R101-amphi06

Alain Casali

Aix Marseille Univ

Identificateur

`char`

Valeurs

N'importe quel caractère ASCII entre '

ACSII : American Standard Code for Information Interchange

Opérations de comparaison

Produisent un booléen

< <= > >=

Opérations d'identité

Produisent un booléen

== !=

Élément suivant

Produit un `char`
La fonctions `succ ()` n'existe pas en C++.

Pour passer au caractère suivant, on applique l'opérateur `++` à gauche de la variable.

Exemple

```
char c = 'a';
char next = ++c;
cout << next;
```

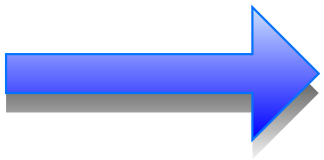
b

```
char next = ++'a';
cout << next;
```

Erreur de compilation

Élément précédant

Produit un `char`
La fonctions `pred ()` n'existe pas en C++.



Pour passer au caractère suivant, on applique l'opérateur `--` à gauche de la variable.

Exemple

```
char c = 'b';
char prev = --c;
cout << prev;                    a
```

Fonctions applicables

Produisent un `char`

Pré-requis pour utiliser les fonctions sur les caractères :

```
#include <cctype>
using namesapce std;
```

- `tolower (aCharacter);`

  renvoie le caractère en minuscule si c'est une majuscule, sinon renvoie le caractère.

- `toupper (aCharacter);`

  renvoie le caractère en majuscule si c'est une minuscule, sinon renvoie le caractère.

Exemple

```
cout << toupper (next);        B
cout << tolower ('C');         c
```

Prédicats applicables

Produisent un `bool`

Pré-requis pour utiliser les prédicats sur les caractères :

```
#include <cctype>
using namesapce std;
```

- `islower (aCharacter)`

  renvoie vrai si le caractère est une minuscule

- `isalpha (aCharacter)`

  renvoie vrai si le caractère est alphanumérique

D'autres prédicats sont disponibles sur : `http://www.cplusplus.com/reference/cctype/?kw=cctype`

Exemple

```
if (islower (next))
{
    cout << "caractere minuscule";
}
```

Caractères spéciaux

Liste (non exhaustive) des caractères spéciaux :

Caractère	Signification
`'\t'`	tabulation
`'\n'`	Line Feed (saut de ligne)
`'\r'`	Carriage Return (retour chariot)
`'\' '`	caractère '

Dans les systèmes compatibles Unix, le caractère matérialisant un retour à la ligne dans un fichier est `'\n'`, alors que dans les systèmes compatibles Windows c'est la combinaison des deux caractères `'\r' '\n'`

Encodage des caractères

Le type `char` est codé sur 1 octet $\longrightarrow 2^8$ (256) valeurs possibles.
Le type `wchar_t` est codé sur 4 octets $\longrightarrow 2^{32}$ valeurs possibles.
Encodages populaires en France :

- Unicode (compatible avec UTF-8, UTF-16 et UTF-32) ;
- ISO 8859-15.

NB : UTF signifie Universal Transformation Format

Identificateur

`string`

Valeurs

N'importe quelle suite de caractères ASCII entre double quotes "

Opérations de comparaison

Produisent un booléen

< <= > >=

$\Longrightarrow$ il existe une relation d'ordre (ordre lexicographique)

Opérations d'identité

Produisent un booléen

== !=

Taille d'une string

Produit un `size_t` ⇔ `unsigned long long`
Pour connaître la taille d'une `string`, on appelle la méthode `size ()` sur l'objet (la variable) de type `string`.

Exemple

```
string str;
cin >> str;                    Mario
cout << str.size();            5
```

[C++17] Pour connaître la taille d'une `sting`, on appelle la `fonction size ()`. Cette fonction prend en paramètre l'objet (la variable) de type `string`.

Exemple

```
cout << size (str);   // 5
```

Accès à un élément

Produit un `char`
On accède au $(i+1)^{\text{ème}}$ élément d'une `string` en utilisant `[i]`.

- La première case de la string a pour indice 0 ;
- La dernière case de la string a pour indice `varIdent.size () - 1` ;
- Aucun contrôle quant à la validité de l'indice n'est effectué lors d'un accès à une case, que ce soit en lecture ou en écriture.

