haute école
neuchâtel berne jura
arc ingénierie
saint-imier le locle delémont

VSL COMPILER

VSL for dummies

Auteur : David Jacot, Anthony Mougin

Lieux et date : Saint-Imier, le 13/01/09

Table des matières

1 Introduction

VSL est l'acronyme de ***very simple language*** (langage très simple). Ce langage à été développé pour s'apparenter à du ***C*** mais typeless (moins de typage). Il n'y a donc aucune déclaration de type tel que :

- int i;
- float j;

Cette manière de manier les objets ressemble donc à python. La notion de pointeur, de structure est aussi absente. Mais les principales fonctionnalités sont présentent. La compilation et l'exécution quand à elle ressemble à java : un compilateur vers du byte code et une machine virtuel pour l'interpréter.

2 Le code

2.1La base

Tout code VSL doit absolument posséder une fonction (voir chapitre 2.12) nommé ***main*** qui correspondra à l'entrée du programme. Elle possédera obligatoirement un et un seul argument qui sera les arguments du programme sous forme de tableau (le premier sera son nom).

2.2Commentaires

2.2.1 Ligne simple

Les commentaires simples sont précédés de //. Tout ce qui se situe après sur la même ligne est considéré comme commentaire.

Ex: toto=2; // Assignation de toto

2.2.2 Multi-ligne

Les commentaires multi-ligne sont précédées par /* et se terminent par */. Tout ce qui est entre est considéré comme commentaire. Il peut contenir des retours de ligne.

Ex: / Bonjour, cette fonction est*

*génial */*

function main() {

...

}

2.3 Les types

Les types acceptés sont:

- Les entiers (int)

  ex: 2, 234, 1234, -1...

- Les réels (float)

  ex: 2.0, 23.5, -7.3...

- Les chaînes de caractères (str)

  ex: "Bonjour!", "belle chaîne, non?"...

- Les tableaux (array)

  ex: t[0], array[2]...

On peut à tout moment connaître la longueur d'un chaîne grâce à la fonction ***len().***

2.4 Les variables

Les variables sont des chaînes de caractère ne commencent pas par un chiffre pouvant contenir tout caractères alpha-numériques et des soulignements (_).

2.4.1 Les locales

Toutes variables étant déclarée dans une fonction (assignation) est local à cette fonction si aucune déclaration de variable global du même nom ne ce trouve au début de cette fonction.

2.4.2 Les globales

A l'inverse, elle est globale si une telle déclaration existe.

2.4.3 Exemples

Function test() {

```
    global i;
    u=0;
    i=2;
}
```

Dans cette exemple, la variable *u* est local, alors que *i* est global.

2.5 Les constantes

Plusieurs constantes on déjà pré-implémentée:

- True / False : 1 / 0
- NULL : 0
- PI : 3.14.....

2.6 Les opérations arithmétiques

Les opérateurs arithmétiques simples sont implémentés: +, -, *, /. Le moins et le plus peuvent être utilisés comme opérateur unaire.

*Ex: 2*3, i+1, toto/hello(), -5*

Les chaîne de caractères n'accepte que l'opérateur +, et ne peuvent être concaténée qu'à des chaîne de caractères.

Ex: "Toto"+" dit bonjour.", "resultat: "+str(2)

*On peut également cumuler l'opération et l'assignation comme en C grâce à +=, -=, *= et /=.*

On peut également incrémenter et décrémenter grâce à ++ et --

2.7 Les opérations logiques

Les opérateurs logiques disponibles sont les opérateurs non bit-à-bit:

- && : le ***et*** logique
- || : le ***ou*** logique
- ^ : le ***xor*** logique

- ! : le ***non*** logique (opérateur unaire)

2.8 Les opérations mathématiques

Il existe plusieurs implémentations de méthode mathématiques (les fonctions prenant des angles en arguments sont en radians):

- ** : puissance

  *ex: 2**3 = 8*

- sqrt() : racine carré

  ex: sqrt(16) = 4

- exp() : exponentielle

  ex: exp(0) = 1

- log() : logarithme

  ex: log(1) = 0

- abs() : absolu

  ex: abs(-2) = 2

- cos(), sin(), tan() : cosinus, sinus, tangente
- arccos(), arcsin(), arctan() : inverse des versions précédente.

2.9 Les fonctions réservées

Il existe toute sorte de fonction réservée.

2.9.1 Les entrées-sorties

- Print():

  Affiche le paramètre sur la sortie standard. Un espace est automatiquement ajouter à la fin.

  Ex: *print("Bonjour");*

  print("Resultat: "+str(2));

  print(toto);

- println():

  Affiche un saut de ligne (équivaut à print("\n") ou print(""))

- read():

Lit sur l'entrée standard (clavier). Le retour est toujours une string.

2.9.2 Type des objets

On peut facilement connaître le type d'une variable, ou d'un élément. Ces fonctions sont on nombre de 4: ***isInt(), isFloat(), isStr(), isArray()***.

