>double</i> <code>cos(double x)</code>           | Cosinus de $x$ (en radian)                         | <code>cos(3.14) ≈ -1.0</code>    |
| <i>double</i> <code>sin(double x)</code>           | Sinus de $x$ (en radian)                           | <code>sin(1.57) ≈ 1.0</code>     |
| <i>double</i> <code>exp(double x)</code>           | $e^x$ ( $e$ exposant $x$ , où $e = 2,71828\dots$ ) | <code>exp(2.0) ≈ 7.389</code>    |
| <i>double</i> <code>pow(double x, double y)</code> | $x^y$ ( $x$ exposant $y$ )                         | <code>pow(3.0, 2.0) = 9.0</code> |
| <i>double</i> <code>log(double x)</code>           | Logarithme naturel de $x$ ( $\ln x$ )              | <code>log(7.389) ≈ 2.0</code>    |

Notation: *type du résultat* `fonction`(*type des paramètres*)

Les types ne doivent pas être écrits dans le programme lors de l'utilisation de la fonction.

# Fonctions sur les entiers et réels

⇒ Définies dans le fichier **cmath**

| Fonction                                         | Description                                               | Exemple                    |
|--------------------------------------------------|-----------------------------------------------------------|----------------------------|
| <i>double</i> <b>log10</b> ( <i>double</i> $x$ ) | Logarithme en base 10 de $x$                              | <b>log10</b> (100.0) = 2.0 |
| <i>double</i> <b>sqrt</b> ( <i>double</i> $x$ )  | Racine carrée de $x$                                      | <b>sqrt</b> (16.0) = 4.0   |
| <i>int</i> <b>abs</b> ( <i>int</i> $x$ )         | Valeur absolue de $x$ (entier)                            | <b>abs</b> (-2) = 2        |
| <i>double</i> <b>fabs</b> ( <i>double</i> $x$ )  | Valeur absolue de $x$ (réel)                              | <b>fabs</b> (-12.3) = 12.3 |
| <i>double</i> <b>ceil</b> ( <i>double</i> $x$ )  | $\lceil x \rceil$ Arrondi au plus petit entier $\geq x$   | <b>ceil</b> (9.2) = 10.0   |
| <i>double</i> <b>floor</b> ( <i>double</i> $x$ ) | $\lfloor x \rfloor$ Arrondi au plus grand entier $\leq x$ | <b>floor</b> (9.8) = 9.0   |

# Le type booléen en C

- ❑ Il n'existait pas de type booléen proprement dit en langage C (avant C99).
- ❑ Est considérée FAUSSE toute variable, constante ou expression qui vaut 0 et VRAIE toute variable, constante ou expression différente de 0.

| Opération                   | Opérateur |
|-----------------------------|-----------|
| Négation                    | !         |
| Conjonction (ET)            | &&        |
| Disjonction inclusive (OU)  |           |
| Disjonction exclusive (OUE) | ^         |

## Le type booléen en C

- ❑ Une variable de type entier est souvent utilisée comme variable booléenne. Le contexte d'utilisation devrait préciser la nature booléenne de cette variable entière.
- ❑ Le résultat d'une expression booléenne est soit 0 (pour faux) ou 1 (pour vrai).

## Le type booléen en C++

- ⊗ Le C++ offre un type booléen.
- ⊗ L'identificateur de type est *bool*
- ⊗ Les deux seules valeurs que peut prendre une variable de type *bool* sont *true* ou *false*.
- ⊗ Il est préférable d'utiliser ce type booléen lorsqu'il s'agit d'une variable ne manipulant que deux valeurs.
- ⊗ Augmente la compréhensibilité d'un programme.
- ⊗ Le manipulateur d'E/S *boolalpha* permet d'afficher une chaîne correspondante à *true* ou *false*, autrement les valeurs 0 ou 1 sont affichées.