Exemple

Afficher le contenu d'une `string` en allant à la ligne après chaque caractère

```
pour (i variant_de 0 a str.size() - 1)
{
    cout << str [i] << endl;
}
```

Opérateur de concaténation +

Produit un `string`

L'opérateur de concaténation + est un opérateur binaire. Pour qu'il puisse produire une `string`, on doit avoir :

- À gauche un littéral ou une variable (lhs) ;
- À droite un littéral ou une variable (rhs).

lhs	rhs	Compatibilité
`char`	`string`	C++11
`string`	`char`	C++11
`string`	`string`	C++

Exemple

```
str = "0123456789";
cout << str.size ();          10
str = str + "aeiouy\"";
cout << str.size ();          17
```

Boucle while

Modèle général algorithmique :

```
tant_que (condition)
faire
    ...
ffaire
```

Modèle général C++ :

```
while (condition)
{
    ...
}
```

Pas de ';' après l'instruction while (condition).

Exemple

Algo :

```
declarer i : entier_naturel;
i <- 0;
tant_que (i < taille (str))
faire
    afficher (str[i]);
    ligne_suivante;
    i <- i + 1;
ffaire
```

C++ :

```
unsigned long long i; //size_t i
i = 0;
while (i < str.size())
{
    cout << str [i]
         << endl;
    i = i + 1;
}
```

Modèle général algorithmique :

```
repeter

    ...

tant_que (condition)
```

Modèle général C++ :

```
do
{
    ...
}
while (condition);
```

Le ';' est obligatoire après l'instruction `while (condition)`.

Exemple

```
unsigned long i;
i = 0;
do
{
    cout << str[i] << endl;
    i = i + 1;
}
while (i < str.size ());
//while (i != str.size ());
```

Le code est faux si `str` est une `string` de taille nulle !

Boucle for

Modèle général algorithmique :

```
pour (nomVar variant_de borneMin a borneMax)
faire
    ...
ffaire
```

Cette boucle n'existe pas telle qu'elle en C++, mais la boucle `for` existe.

Profil de la boucle for

Modèle général C++ :

```
for ([expression1]; [expression2]; [expression3])
{
    instruction(s);
}
```

- `instruction(s)` : un ensemble d'instructions
- `expression1` : une instruction unique, exécutée avant l'entrée dans l'itération ; c'est l'étape d'initialisation des variables ;
- `expression2` : une instruction unique et booléenne représentant la condition de continuation ;
- `expression3` : une instruction unique, exécutée avant la prochaine itération.

Danger : éviter de mettre le caractère ';' après l'instruction `for ( ; ;)`.

Exemple

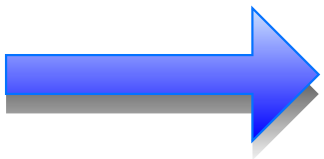
```
for (size_t i = 0; i < str.size (); i = i + 1)
{
    cout << str[i] << endl;
}
```

Équivalent à :

```
for (size_t i = 0; i < str.size (); )
{
    cout << str[i] << endl;
    i = i + 1;
}
```

Équivalent à : ?

```
size_t i = 0;
for (; i < str.size (); i = i + 1)
{
    cout << str[i] << endl;
}
```



Non : portée de variable

Exemple

```
for (size_t i = 0; i < str.size (); i = i + 1)
{
    cout << str[i] << endl;
}
```

Équivalent à : ?

```
for (int i = 0; i < str.size (); i = i +1)
{
    cout << str[i] << endl;
}
```

Non : si votre tableau est de taille >= *max (int) + 1*, vous ne pouvez pas accèder à ces éléments

Effet de bord

```
int i (0);
...
i = -1;
for ( ; i < str.size (); i = i + 1)
{
    cout << str[i] << endl;
}
```

Accès à la case d'indice -1.

Boucle infinie

Modèle général algorithmique :

```
boucle
    instruction(s);
fboucle
```

Modèle général C++ :

```
for ( ; ; )
{
    instruction(s);
}
```

```
while (true)
{
    instruction(s);
}
```

Pas de condition de sortie explicite.

