Le Langage C++

Cours d'Algorithmique 1er Semestre (Frédéric Koriche)

IUT Informatique de Lens

UNIVERSITÉ D'ARTOIS

Types Simples

Type	Octets	Domaine
`bool`	1	0 et 1
`char`	1	Codes ASCII
`int`	4	de -2^{31} à $2^{31}-1$
`long long`	8	de -2^{63} à $2^{63}-1$
`float`	4	$\pm 3.4 10^{\pm 38}$ (7 décimales)
`double`	8	$\pm 1.7 10^{\pm 308}$ (15 décimales)

Les types non signés (nombres positifs) peuvent être construits en utilisant le mot clé `unsigned` devant le type de données (ex: `unsigned int`).

Opérateurs sur les types simples

- Arithmétiques (entiers): toutes les entrées sont des entiers et la sortie est un entier.

Nom	Opérateur C++
addition	$+$
soustraction	$-$
multiplication	$*$
division entière	$/$
reste	%
inversion de signe	$-$

- Arithmétiques (décimaux): au moins une entrée est un décimal et la sortie est un décimal.

Nom	Opérateur C++
addition	$+$
soustraction	$-$
multiplication	$*$
division	$/$
inversion de signe	$-$

- Comparaisons: les deux entrées sont du même typer et la sortie est un booléen.

Nom	Opérateur C++
est égal à	$==$
est différent de	$!=$
est plus petit que	$<$
est plus grand que	$>$
est plus petit ou égal à	$<=$
est plus grand ou égal à	$>=$

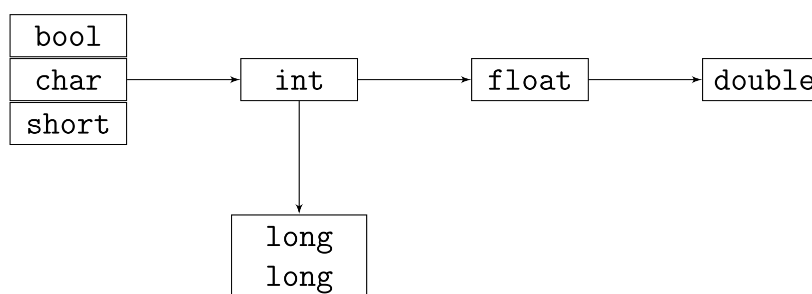
- Logiques: les entrées sont des booléens et la sortie est un booléen.

Nom	Opérateur C++
conjonction	&&
disjonction	||
négation	!

Conversions de types

Une *conversion de type* est une transformation d'un type en un autre.

- Conversion implicite: le compilateur réalise automatiquement de la conversion en utilisant le graphe suivant:



- Conversion explicite: le programmeur doit intervenir pour réaliser la conversion en utilisant l'opérateur de transtypage `()`.

Exemple: soit *i* un `int` et *x* un `float`

- `x = i` est une conversion implicite.
- `i = (int)x` est une conversion explicite.

Expressions

Une expression est une *composition* d'opérations dont l'ordre est spécifié par la priorité des opérateurs et le parenthésage. Le type d'une expression est donné par le type de sa valeur de sortie. L'ordre de priorité décroissante des opérateurs est donné par le tableau suivant.

Accès aux éléments d'un tableau	`[]`
Accès aux composants d'une structure	`.`
Incrémentation et décrémentation	`++,--`
Opérateur d'adresse	`&`
Inversion de signe	`-`
Négation logique	`!`
Multiplication, division, mod	`*,/,%`
Addition, soustraction	`+,-`
Décalage	`<<,>>`
Comparaisons	`<=,<,=,!=,>,>=`
ET logique	`&&`
OU logique	`||`
Affectations	`=, +=, -=, *=, /=, %=`

Instructions

Une instruction est expression terminée par un point-virgule. Les quatre instructions de base sont la déclaration (mémoire), l'assignation (calcul), la lecture (entrées) et l'écriture (sorties).

Instruction	Spécification
Déclaration	`type variable;`
Assignation	`variable = expression;`
Lecture	`cin >> variable;`
Ecriture	`cout << expression;`

Le langage C++ autorise de nombreuses abréviations pour les assignations. Par exemple:

Instruction	Opération	Abréviation
Incrémentation	`i = i + 1;`	`i++;`
Décrémentation	`i = i - 1;`	`i--;`
Assignation avec addition	`x = x + 2;`	`x += 2;`
Assignation avec modulo	`x = x % 2;`	`x %= 2;`

Blocs

Un bloc est une séquence d'instructions identifiée par une accolade ouvrante et une accolade fermante.

