LANGAGE SPL

Simplified Procedural Language

I. Introduction

Le **SPL** (Simplified Procedural Language) est un langage de programmation basique interprété en **C** (i.e. l'interpréteur traduit le **SPL** en **C** avant d'utiliser un compilateur **C** pour générer un exécutable).

Beaucoup plus simple et basique que le **C** pour un débutant, il est aussi beaucoup plus simple d'approche.

Ce document détaille les différentes règles d'écriture du langage.

II. Généralités

Divers

La syntaxe du langage suit le format suivant :

```
Nom_de_l'élément
[
        Instructions ;
        Autre_élément
        [
                Autre instruction ;
        ]
]
```

Les instructions doivent obligatoirement être écrites dans une fonction, dans la section code (Voir la section sur les fonctions).

Écrire du code en dehors de ces balises causera une erreur de l'interpréteur

Si une variable est déclarée en dehors de toute fonction, alors celle-ci sera globale (i.e. accessible depuis n'importe quel point du programme). (Voir la section sur les variables).

La SPL est un langage qui n'est pas sensible à la casse, pour lui la variable a et A appellerons la même variable. De même, les espaces n'ont aucune importance, et tout caractère non ASCII ou n'ayant aucune utilisé dans le langage sera supprimé lors de la compilation.

foo(a,b) représentera la même chose que foo (a, b)

Commentaires

Les commentaires peuvent être multi ligne ou bien sur une seule ligne. Pour ces derniers il suffit de commencer la ligne avec **#**. Pour les multi-lignes, il faut utiliser les balises **#--** et **--#** autour de la zone à ne pas tenir compte.

III. Variables

Déclaration

Une déclaration de variable doit toujours avoir la même forme :

```
Var
[
        Name = nom_de_votre_variable ;
        Type = int ;
        Value = 0 ;
]
```

Name :

Représente le nom de la variable. Ce nom ne peut être composé que des caractère de **a** à **z**, de **A** à **Z** ou du tiret-bas (**_**).

Cet élément de la déclaration est obligatoire.

Type :

Représente le type de la variable. Pour plus d'informations voir la partie sur les types de données.

Cet élément de la déclaration est obligatoire.

Value :

Représente la valeur d'initialisation de la variable. Si aucune valeur n'est indiquée, alors 0 sera attribué.

Cet élément de la déclaration est facultatif.

Etant insensible à la casse et ne tenant pas compte des espaces la déclaration suivante fait exactement la même chose que celle donnée en exemple :

var[name=Nom_De_Votre_Variable ;type=INT;]

Portée des variables

Variables globales

Les variables globales, déclarées en dehors de toute fonction, ont pour portée tout le programme. C'est-à-dire que chaque variable globale est accessible n'importe où dans le programme.

Arguments de fonctions

Les arguments de fonction ont pour portée l'intérieur de leur fonction associée.

Variables usuelles

Les variables usuelles ont pour portée l'élément (Boucle, condition, ou même tout le code de la fonction) dans lequel elles sont déclarées.

Recouvrement de variable

Déclarer une variable avec le même nom qu'une variable ayant une portée plus importante aura pour effet de recouvrir cette dernière. Même si la variable en elle-même n'est pas affectée, il sera impossible de l'utiliser.

Déclarer deux variables avec le même nom et le même niveau est interdit.

Déclarer 2 variables de même nom au même niveau causera une erreur de l'interpréteur.

IV. Fonctions

Déclaration

Une fonction doit toujours être déclarée de la même manière :

```
Func
[
        Name = nom_de_votre_fonction ;
        Type = int ;
        Args
        [
                Var[ name = a ; type = int ; ]
        ]
        Code
        [
                # Code
        ]
]
```

Name :

Représente le nom de la fonction. Ce nom ne peut être composé que des caractère de **a** à **z**, de **A** à **Z** ou du tiret-bas (**_**).

Cet élément de la déclaration est obligatoire.

Type :

Représente le type de retour de la fonction. Pour plus d'information voir la partie sur les types de données.

Cet élément de la déclaration est facultatif, et sera mis à void en cas de non indication.

Args :

Représente la liste des arguments de la fonction. Il suffit de les déclarer comme des variables normales. Les valeurs envoyées en argument sont copiée dans la fonction, pour pouvoir les modifier directement voir les références.

Code :

Il s'agit du code de la fonction. Voir la partie sur les instructions du langage pour plus d'informations.

Cet élément de la déclaration est obligatoire.

Point d'entrée du programme

Le point d'entrée est simplement définit par start :

```
Start
[
        # Code
]
```

Il est obligatoire de définir un point d'entrée dans le programme. Il ne peut y avoir plus d'un seul point d'entrée.

Ne pas déclarer de point d'entrée, ou en déclarer plus d'un causera une erreur de compilation.

Règle de nommage

Il n'est pas possible de créer deux fonctions portant le même nom. Il est possible de recouvrir les fonctions built-in, mais elles seront alors inaccessibles dans tout le programme.

Il est possible d'avoir des fonctions et des variables ayant le même nom.

Fonctions built-in

Il existe pour le moment deux fonctions prédéfinies :

Return()

Sert à retourner une valeur et à quitter la fonction ou le programme si utilisé dans le principal du programme.