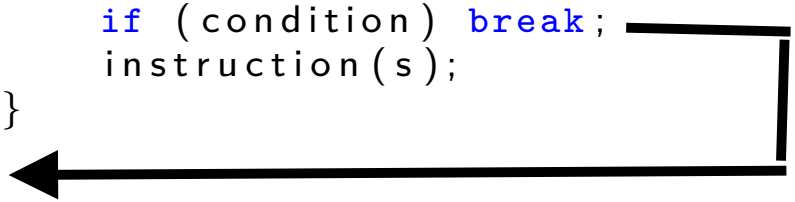
Rupture de schéma répétitif : instruction `break`

Modèle général algorithmique :

```
boucle
    instruction (s);
    si (condition) sortie;
    instruction (s);
fboucle
```

Modèle général C++ :

```
for([e1]; [e2]; [e3])
{
    instruction(s);
    if (condition) break;
    instruction(s);
}
```



On sort de la boucle dès que `condition` est vraie.

Exemple 1 : générer un entier naturel > 10

Exemple

```
declarer N : entier_naturel;
boucle
    afficher ("Entrer une valeur > 10 : ");
    saisir (N);
    si (N > 10) sortie;
    afficher (">10 svp!");
fboucle
```

Exemple

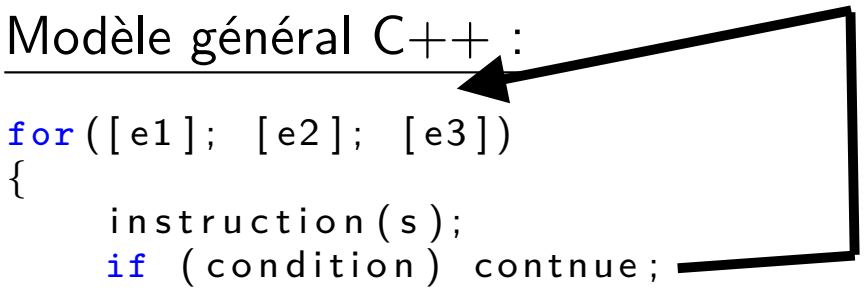
```
unsigned N;
while (true)
{
    cout << "Entrer une valeur > 10 : ";
    cin >> N;
    if (N > 10) break;
    cout << ">10 svp!";
}
```

Rupture de schéma répétitif : instruction `continue`

Modèle général algorithmique :

```
boucle
    instruction (s);
    si (condition) continue;
    instruction (s);
fboucle
```

Modèle général C++ :



```
for([e1]; [e2]; [e3])
{
    instruction(s);
    if (condition) contnue;
    instruction(s);
}
```

On exécute e3 (si présent) puis e2 dès que `condition` est vraie.

Exemple : afficher une lettre sur deux d'un mot saisi

Exemple

```
string str;
cin >> str;
for (size_t i = 0; i < str.size (); i = i + 1)
{
    if (0 == i%2)
    {
        cout << str [i];
    }
}
```

2 blocs ⇒ 2 indentations
⇒ 2 fois moins lisible

Exemple

```
string str;
cin >> str;
for (size_t i = 0; i < str.size (); i = i + 1)
{
    if (0 != i%2) continue;
    cout << str [i];
}
```

1 bloc ⇒ 1 indentation
⇒ plus lisible

Combinaison de break et de continue

Exemple

```
while (ICanFrag ())
{
    if (kill ()) continue;
    if (someoneNearMe ()) break;
}
findAnotherSpot ();
```

1. Le type caractère
2. Le type `string`
3. Boucle `while`
4. Boucle `for`
5. Boucle infinie
6. Rupture de schéma répétitif
7. Schéma séquentiel `switch / case`
   - Absence d'instruction `break` dans un `switch/ case`
   - Instruction `break` et boucle
8. Les sous-programmes
9. Cosmétique
10. Le type `vector` (tableau de taille variable)
11. Boucle `for` pour les collections