L'instruction de test "if else"

Dans **if**(*condition*) {*bloc 1*} **else** {*bloc 2*}, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple: racine carrée

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

int main()
{
float x,y;
cin >> x;
if(x >= 0)
     {
     y = sqrt(x);
     cout << y;
     }
else
     cout << "valeur indefinie";
return 0;
}
```

L'instruction de test "switch case"

Dans l'instruction **switch**(*condition*) **case** v_1 : *bloc 1* **case** v_2 : *bloc 2* ..., la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté jusqu'à rencontrer un **break**.

Exemple: choix de menu

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

int main()
{
int menu;
cin >> menu;
switch(menu)
     {
     case 1:  cout << "enfants"; break;
     case 2:  cout << "végétarien"; break;
     default:  cout << "standard"; break;
     }
}
```

L'instruction de boucle "for"

Cette instruction ne doit être utilisée que lorsque le nombre d'itérations est connu à l'avance. Dans l'instruction **for**(*initialisation*; *maintenance*; *modification*) {*bloc*}, la première instruction initialise un compteur, la seconde teste la condition d'itération et la troisième modifie le compteur.

Exemple: Somme des entiers de 1 à *n*

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

int main()
{
int n, s = 0;
cin >> n;
for(int i = 1; i <= n; i++)
     s += i;
cout << s;
}
```

Les instructions de boucle "while" et "do while"

Ces instructions ne doivent être utilisées que lorsque le nombre d'itérations est a priori inconnu.

- Dans la boucle **while**(*condition*) {*bloc*}, le bloc d'instructions est exécuté tant que la condition reste vraie: il peut être exécuté zero fois.
- Dans la boucle **do** {*bloc*} **while**(condition), le bloc d'instructions est exécuté tant que la condition reste vraie: il est exécuté au moins une fois.

Exemple: Somme des entrées saisies par l'utilisateur

```
int main()
{
int n = 1, s = 0;
while(n != 0)
     {
     cin >> n;
     s += n;
     }
cout << s;
}
```

```
int main()
{
int n, s = 0;
do
     {
     cin >> n;
     s += n;
     }
while(n != 0);
cout << s;
}
```

Tableaux Statiques Unidimensionnels

En C++, les tableaux sont stockés sur des zones contiguës de mémoire. Le premier index d'un tableau de N éléments est 0 et le dernier index est $N-1$. Les seules opérations possibles sont la déclaration, l'initialisation (déclaration avec valeurs initiales) et l'accès à ses éléments.

Opération	Spécification	Exemple
Déclaration	`type nom[taille];`	`float tab[10];`
Initialisation	`type nom[n]` $= \{v_1, \cdots, v_n\}$;	`char voyelles[5] = {'a','e','i','o','u','y'};`
Accès	`nom[index]`	`voyelles[i]`

Exemple: Calculer la moyenne des notes

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

int main()
{
int notes[3] = {10,16,12};
int i, s = 0;
for(i=0; i<3; i++)
      s += notes[i];
cout << s/3.0;
}
```

Tableaux Statiques Multidimensionnels

Un *tableau statique de dimension d* est une séquence de tableaux de dimension $d-1$. En particulier, une *matrice* est un tableau de dimension 2. Comme pour tous les tableaux statiques, les seules opérations possibles sont la déclaration, l'initialisation et l'accès à ses éléments.

Op.	Spécification	Exemple
Décl.	`type nom[rangees][colonnes];`	`float matrice[4][4];`
Init.	`type nom[m][n]` $= \{\{v_{11}, \cdots, v_{1n}\}, \cdots, \{v_{m1}, \cdots, v_{mn}\}\}$;	`float unité[2][2] = {{1,0},{0,1}};`
Accès	`nom[rangee][colonne]`	`matrice[i][j]`

Tableaux Dynamiques

En C++, les tableaux dynamiques sont implémentés à partir de la classe `vector` de la bibliothèque STL. En plus des opérations de tableaux statiques, les vecteurs permettent des opérations de copie et de modification.

Opération	Spécification	Exemple
Déclaration	`vector<type> nom;`	`vector<int> notes;`
Initialisation	`vector<type> nom(quantité,valeur);`	`vector<int> nom(10,0);`
Copie	`nom1 = nom2;`	`v = w;`
Accès aux éléments	`nom[index]`	`v[i]`
Accès à la taille	`nom.size()`	`v.size()`
Test du vecteur vide	`nom.empty()`	`if(v.empty()){...}`
Ajout d'un élément à la fin	`nom.push_back(valeur);`	`v.push_back(12);`

Chaînes

En C++, les chaînes sont implémentées à partir de la classe `string` de la bibliothèque STL. En plus des opérations de vecteurs, les chaînes permettent des opérations de comparaison lexicographique.

Opération	Spécification	Exemple
Déclaration	`string nom;`	`string s;`
Initialisation	`string nom = "...";`	`string s = "Bonjour";`
Copie	`nom1 = nom2;`	`s = t;`
Accès aux éléments	`nom[index]`	`s[i]`
Accès à la taille	`nom.size()`	`s.size()`
Test de la chaîne vide	`nom.empty()`	`if(s.empty()){...}`
Concaténation	`+`	`s += " tout le monde !";`
Comparaisons	`<=,<,=,!=,>,>`	`if(s == t){...}`

Typage

En C++, il est possible de construire de nouveaux types à partir de types prédéfinis en utilisant le mot-clé `typedef`. Les types doivent apparaître avant la déclaration des fonctions.