# Le type caractère

- ❑ L'identificateur de type est *char*
- ❑ Un caractère est identifié en le plaçant entre apostrophes:
  - `char lettre = 'A';`
- ❑ Un caractère est représenté par un entier, les opérateurs sur les entiers peuvent donc être utilisés; ex.: `'A'+1 == 'B'`
- ❑ Il est possible d'utiliser les opérateurs relationnels sur les caractères ( `==`, `!=`, `<`, `>`, `<=`, `>=` ). Noter que toutes les majuscules sont avant les minuscules (`'Z' < 'a'`).
- ❑ Les fonctions sur les expressions de type *char* sont définies dans le fichier **cctype**

# Le type caractère

```
cout << "Veuillez entrer un caractère: ";
char lettre;
cin >> lettre;

// - Uniquement avec les opérations de base (pas besoin de "ctype") -
if ((lettre >= 'a' && lettre <= 'z') || (lettre >= 'A' && lettre <= 'Z')) {
 cout << "Est un caractère alphabétique." << endl;
 cout << "Le code ASCII de la lettre est " << int(lettre) << endl;
 cout << "La lettre suivante est " << char(lettre + 1) << endl;
 if (lettre >= 'a' && lettre <= 'z')
 cout << "La lettre en majuscule est: " << char(lettre + ('A' - 'a'))
 << endl;
}
else if (lettre >= '0' && lettre <= '9')
 cout << "Est un chiffre." << endl;
else
 cout << "N'est pas alphanumérique." << endl;

// - En utilisant les fonctions de "ctype" (seulement les différences) -
if (isalpha(lettre)) {
 // ...
 if (islower(lettre))
 cout << char(toupper(lettre));
}
else if (isdigit(lettre))
 cout << "Est un chiffre." << endl;
else
 cout << "N'est pas alphanumérique." << endl;
```

II\_caracteres.cpp

# Fonctions sur les caractères

⇒ Définies dans le fichier **cctype**

| Fonction                        | Description                                                                  | Exemple                          |
|---------------------------------|------------------------------------------------------------------------------|----------------------------------|
| <code>int isalpha(int x)</code> | Teste si <i>x</i> est une lettre de l'alphabet, soit 'A'...'Z' ou 'a'...'z'. | <code>isalpha('d') = true</code> |
| <code>int isdigit(int x)</code> | Teste si <i>x</i> est un chiffre, soit '0'...'9'.                            | <code>isdigit('2') = true</code> |
| <code>int isalnum(int x)</code> | Teste si <i>x</i> est alphanumérique (lettre ou chiffre).                    | <code>isalnum('X') = true</code> |
| <code>int islower(int x)</code> | Teste si <i>x</i> est une lettre minuscule, soit 'a'...'z'.                  | <code>islower('a') = true</code> |
| <code>int isupper(int x)</code> | Teste si <i>x</i> est une lettre majuscule, soit 'A'...'Z'.                  | <code>isupper('T') = true</code> |
| <code>int tolower(int x)</code> | Convertit un caractère en minuscule                                          | <code>tolower('S') = 's'</code>  |
| <code>int toupper(int x)</code> | Convertit un caractère en majuscule                                          | <code>toupper('c') = 'C'</code>  |

Noter que ces fonctions donnent le caractère résultant dans un entier, il faut le reconverter en **char** pour l'afficher comme tel. Ex.: `cout << char(toupper('c'));`



# Les chaînes de caractères en C++

- ❑ Déclaration: `string laChaine;`
- ❑ Une chaîne de caractères C est identifiée en la plaçant entre guillemets (l'affectation à la `string` la rend C++).
  - `string laChaine = "Que la vie est belle! ";`
- ❑ L'accès à un caractère de la chaîne s'effectue en précisant sa position:
  - `laChaine[2]` correspond au troisième caractère de la chaîne
  - **Le premier caractère d'une chaîne est à la position 0**
- ❑ Il est possible d'utiliser les opérateurs relationnels sur les chaînes de caractères en se basant sur l'ordre alphabétique. (`==`, `!=`, `<`, `>`, `<=`, `>=`)
- ❑ Les fonctions agissant sur les chaînes de caractères sont dans le fichier **string**.