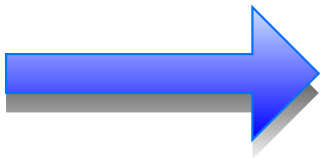
Schéma séquentiel switch / case

Modèle général algorithmique :

```
choix_sur Variable_de_choix
entre
  cas Ens_1 :
    Sequ_1;
  cas Ens_2 :
    Sequ_2;
  cas Ens_3 :
    Sequ_3;
  autre :
    Sequ_4;
fchoix
```

Modèle général C++ :

```
switch (varIdent)
{
case literal1:
     instruction1;
     break;
case literal2:
     instruction2;
     break;
...
default:
     instruction4;
}
```



L'instruction `break` permet la sortie du bloc courant (i.e. la sortie du `switch` / `case`)

- `varIdent` doit être de type entier ou caractère ;
- Pas de `break` après `instruction4` ;

Exemple : gérer le déplacement (touches uniques)

Exemple

```
char c;
cin >> c;
switch (c)
{
case 'a':
     mouveLeftward ();
     break;
case 'z':
     mouveUpward ();
     break;
case 'e':
     mouveRightward ();
     break;
case 's':
     mouveDownward ();
//   break;
}
```

Remarque : pas de {} pour les blocs case.

Exemple : gérer le déplacement (touches multiples - 1)

Exemple

```
char c;
cin >> c;
switch (c)
{
case 'A':
case 'a':
    mouveLeftward ();
    break;
case 'Z':
case 'z':
    mouveUpward ();
    break;
case 'E':
case 'e':
    mouveRightward ();
    break;
case 'S':
case 's':
    mouveDownward ();
}
```

Exemple : gérer le déplacement (touches multiples - 2)

Exemple

```
char c;
cin >> c;
switch (tolower (c))
{
case 'a':
     mouveLeftward ();
     break;
case 'z':
     mouveUpward ();
     break;
case 'e':
     mouveRightward ();
     break;
case 's':
     mouveDownward ();
}
```

Exemple : gérer le déplacement (touches multiples - 2)

Exemple

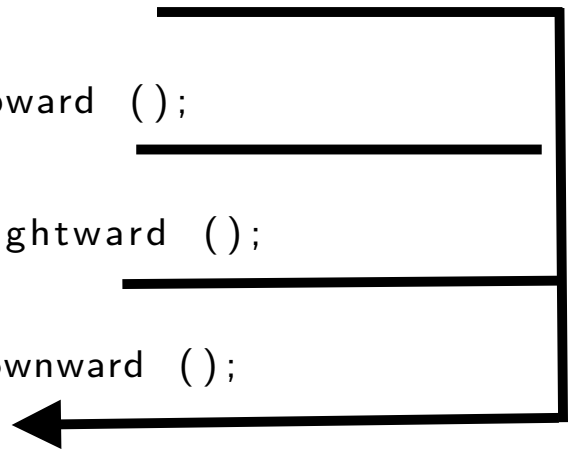
```
char c;
cin >> c;                          a
switch (tolower (c))
{
case 'a':
     mouveLeftward ();             Deplacement gauche
//   break;
case 'z':
     mouveUpward ();               Deplacement haut
     break;
case 'e':
     mouveRightward ();
     break;
case 's':
     mouveDownward ();
}
```

Puisqu'il n'y a pas d'instruction `break`, le `case` est ignoré.

Instruction break et boucle (1)

Exemple

```
char c;
cin >> c;
for ( ; 'q' != tolower (c); cin >> c)
{
    switch (tolower (c))
    {
    case 'a':
        mouveLeftward ();
        break;
    case 'z':
        mouveUpward ();
        break;
    case 'e':
        mouveRightward ();
        break;
    case 's':
        mouveDownward ();
    }
...
}
```