Exemple: déclaration d'un type de matrice

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

typedef float RotationMatrix[2][2];

int main()
{
RotationMatrix R;
}
```

Enumérations

En C++, les types énumérés sont définis à partir du mot-clé `enum`.

Opération	Spécification	Exemple
Décl. type	`enum Nom` $\{v_1, \cdots, v_n\}$;	`enum Couleurs {rouge,vert,bleu};`
Décl. variable	`Nom variable;`	`Couleurs c;`
Copie	`variable_enum = donnee_enum;`	`c = bleu;`
Conversion	`variable_enum = (Nom) donnee_int;`	`c = (Couleurs) 2;`
Comparaisons	`<=,<,=,!=,>,>`	`if(c == bleu){...}`

Structures

En C++, les types structurés sont définis à partir du mot-clé `struct`.

Opération	Spécification	Exemple
Décl. de type	`struct Nom` $\{Type_1\ nom_1; \cdots; Type_k\ nom_k;\}$;	`struct Point {float x; float y;};`
Décl. de variable	`Nom variable;`	`Point p;`
Copie	$variable_1 = variable_2$;	`p = q;`
Accès	`.`	`p.x`

Exemple: Déclarer un tableau de structures

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

struct Point {float x; float y;};
typedef Point Figure [100];

int main()
{
Figure f;
cout << f[0].x;
}
```

Déclaration de Fonctions

La *déclaration* d'une fonction consiste à spécifier son en-tête: le nom de la fonction, le type de ses variables d'entrée, et le type de sa valeur de sortie (`void` si aucune sortie). Les variables d'entrée peuvent être transmises de deux manières:

- par valeur: une copie de la variable est réalisée, et seulement cette copie est transmise à la fonction,
- par adresse: la variable est directement transmise à la fonction et peut donc être modifiée.

Afin d'éviter qu'une variable transmise par adresse soit modifiée par la fonction, il est possible de la fixer comme constante en utilisant le mot-clé `const`.

Transmission	Exemple
par valeur	`float somme(float x, float y);`
par adresse	`void permuter(float& x, float& y);`
par adresse fixe	`float norme(const Point& p);`

Définition de Fonctions

La *définition* d'une fonction consiste à spécifier à la fois son en-tête et son corps, c'est à dire la séquence d'instruction permettant d'obtenir le résultat désiré. Le mot-clé `return` est utilisé pour retourner une valeur de sortie.

Exemples

```
long long factorielle(int n)
{
long long f = 1;
for(int i = n; i > 0; i--)
      f *= i;
return f;
}
```

```
void permuter(int& a, int& b)
{
int temp = a;
a = b;
b = temp;
}
```

Flux

Un *flux* est une séquence de caractères qui évolue au cours du temps. L'état d'un flux indique si l'opération d'entrée ou de sortie appliquée sur le flux a fonctionné, ou non. En C++, l'état d'un flux `monFichier` est accessible par 4 fonctions

- `monFichier.good()`: vrai si on peut accéder (lire/écrire) au caractère courant
- `monFichier.eof()`: vrai si le flux vient d'atteindre la fin de fichier
- `monFichier.fail()`: vrai si le flux n'arrive pas à accéder au caractère courant
- `monFichier.bad()`: vrai si le flux est endommagé

Les caractères d'un flux peuvent être lus selon un certain type de format (préfixé par `ios::`).

Champ	Format	Désignation
Général	`boolalpha`	Formatte 1 et 0 en `true` et `false`
	`showbase`	Insère la base du caractère
	`showpoint`	Insère '.' pour les décimales
	`showpos`	Insère '+' pour les nombres positifs
	`uppercase`	Met tous les caractères en majuscules
Base	`dec`	Lit/écrit les caractères en base décimale
	`oct`	Lit/écrit les caractères en base octale
	`hex`	Lit/écrit les caractères en base hexadécimale
Flottants	`fixed`	Lit/écrit les flottants en notation fixe
	`scientific`	Lit/écrit les flottants en notation scientifique
Ajustement	`left`	Ajuste les caractères à gauche
	`right`	Ajuste les caractères à droite

Les *fichiers* sont des flux entrants (lecture) ou sortants (écriture) associés à la mémoire persistante. En C++, ils sont implémentés par la classe `fstream` de la STL.

Lecture et écriture de fichiers

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

int main()
{
ifstream monFichier;
string phrase;
monFichier.open("nomEntree.txt");
while(monFichier.good())
      {
      getline(monFichier,phrase);
      cout << phrase;
      }
monFichier.close();
return 0;
}
```

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

int main()
{
ofstream monFichier;
double f;
monFichier.open("nomSortie.txt");
monFichier.flags(ios::scientific);
monFichier.precision(5);
for(int i = 0; i < 100; i++)
      {
      cin >> f;
      monFichier << f << endl;
      }
monFichier.close();
return 0;
}
```