# Fonctions sur les chaînes de caractères

⇒ Définies dans le fichier **string**

| Fonction                               | Description                                                                                                                                                                                                                             | Exemple                                                                                                                                                                 |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>dest = source;</code>            | Copie la chaîne <i>source</i> dans la chaîne <i>destination</i> .<br><br>La <i>source</i> peut aussi être une chaîne C.                                                                                                                 | <code>chaîne = "bonjour";</code><br><code>// chaîne contient "bonjour"</code>                                                                                           |
| <code>dest += source;</code>           | Concatène la chaîne ou caractère <i>source</i> à la chaîne <i>destination</i> .<br><br>La <i>source</i> peut aussi être une chaîne C.                                                                                                   | <code>// si chaîne contient "bonjour"</code><br><code>chaîne += " les ami";</code><br><code>chaîne += 's';</code><br><code>// chaîne contient "bonjour les amis"</code> |
| <code>dest = source1 + source2;</code> | Concaténation de deux chaînes, ou une chaîne et un caractère, le résultat est affecté à <i>dest</i> .<br><br><b>Il doit y avoir une 'string' C++ au moins d'un côté de l'addition</b> , l'autre peut être une chaîne C ou un caractère. | <code>chaîne = "bonjour";</code><br><code>chaîne = chaîne + " les ami"</code><br><code>          + 's';</code><br><code>// chaîne contient "bonjour les amis"</code>    |

# Fonctions sur les chaînes de caractères (suite)

⇒ Définies dans le fichier **string**

| Fonction                                                                                                                    | Description                                                                                                                                                                                                                                                                                                                            | Exemple                                                                                                                                                                                        |
|-----------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>chaine.size()</code><br>ou <code>chaine.length()</code>                                                               | Retourne la longueur de la <i>chaine</i> .<br><b>Attention:</b> le type du résultat est <code>size_t</code> , qui est <b>unsigned</b> .                                                                                                                                                                                                | <pre>chaine = "bonjour les amis"; cout &lt;&lt; chaine.size(); // affiche 16</pre>                                                                                                             |
| $chaine1 \lesseqgtr chaine2$<br>(où $\lesseqgtr$ est un des opérateurs relationnels)                                        | Compare <i>chaine1</i> à <i>chaine2</i> selon l'ordre alphabétique.<br><b>Il doit y avoir une 'string' C++ au moins d'un côté de l'opérateur</b> , l'autre peut être une chaîne C.<br><b>Attention:</b> les minuscules et les majuscules sont distinctes.                                                                              | <pre>// si chaine1 contient "bonjour" // et chaine2 contient "bonsoir"  chaine1 &lt; chaine2 vaut true chaine2 &lt; chaine1 vaut false chaine1 != chaine2 vaut true</pre>                      |
| <code>chaine.find(<i>autre</i>)</code><br><br><code>chaine.find(<i>autre</i>,<br/>                  <i>position</i>)</code> | Cherche la chaîne de caractères <i>autre</i> dans <i>chaine</i> et retourne sa position.<br>Si <i>autre</i> n'est pas dans <i>chaine</i> , la valeur de retour est <code>chaine.npos</code> .<br>Cherche la chaîne de caractères <i>autre</i> dans <i>chaine</i> à partir de <i>position</i> et retourne la position d' <i>autre</i> . | <pre>chaine = "Bon matin"; autre = "matin";  cout &lt;&lt; chaine.find(autre); // affiche 4  chaine = "bonjour les amis"; autre = " "; cout &lt;&lt; chaine.find(autre,8); // affiche 11</pre> |

# Fonctions sur les chaînes de caractères (suite)

⇒ Définies dans le fichier **string**

| Fonction                                                                          | Description                                                                                                                                                                                    | Exemple                                                                                         |
|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <code>chaine.erase(pos, nbre);</code>                                             | Enlève <i>nbre</i> caractères de la <i>chaine</i> , à partir de la position <i>pos</i> .                                                                                                       | <pre>chaine = "bonjour les amis"; chaine.erase(7, 4); // chaine contient "bonjour amis"</pre>   |
| <code>chaine.insert(pos, ajout);</code>                                           | Insère <i>ajout</i> dans <i>chaine</i> , à partir de la position <i>pos</i> .                                                                                                                  | <pre>chaine = "la lo lu"; chaine.insert(3, "le li "); // chaine contient "la le li lo lu"</pre> |
| <code>chaine.resize(longueur);</code><br><code>chaine.resize(longueur, c);</code> | <p>Redimensionne la <i>chaine</i> à la <i>longueur</i> spécifiée.</p> <p>La chaîne peut être tronquée ou augmentée (en ajoutant des copies du caractère <i>c</i>, qui par défaut est nul).</p> | <pre>chaine = "bon beigne"; chaine.resize(7); // chaine contient "bon bei"</pre>                |