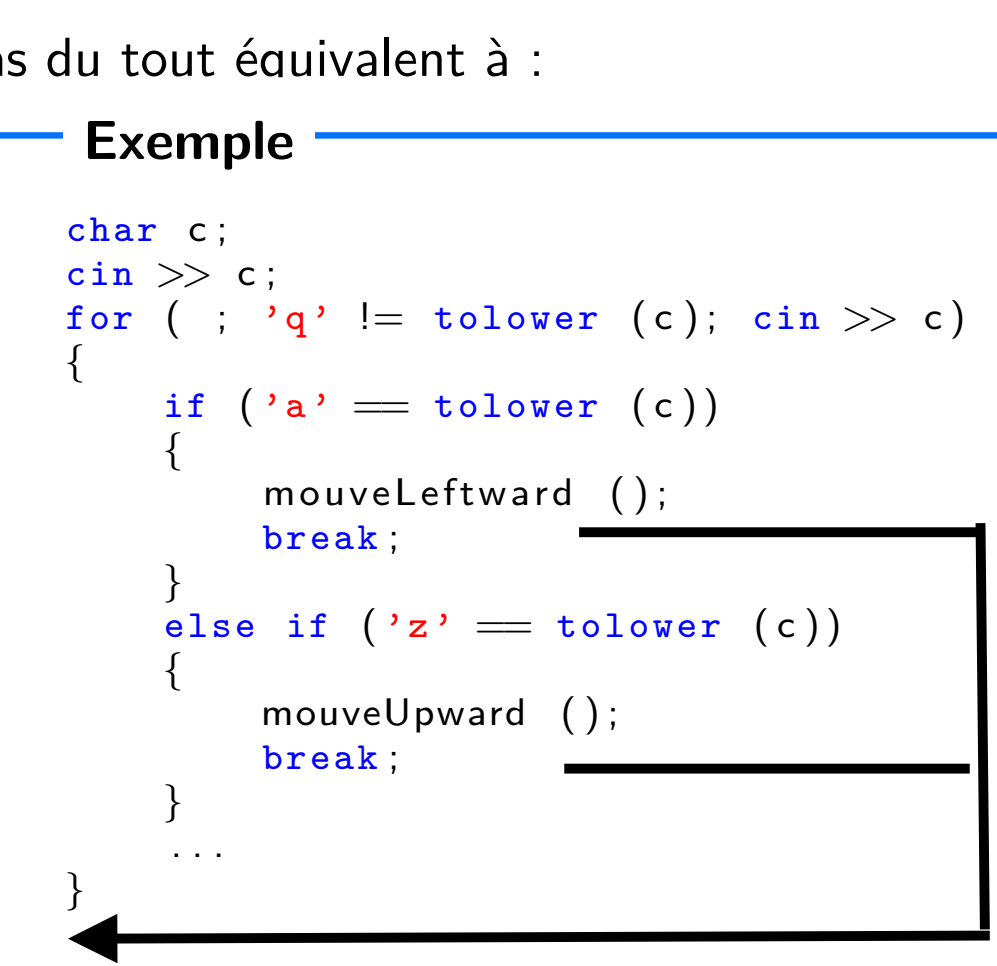


Instruction break et boucle (2)

Pas du tout équivalent à :

Exemple

```
char c;
cin >> c;
for ( ; 'q' != tolower (c); cin >> c)
{
    if ('a' == tolower (c))
    {
        mouveLeftward ();
        break;
    }
    else if ('z' == tolower (c))
    {
        mouveUpward ();
        break;
    }
    ...
}
```



Dans une boucle, l'instruction `break` provoque la sortie de la boucle (la plus interne).

1. Le type caractère
2. Le type `string`
3. Boucle `while`
4. Boucle `for`
5. Boucle infinie
6. Rupture de schéma répétitif
7. Schéma séquentiel `switch / case`
8. Les sous-programmes
   - Procédure
   - Fonction
   - Passage de paramètres donnés
   - Passage de paramètres [données] résultats
9. Cosmétique
10. Le type `vector` (tableau de taille variable)
11. Boucle `for` pour les collections

Procédure

Modèle général algorithmique :

```
procedure nomProc (variable1 : [in|out|in_out] type1,
                   variable2 : [in|out|in_out] type2, ...);
```

Modèle général C++ :

```
void procName (variable1 : [in|out|in_out] type1,
               variable2 : [in|out|in_out] type2, ...);
```

Exemple

Procédure qui affiche "bonjour"

En algorithmique :

```
procedure affichBonjour ()
debut
    afficher ("bonjour");
    ligne_suivante;
fin
```

En C++ :

```
void showHello ()
{
    cout << "bonjour" << endl;

}// showHello ()
```

Fonction

Modèle général algorithmique :

```
fonction nomFonct (variable1 : [in|out|in_out] type1,
                   variable2 : [in|out|in_out] type2, ...)
renvoie TypeDeRetour;
```

Modèle général C++ :

```
returnType fctName (variable1 : [in|out|in_out] type1,
                    variable2 : [in|out|in_out] type2, ...);
```

A l'intérieur d'une fonction C++, l'instruction algorithmique `renvoie` se traduit par `return`.

