

ICC :

Cours de programmation (C++)

FICHES RÉSUMÉ DE C++

Jamila Sam

Laboratoire d'Intelligence Artificielle
Faculté I&C



Variables



En C++, une **valeur** à conserver est stockée dans une variable caractérisée par :

- ▶ son **type**
- ▶ et son **identificateur** ;

(définis lors de la **déclaration**)

La **valeur** peut être définie une première fois lors de l'**initialisation**, puis éventuellement modifiée par la suite.

Rappels de syntaxe :

<code>type nom ;</code>	(déclaration)
<code>type nom(valeur);</code>	(initialisation)
<code>nom = expression ;</code>	(affectation)

Types élémentaires :

`int`
`double`
`char`
`bool`

Exemples :

```
int val(2) ;  
const double z(x+2.0*y);  
C++11 constexpr double pi(3.141592653);  
i = j + 3;
```



Opérateurs



Opérateurs arithmétiques

*	multiplication
/	division
%	modulo
+	addition
-	soustraction
++	incrément (1 opérande)
--	décrément (1 opérande)

Opérateurs de comparaison

==	teste l'égalité logique
!=	non égalité
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal

Opérateurs logiques

&&	"et" logique
	ou
^	ou exclusif
!	négation (1 opérande)

Priorités (par ordre décroissant, tous les opérateurs d'un même groupe sont de priorité égale) :

! ++ --, * / %, + -, < <= > >=, == !=, ^ &&, ||



Les structures de contrôle



les branchements conditionnels : *si ... alors ...*

```
if (condition)
    instructions
.....
if (condition 1)
    instructions 1
...
else if (condition N)
    instructions N
else
    instructions N+1

switch (expression) {
    case valeur:
        instructions;
        break;
    ...
    default:
        instructions;
}
```

les boucles conditionnelles : *tant que ...*

```
while (condition)
    Instructions

do
    Instructions
while (condition);
```

les itérations : *pour ... allant de ... à ...*

```
for (initialisation ; condition ; increment)
    instructions
```

les sauts : *break;* et *continue;*

Note : *instructions* représente une instruction élémentaire ou un bloc.
instructions; représente une suite d'instructions élémentaires.



Les fonctions



Prototype (à mettre **avant** toute utilisation de la fonction) :

```
type nom ( type1 arg1, ..., typeN argN [ = valN ] );
```

`type` est `void` si la fonction ne retourne aucune valeur.

Définition :

```
type nom ( type1 arg1, ..., typeN argN )
{
    corps
    return value;
}
```

Passage par **valeur** :

```
type f(type2 arg);
```

`arg` ne peut pas être modifié par `f`

Passage par **référence** :

```
type f(type2& arg);
```

`arg` peut **être modifié** par `f`

Surcharge (exemple) :

```
void affiche (int arg);
void affiche (double arg);
void affiche (int arg1, int arg2);
```



Le template vector



```
#include <vector>
```

Déclaration : `vector< type > identificateur;`

Déclaration/Initialisation :

```
vector< type > identificateur(taille);
```

Accès au (i+1)-ème élément : `tab[i];`

Fonctions spécifiques :

`int tab.size()` : renvoie la taille

`bool tab.empty()` : détermine s'il est vide ou non

`void tab.clear()` : supprime tous les éléments

`void tab.pop_back()` : supprime le dernier élément

`void tab.push_back(valeur)` : ajoute un nouvel élément à la fin



Les tableaux de taille fixe



```
#include <array>
```

Déclaration : `array<type, taille> identificateur;`

Déclaration/Initialisation :

```
array<type, taille> identificateur =  
    {val1, ... , valtaille};
```

Accès aux éléments : `tab[i]` *i* entre **0** et **taille-1**

Fonctions spécifiques :

`size_t tab.size()` : renvoie la taille

Tableau multidimensionnel :

```
array<array<type, nb_colonnes>, nb_lignes>  
identificateur;  
tab[i][j] = ...;
```