# Fonctions sur les chaînes de caractères (suite)

⇒ Définies dans le fichier **string**

| Fonction                                       | Description                                                                                                                                               | Exemple                                                                                                                             |
|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <code>chaine.substr(pos, long);</code>         | Retourne la sous-chaîne de longueur <i>long</i> à partir de la position <i>pos</i> .                                                                      | <pre>chaine = "bonjour les amis"; autre = chaine.substr(8, 3); // autre contient "les"</pre>                                        |
| <code>chaine.replace(pos, long, autre);</code> | Remplace <i>long</i> caractères dans <i>chaine</i> par <i>autre</i> , à partir de la position <i>pos</i> ; <i>chaine</i> peut être augmentée ou diminuée. | <pre>chaine = "milou et tintin"; chaine.replace(6, 2,                "le chien de"); // chaine est "milou le chien de tintin"</pre> |
| <code>chaine.c_str();</code>                   | Retourne <i>chaine</i> sous forme de chaîne de caractères C;<br>la chaîne C est « const » (ne peut pas être modifiée).                                    | <pre>char strC[40]; strcpy_s(strC, chaine.c_str()); // strC contient une copie de la chaine</pre>                                   |

# Le type pointeur

- ❑ Une variable pointeur correspond à une variable qui contient l'adresse mémoire d'une donnée.
- ❑ Déclaration: `int* ptrEntier;`
- ❑ L'opérateur & «adresse de» est utilisé pour connaître l'adresse d'une donnée.

```
int entier;
int* ptrEntier = &entier;
```

- ❑ L'identificateur d'une chaîne de caractères C contient l'adresse du premier caractère de la chaîne, il correspond donc à un pointeur.
- ❑ Attention aux multiples significations de \* et &, selon qu'ils sont utilisés comme opérateur binaire, unaire ou comme type.

Les pointeurs seront vus plus en détails au chapitre 7.

# Le type pointeur

```
int main()
{
 int ageAlex = 7, ageSandie = 27;
 double x = 1.2345, y = 32.14;

 int* ptrEntier;
 ptrEntier = &ageAlex;
 *ptrEntier += ageSandie;
 cout << "ageAlex est " << ageAlex << endl;

 double* ptrReel;
 ptrReel = &x;
 y += 5 * (*ptrReel);
 cout << "y contient la valeur " << y << endl;
}
```