Ex: *t="Hello!";*

print(isStr(t)); // Affiche '1'

2.9.3 Transtypage des objets

Il est possible de convertir des objets d'un type vers un autre. Les méthodes sont au nombre de 3: ***int(), float, str()***.

Ex: *t=int(read()) // Permet de lire un entier*

2.9.4 Tableaux

Les tableaux sont déclarés comme en python, c'est une assignation du constructeur.

Ex: tab = [ ];

On peut ensuite accéder à chaque élément grâce a son index (si ce dernier existe).

Ex: *a = tab[0];*

print(tab[index]);

On peut aussi donner une valeur à un élément (si ce dernier existe).

Ex: *tab[0] = 2;*

tab[index] = "Bonjour"

Un tableau peut contenir plusieurs types de données différentes. Plusieurs méthodes sont à votre disposition:

- get(p,i) : prendre l'objet à l'indice i dans le tableau p
- set(p,i,e) : attribué à l'objet à l'indice i dans le tableau p la valeur e
- append(p,e) : ajouter au tableau p la valeur e (agrandissement)

- insert(p,i,e) : ajouter à l'indice i du tableau p la valeur e (agrandissement)
- remove(p,i) : suppression de l'élément i du tableau p (rétrécissement)

On peut à tout moment connaître le nombre d'élément dans un tableau grâce à la fonction ***len().***

2.9.5 Longueur

La méthode **len()** peut-être utilisé dans 2 cas:

- Sur les chaînes de caractère pour connaître leur longueur.
- Sur les tableaux pour obtenir le nombre d'éléments.

2.10 Embranchement conditionnel

Les embranchements conditionnels sont comparable au C, mis part que les *else if*(...) sont remplacer par les *elif*(...) comme en python. Si le programme à exécuter fait plus d'une ligne, il faut alors l'entourer de '{'...'}'. Sinon, aucune accolade n'est nécessaire.

```
Ex:   if(Ok) print("Ok!")
      if(Ok) {
            ...
      }
      elif(Ok2) {
            ...
      }
      else print("Stop");
```

2.11 Les boucles

2.11.1 For

Les boucles ***for*** sont identique au C, mis a part qu'elle n'accepte au maximum qu'une initialisation. Même principe pour les accolades qu'avec le ***if***.

```
Ex:   for(t=2;t<10;t++) {
```

```
        ...
}

for(;;) {
        ... // Boucle infini
}
```

2.11.2 While

Les ***while*** quand à elles sont comme ne C. La condition est évaluée la première fois. Même principe pour les accolades qu'avec le ***if***.

```
Ex:   while(ok) {
        ...
}

while(!Toto) print("Pas de toto");
```

2.11.3 Do – while

Les do-while quand elle exécute au moins une fois le programme avant de vérifier la condition. Contrairement au C, la fin de la structure ne nécessite aucun point-virgule. Même principe pour les accolades qu'avec le ***if***.

```
Ex:   do {
        ...
}while(Ok)
```

2.12 Les fonctions

Les fonctions sont déclarée grâce au mot réservé ***function***. Il est suivit du nom de la fonction puis d'une parenthèse ouvrante et une parenthèse fermante. Il peut y avoir entre ces parenthèses 0 ou plusieurs arguments (variables) séparé par des virgules. Le programme de la fonction doit impérativement être délimité par des accolades. Tous les noms mis à part ***main*** peuvent être utilisé.

```
Ex:   function printDouble() {
```

```
        print("Double");
    }
```

La fonction peut retourner ou non quelque chose. Si tel est le cas, le retour ne peut se faire qu'à la fin du sous-programme. Elle se fait grâce au mot réservé ***return*** suivant de l'expression à retourner.

```
Ex:  function double(x) {
        if(isInt(x)) retour = 2*x;
        else retour = "Not an integer"
        return retour;
    }
```

Pour exécuter la fonction, il suffit de donner son nom suivit d'une parenthèse ouvrante, des paramètres si nécessaire, et d'une parenthèse fermante.

```
Ex:  printDouble();
     print(double(2));
     print(double("Blague!"));
```

2.13 Import

Il est possible de séparer les méthodes en plusieurs fichiers. Pour utiliser une méthode qui se situe dans un fichier VSL différent, il suffit de préciser en haut du fichier dans quelle fichier on doit chercher grâce au mot réservé ***import*** suivit du chemin du fichier:

```
Ex: import "string.vsl";
```

Il est possible de prendre un raccourci pour l'import:

```
Ex: import "string";
```

3 Compilation

Pour compiler un fichier VSL, il existe deux manières:

- vslc monfichier.vsl

  Le fichier à compiler est le fichier principal (celui qui contient le main)

- vmake monfichier.vsl

  Permet de créer un makefile dans le dossier courant permettant de compiler le fichier en paramètre grâce à la commande make. Si aucun fichier n'est spécifié, le makefile compilera tout les fichiers comme si il était des programmes uniques.

4 En savoir plus

Pour en savoir plus, vous pouvez nous contacter:
david.jacot@he-arc.ch - anthony.mougin@he-arc.ch

Vous pouvez aussi consulter les exemples fournit dans le dossier **sample**.