Les chaînes de caractères



`#include <string>`

déclaration/initiaisation : `string identificateur("valeur");`

Affectation : `chaine1 = chaine2;`
`chaine1 = "valeur";`
`chaine1 = 'c';`

Concaténation : `chaine1 = chaine2 + chaine3;`
`chaine1 = chaine2 + "valeur";`
`chaine1 = chaine2 + 'c';`

Accès au (i+1)-ème caractère : `chaine[i];`

Fonctions spécifiques :

taille : `chaine.size()`
insertion : `chaine.insert(position, chaine2)`
remplacement : `chaine.replace(position, longueur, chaine2)`
suppression : `chaine.replace(position, longueur, "")`
sous-chaîne : `chaine.substr(position, longueur)`
recherche : `chaine.find(souschaine)`
`chaine.rfind(souschaine)`

valeur "pas trouvé" d'une recherche : `string::npos`



Les structures



Déclaration du type correspondant :

```
struct Nom_du_type {  
    type1 champ1 ;  
    type2 champ2 ;  
    ...  
};
```

Déclaration d'une variable :

```
Nom_du_type identificateur;
```

Déclaration/Initialisation d'une variable :

```
Nom_du_type identificateur = { val1, val2, ... };
```

Accès à un champ donné de la structure :

```
identificateur.champ
```

Affectation globale de structures :

```
identificateur1 = identificateur2
```



Pointeurs & références



Déclaration : `type* pointeur;`

Déclaration/Initialisation :

```
type* pointeur(adresse);  
unique_ptr<type>(new type(valeur));  
type& reference(objet);
```

Adresse d'une variable : `&variable`

Accès au contenu pointé par un pointeur : `*pointeur`

Allocation mémoire :

```
pointeur = new type;  
pointeur = new type(valeur);
```

Libération de la zone mémoire allouée :

`delete pointeur` (pour les « pointeurs classiques », obligatoire)



Les entrées/sorties



Clavier / Terminal : `cin / cout` et `cerr`

Fichier de définitions : `#include <iostream>`

Utilisation :

écriture : `cout << expr1 << expr2 << ... ;`

lecture : `cin >> var1 >> var2 >> ... ;`

Saut à la ligne : `endl`

Lecture d'une ligne entière : `getline(cin, string);`

.....

Formatage :

Manipulateurs	
<code>#include <iomanip></code> <code>cout << manip << expr << ...</code>	
<code>dec, oct, hex</code> <code>setprecision(int)</code>	changement de base nombre de chiffres à afficher
<code>setw(int)</code>	largeur de colonne
<code>setfill(char)</code>	nombre de caractères à lire caractère utilisé dans l'alignement
<code>cin >> ws</code>	saute les blancs

Options	
<code>setf(ios::option)</code> <code>unsetf(ios::option)</code>	
<code>ios::left</code>	alignement à gauche
<code>ios::showbase</code>	afficher la base
<code>ios::showpoint</code>	afficher toujours la virgule
<code>ios::fixed</code>	notation fixe
<code>ios::scientific</code>	notation scientifique



Les entrées/sorties (2)



Fichiers :

Fichier de définitions : `#include <fstream>`

Flot d'**entrée** (similaire à `cin`) : `ifstream`

Flot de **sortie** (similaire à `cout`) : `ofstream`

Création : `type_flot nom_de_flot;`

Lien (ouverture) : `flot.open("fichier");`

ouverture en binaire :

pour lecture : `ifstream flot("fichier", ios::in|ios::binary);`

pour écriture : `ofstream flot("fichier", ios::out|ios::binary);`

Utilisation : comme `cin` et `cout` :

`flot << expression << ... ;`

`flot >> variable_lue >> ...;`

Test de fin de fichier : `flot.eof()`

Fermeture du fichier : `flot.close()`

Test d'échec de lecture/écriture/ouverture sur le flot : `flot.fail()`