Résultat de l'exécution:  
ageAlex est 34  
y contient la valeur 38.3125

II\_pointeur.cpp

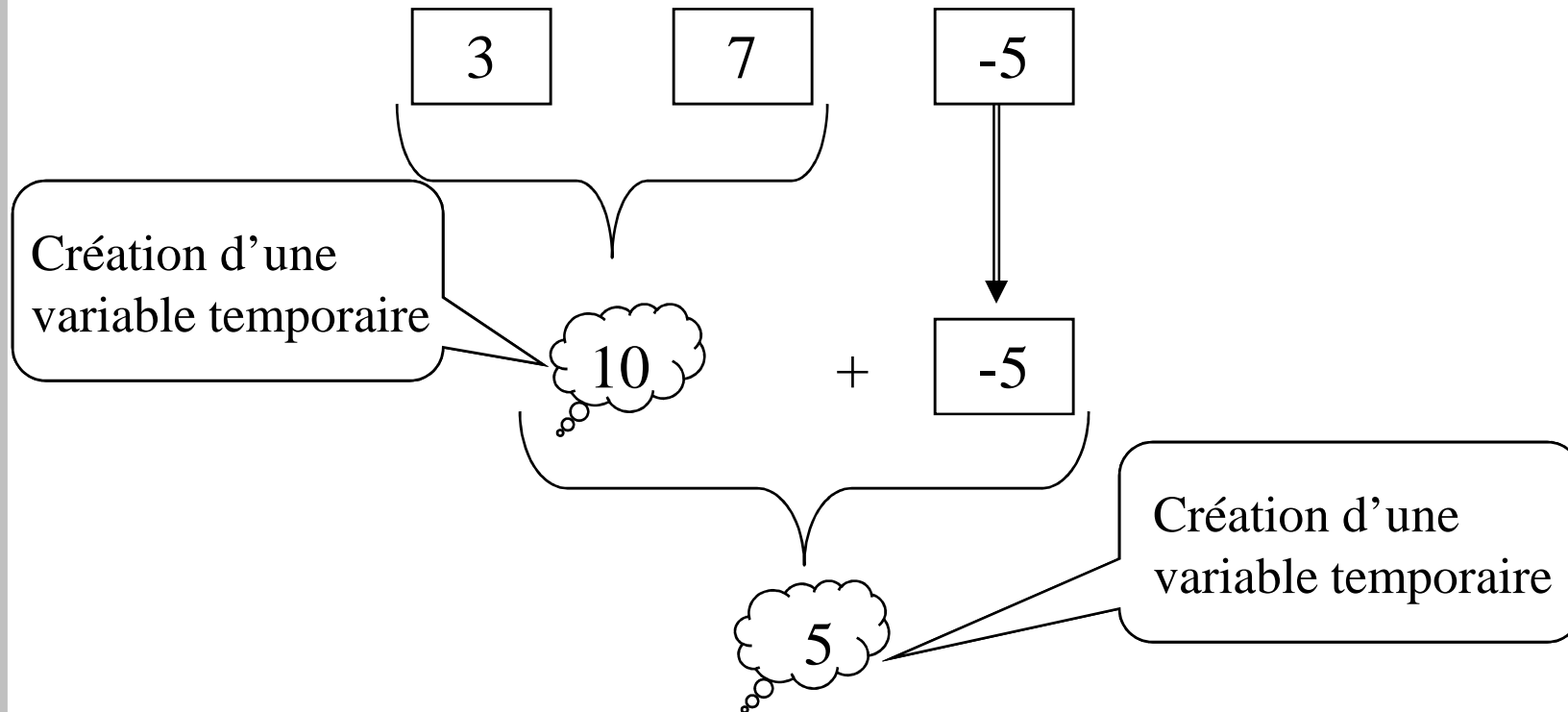
# Opérateurs ++ et --

| Opérateur | Opération                        | Exemple     | Instruction équivalente |
|-----------|----------------------------------|-------------|-------------------------|
| ++        | Ajouter 1 à la variable entière. | ++variable; | variable = variable+1;  |
| --        | Enlever 1 à la variable entière  | --variable; | variable = variable-1;  |