Exemple

Fonction qui affiche "bonjour" et renvoie cette `string`

En algorithmique :

```
fonction affichEtRenvoieBonjour () renvoie string
debut
    afficher ("bonjour");
    ligne_suivante;
    renvoie "bonjour";
fin
```

En C++ :

```
string showHello ()
{
    cout << "bonjour" << endl;
    return "bonjour";
} // showHello ()
```

Passage de paramètres données

Modèle général algorithmique :

```
SP nomSP (variable1 : in type1, variable2 : in type2, ... )
[renvoie typeDeRetour];
```

Modèle général C++ :

```
[returnType|void] fctName (const type1 & varIdent1,
                           const type2 & varIdent2, ...);
```

Exemple : renvoyer le nombre de caractères d'espacement d'une `string` passée en paramètre

Exemple

```
unsigned spaceCount (const string & str)
{
    unsigned cpt = 0;
    for (size_t i = 0; i < str.size (); i = i + 1)
    {
        if (isspace (str[i])) cpt = cpt + 1;
    }
    return cpt;
} //spaceCount ()
```

Passage de paramètres [donnés] résultats

Modèle général algorithmique :

```
SP nomSP (variable1 : [in_]out type1, variable2 : [in_]out type2, ... )
[renvoie typeDeRetour];
```

Modèle général C++ :

```
[returnType|void] fctName (type1 & varIdent1, type2 & varIdent2, ...);
```

Exemple : écrire la procédure qui calcule un entier aléatoire entre les bornes `min` et `max`

Exemple

```
void rand (const int & min, const int & max, int & res)
{
    srand (time (NULL));
    res = min + rand () % (max - min);
} // rand ()
```

1. Le type caractère
2. Le type `string`
3. Boucle `while`
4. Boucle `for`
5. Boucle infinie
6. Rupture de schéma répétitif
7. Schéma séquentiel `switch / case`
8. Les sous-programmes
9. Cosmétique
   - Manipulateur `setw ()` ou cadrage à droite
   - Manipulateur `setfill ()`
10. Le type `vector` (tableau de taille variable)
11. Boucle `for` pour les collections

Manipulateur setw () ou cadrage à droite

Pré-requis :

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

Profil :

```
/*undefined*/ setw (int n);
```

Définition :

L'injection, dans un flux de sortie (cout), du manipulateur setw (Nb), provoque l'affichage de la variable ou du littéral qui le suit sur nb caractères (cadrés à droite). Les caractères manquants sont remplacés par des espaces.

Exemple

```
int a (10), b (20), c (30), d (40);
cout << setw (4) << a << setw (4) << b   __10__20
        << endl;
cout << setw (4) << c << setw (4) << d   __30__40
        << endl;
```

Manipulateur setfill ()

Pré-requis :

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

Profil :

```
/*undefined*/ setfill (char c);
```

Définition :
Le manipulateur `setfill ()` s'utilise avant `setw ()`. Il provoque le remplacement des espaces par le caractère passé en paramètre. `setfill ()` a un effet permanent.

Exemple

```
int a (10), b (20), c (30), d (40);
cout << setfill ('x') << setw (4) << a
     << setfill ('?') << setw (4) << b  << endl
     << setw (4) << c
     << setfill ('␣') << setw (4) << d
     << endl;
```

```
xx10??20
??30__40
```

Pré-requis

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

Identificateur

Modèle général algorithmique :

```
tableau_de unType;
```

Modèle général C++ :

```
vector <aType>
```

Exemple

```
vector <int> vInt;
vector <float> vFloat;
```

La taille du vecteur est nulle lors de sa création

Taille d'un tableau

Pour connaître la taille d'un `vector`, on appelle la `méthode () size ()` sur l'objet (la variable) de type `vector` (même nom que pour connaître la taille d'une `string`).