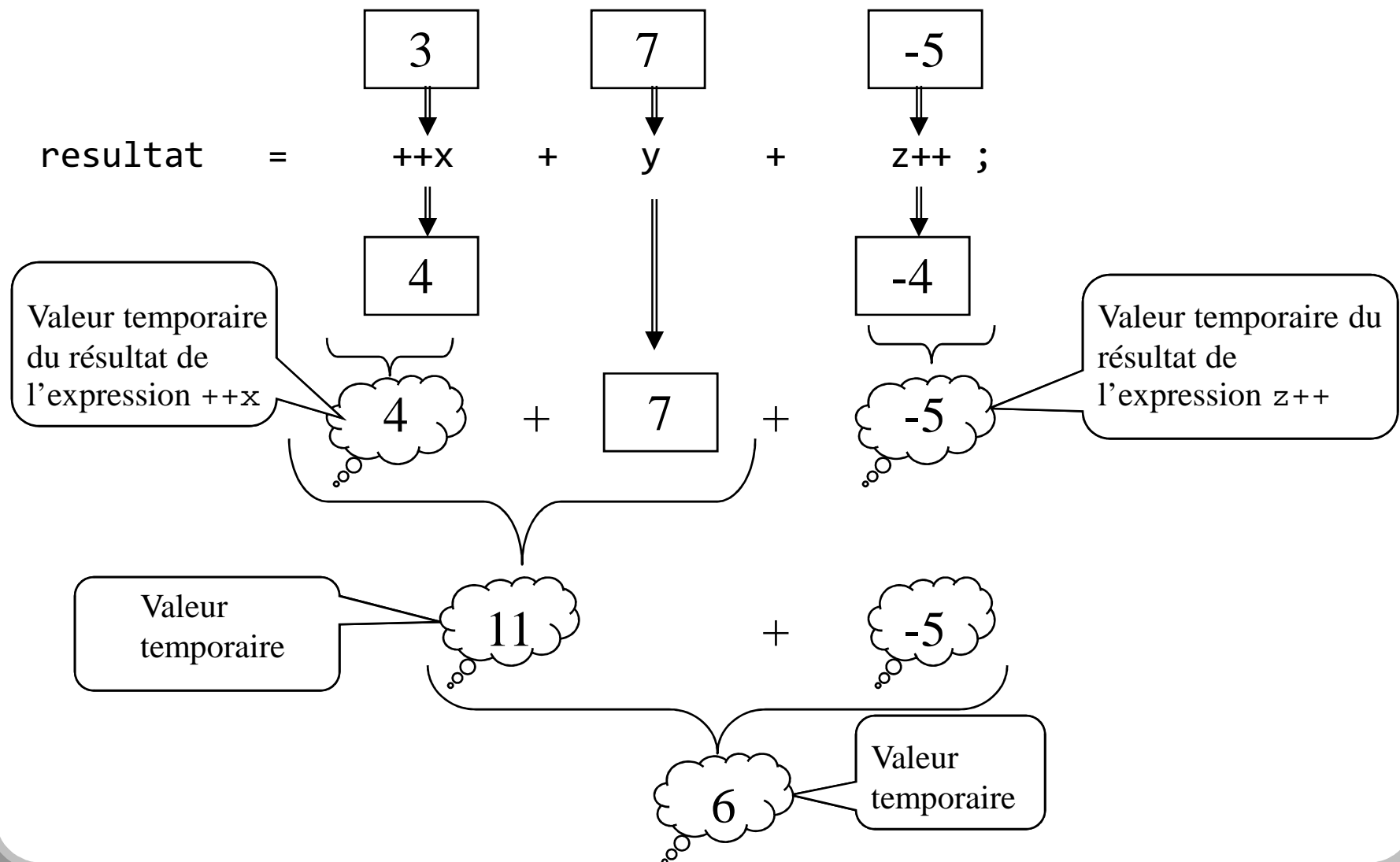


## Exécution de l'évaluation d'une expression

```
resultat = x + y + z;
```



# Exécution de l'évaluation d'une expression



# ++ et -- en position préfixe (avant) ou postfixe (après)

| Instruction                     | Instruction équivalente                                                       |
|---------------------------------|-------------------------------------------------------------------------------|
| <code>++i; ou i++;</code>       | <code>i = i + 1;</code>                                                       |
| <code>--i; ou i--;</code>       | <code>i = i - 1;</code>                                                       |
| <code>i = j++ + 4;</code>       | <code>i = j + 4;</code><br><code>j = j + 1;</code>                            |
| <code>i = ++j - 7;</code>       | <code>j = j + 1;</code><br><code>i = j - 7;</code>                            |
| <code>k = (--j) + (m++);</code> | <code>j = j - 1;</code><br><code>k = j + m;</code><br><code>m = m + 1;</code> |

Attention: ne pas écrire « `i = i++;` » ou « `i = i--;` »

# Priorité des opérateurs et leur associativité

| Priorité | Opérateur                                                                                                                   | Associativité   |
|----------|-----------------------------------------------------------------------------------------------------------------------------|-----------------|
| 1        | [ ] ( ) -> . ++( <i>postfixe</i> )<br>--( <i>postfixe</i> )                                                                 | gauche à droite |
| 2        | ! ~ ++( <i>préfixe</i> ) --( <i>préfixe</i> )<br>-( <i>unaire</i> ) +(unaire) ( <i>type</i> )<br>*(unaire) &(unaire) sizeof | droite à gauche |
| 3        | * / %                                                                                                                       | gauche à droite |
| 4        | + -                                                                                                                         | gauche à droite |
| 5        | << >>                                                                                                                       | gauche à droite |
| 6        | < > <= >=                                                                                                                   | gauche à droite |
| 7        | == !=                                                                                                                       | gauche à droite |
| 8        | &                                                                                                                           | gauche à droite |
| 9        | ^                                                                                                                           | gauche à droite |
| 10       |                                                                                                                             | gauche à droite |
| 11       | &&                                                                                                                          | gauche à droite |
| 12       |                                                                                                                             | gauche à droite |
| 13       | ? : = += -= *= /= %=<br>>>= <<= &= ^=  =                                                                                    | droite à gauche |
| 14       | ,                                                                                                                           | gauche à droite |

## Instruction et expression

⇒ `nbJours = (mois=4)*30+(jour=14);`  
`(4) *30+ (14)`

⇒ `nbre1 = nbre2 = nbr3 = 0;`

`nbre1 = (nbre2 = (nbre3 = 0));`

⇒ `Valeur += 5;`

`Valeur = Valeur + 5;`

⇒ Attention: ordre d'évaluation pas toujours déterminé

`x = !(cout << "a") + !(cout << "b") * !(cout << "c");`

(multiplication avant addition, mais 'a', 'b' et 'c' peuvent être affichés dans n'importe quel ordre)

# Conversion implicite lors d'une opération

(pour les types vus dans ce chapitre)

| + - * / %<br>&   ^ < ><br>== != ... | <i>double</i>                                 | <i>unsigned</i>                                                                                            | <i>int</i>                                                       | <i>char</i>     | <i>bool</i>      |
|-------------------------------------|-----------------------------------------------|------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|-----------------|------------------|
| <i>double</i>                       | 1.0 + 2.0 → 3.0 (symétrique...) <i>double</i> |                                                                                                            |                                                                  |                 |                  |
| <i>unsigned</i>                     | 1U + 2.0 → 3.0                                | 1U + 2U → 3U<br>1U - 2U → <span style="color:red">♦</span> <i>unsigned</i>                                 |                                                                  |                 |                  |
| <i>int</i>                          | 1 + 2.0 → 3.0                                 | 1 + 2U → 3U<br>1 - 2U → <span style="color:red">♦</span><br>-1 < 1U → <span style="color:red">false</span> | 1 + 2 → 3<br>1 / 2 → <span style="color:red">0</span> <i>int</i> |                 |                  |
| <i>char</i>                         | 'A' + 2.0 → 67.0                              | 'A' + 2U → <span style="color:red">67U</span>                                                              | 'A' + 2 → <span style="color:red">67</span>                      | 'C' - 'A' → 2   |                  |
| <i>bool</i>                         | true + 2.0 → 3.0                              | true + 2U → 3U                                                                                             | true + 2 → 3                                                     | true + 'A' → 66 | true + false → 1 |

En rouge ci-dessus sont les résultats qui ne sont probablement pas ce qu'un humain s'attendrait; ♦ = -1U = 4 294 967 295

```

cout << 1 + true << endl; // Affiche: Type:
cout << 1 + 0.5 << endl; // 2 int
cout << 2.2 / 2 << endl; // 1.5 double
cout << 1 / 2.0 << endl; // 1.1 double
cout << -1 * 2 << endl; // 0.5 double
// Ne donne pas ce qu'on pourrait croire :
cout << -1 * 2 << endl; // -2 int
cout << 1 / 2 << endl; // 0 int
cout << 'A' + 2 << endl; // 67 int, noter que 67=='C'
cout << -1 * 2U << endl; // 4294967294 unsigned

```

II\_conversion.cpp

## Conversion explicite

À partir des déclarations, `double valeur;`  
`char car;`

➔ `int('A' + 5.4)` L'addition donne un `double`, qui est ensuite converti en `int`.

➔ `valeur = double(int(car) + 1);`

La variable `car` est convertie en `int`, l'expression `int(car) + 1` est de type `int`, elle est ensuite convertie explicitement en `double`.

Autre façon en C++

➤ `valeur = static_cast<double>(car + 1);`

# Conversion de type

```
// On aimerait :
// 1 / 2 = 0.5 , 'A' + 2 = 'C' , -1 * 2U = -2 .

// Une des syntaxe C++ :
// Bonne manière :
cout << double(1) / 2 << endl; // 0.5 (division de double)
cout << char('A' + 2) << endl; // C (résultat converti en char)
cout << -1 * int(2U) << endl; // -2 (multiplication de int)
// Mauvaise manière :
cout << double(1 / 2) << endl; // 0 (division de int)
cout << 'A' + char(2) << endl; // 67 (résultat est un int)
cout << int(-1 * 2U) << endl; // -2 (bon résultat dans notre cas mais est
// "implementation-defined" selon C++11 §4.7.3)

// Mêmes conversion mais avec la syntaxe C :
cout << (double)1 / 2 << endl; // 0.5
cout << (char)('A' + 2) << endl; // C
cout << -1 * (int)2U << endl; // -2
// Mauvaise manière :
cout << (double)(1 / 2) << endl; // 0
cout << 'A' + (char)2 << endl; // 67
cout << (int)(-1 * 2U) << endl; // -2
```

II\_conversion.cpp