Exemple

```
cout << vInt.size ();  // 0
```

[C++17] Pour connaître la taille d'un `vector`, on appelle la `fonction size ()`. Cette fonction prend en paramètre l'objet (la variable) de type `vector` (même nom que pour connaître la taille d'une `string`).

Exemple

```
cout << size (v);  // 0
```

Insertion en fin de vecteur

Insertion en fin de vecteur

Pour insérer des éléments compatibles en fin d'un `vector`, on appelle la méthode `push_back ()` sur l'objet (la variable) de type `vector` (cette méthode existe aussi pour les `string`). Le vecteur est redimensionné automatiquement.

Exemple

```
vInt.push_back (0);
vInt.push_back (1);
...
vInt.push_back (9);
cout << vInt.size ();   //10
```

- Même exemple que précédemment en utilisant une boucle `for` :

Exemple

```
for (int insVal (0); insVal < 10; insVal = insVal + 1)
{
    vInt.push_back (insVal);
}
cout << vInt.size ();  //10
```

- Insertion de 10 valeurs saisies au clavier :

Exemple

```
for (size_t nbVal = 0; nbVal < 10; nbVal = nbVal + 1)
{
    int insVal;
    cin >> insVal;
    vInt.push_back (insVal);
}
cout << vInt.size ();  //10
```

Redimensionnement d'un vecteur

Pour redimensionner un `vector`, on appelle la `méthode resize () sur l'objet` (la variable) de type `vector`.

Exemple

```
vInt.resize (42);
```

Si la taille de l'ancien vecteur $<$ taille du nouveau (i.e. on ne fait pas de troncature), alors les nouvelles valeurs sont :

Type	Valeur par défaut
Entier	0
Réel	0.0
`string`	chaine vide
`char`	'\0'

Accès à un élément en lecture / écriture

On accède au $(i+1)^{\text{ème}}$ élément d'un `vector` en utilisant la notation `[i]`.

- La première case de lu `vector` a pour indice 0 ;
- La dernière case du `vector` a pour indice `varIdent.size () -1` ;
- Aucun contrôle quant à la validité de l'indice n'est effectué lors d'un accès à une case, que ce soit en lecture ou en écriture.

Exemple

Affichage du contenu d'un vecteur

```
for (size_t i = 0; i < vInt.size (); i = i + 1)
{
    cout << vInt [i];
}
```

Et surtout pas `cout « vInt;`

La boulette du débutant

Exemple

```
vector <int> vInt;
vInt [0] = 0;
```

Compilation : pas d'erreur
Exécution : Segmentation fault : 11 (anciennement écran bleu)

A l'aide

`vInt` est déclaré, mais non dimensionné

⇒ La case d'indice 0 n'existe pas!!

Recopie d'un tableau dans un autre

On souhaite recopier le contenu de `vInt` dans un second tableau (`vInt2`)

```
vector <int> vInt2;
```

- Solution 1 :

```
for (size_t i = 0; i < vInt.size (); i = i + 1)
{
    vInt2.push_back (VInt[i]);
}
```

- Solution 2 :

```
vector <int> vInt2;
vInt2 = vInt;
// vector <int> vInt2 = vInt;
```

- Solution 3 :

```
vector <int> vInt2 (vInt);
```

Boucle `for` pour les collections

En algo. on aimerait avoir une boucle de parcours de collection du type :

```
pour_chaque (valeur dans idTableau)
faire
    ...
ffaire
```

Où `valeur` prend les valeurs successives des cases de `idTableau`.
Cette boucle existe en C++ :

```
for ([const] Type & varIdent : collectionVarIdent)
{
    ...
}
```

- `const` **DOIT** être présent si aucune modification d'une des cases de la collection ;
- `varIdent` et `collectionVarIdent` doivent être de type compatible.

Exemples

Afficher le contenu d'un vecteur

Exemple

```
for (const int & val : vInt)
{
    cout << val << '\t';
}
```

Saisir le contenu d'un vecteur

Exemple

```
for (int & val : vInt)
{
    cin >> val;
}
```

`vInt` doit avoir une taille définie avant l'entrée de boucle.