Cette fonction peut ne prendre aucun argument, ou un seul argument.

Out()

Sert à afficher une ou plusieurs variables sur l'écran de retour.

Cette fonction peut ne prendre aucun argument, ou bien plusieurs.

V. Références

Les références ne peuvent être utilisées que comme argument de fonction. Elles se déclarent de la même manière qu'une variable classique à la différence qu'on doit les déclarer avec le mot clef « **ref** » au lieu de « **var** ».

```
Ref
[
        Name = nom_de_votre_référence ;
        Type = int ;
        Value = 0 ;
]
```

Les références sont, comme l'indique leur nom, des références vers d'autres variables. Au lieu de copier la valeur, le programme établi un lien entre la valeur envoyé en argument et celle traitée dans le programme.

Ainsi modifier la référence revient à modifier l'argument.

Il est nécessaire que l'envoi de référence se fasse uniquement à partir de variables du même type que l'argument.

Envoyer une expression ou une variable du mauvais type causera une erreur de compilation.

Les éléments de la déclaration sont les mêmes que pour une variable classique.

VI. Types de variables

Il existe en SPL pour le moment 3 types de variables :

- Int, pour les entiers
- Char, pour les caractères
- Float pour les nombres à virgule

Règles de conversions

Le SPL utilise des règles très simples pour la conversion des données. Il fonctionne de façon à éviter de perdre des informations. Ainsi si un entier est multiplié par un flottant, l'entier sera converti en flottant avant le calcul pour ne pas perdre d'informations.

Cependant, si l'on souhaite ranger une valeur dans une variable, alors la valeur sera convertie obligatoirement suivant le type de la variable (De même lors de l'envoi d'argument).

Il n'existe pas pour l'instant d'opérateur de conversion explicite

Il existe également un type vide (void) utilisé pour définir des procédures (fonction sans retours). Ce type ne peut pas être utilisé comme type de variable.

VII. Instructions

Les instructions doivent être situées soit dans la balise code d'une **fonction**, ou bien dans la balise **start** du programme.

Chaque instruction doit être terminée par un ; et ne peut comporter que des instructions ayant un lien entres-elle.

Par exemple « a+b 1-2 ; » est incorrect car les deux séries d'instructions n'ont aucun lien. Ceci causera une erreur du compilateur.

Pour inscrire plusieurs instructions à la suite sur la même « ligne » de code il faut utiliser l'opérateur « , » :

```
a + b, 1 – 2 ;
```

Règle d'utilisation des opérateurs

Priorité des opérateurs

Voici la liste des priorités des opérateurs du SPL. Cette liste concerne l'ordre de traitement :

Opérateur	Définition	Exemple
(..)	Parenthèses	(a*(b-2)) ;
++ --	Incrémentation/Décrémentation	a++ ;
+ -	+ et – unaires	-2 ;
!	Inverseur (si = 0 : 1, sinon 0)	!a ;
*** / % + -**	Opérateurs de calcul	a + b/2 * 4 ;
> <	Opérateurs de comparaison	a > b ;
>= <= == !=	Opérateurs de comparaison	a == b, b != c ;
&& ||	ET et OU	a && b ;
=	Opérateur d'affectation	a = 2 ;
+= -= *= /= %=	Opérateur d'affectation avec calcul	a += 2 ;

Cas spéciaux

L'opérateur d'incrémentation (++, --) ne peut être utilisé que sur une variable. L'utiliser sur autre chose causera une erreur de compilation.

L'opérateur d'affectation (=, *=, /=, %=, -=) ne peut être utilisé que sur une variable. L'utiliser sur autre chose causera une erreur de compilation.

Attention, « a + 1 = 12 ; » causera une erreur car on essaie d'affecter le résultat « 12 » à « a + 1 » qui n'est pas une variable.

Il n'est pas possible d'utiliser un résultat de type void (Par exemple une fonction ne renvoyant aucune valeur) dans une opération.

Utiliser des variables

Pour utiliser une variable il suffit d'utiliser son nom dans le code. Une affectation ou une opération d'incrémentation causera une modification de sa valeur.

Appels de fonctions

Les appels de fonction doivent comporter le nom de la fonction suivit des arguments entre parenthèses séparés par des virgules :

```
Foo(a, b) ;
```

Attention, même si la fonction ne prend pas d'arguments il est obligatoire d'utiliser les parenthèses, sinon celle-ci sera interprétée comme une variable.

Il est nécessaire de fournir autant de variables que la fonction nécessite d'arguments. Dans le cas contraire le compilateur générera une erreur.

Boucles et conditions

Les boucles et les conditions utilisent le même format :

```
Mot_clef(condition)
[
    instructions ;
]
```

La condition est une simple instruction. Il est possible d'en définir plusieurs si on les délimite avec des « , ».

Boucles

La seule boucle utilisable est le **while**. Cette boucle est parcourue tant que la condition définie dans le **while** est vraie.

Il est obligatoire de définir une condition pour une de type while.

Il existe également un type de boucle « vide » sans mot clef et sans condition qui peut servir à uniquement changer de niveau pour par exemple redéfinir une variable.

Conditions

Il existe 3 mots clefs conditionnels. **If** et **elseif** qui prennent une condition, et **else**. Il est à noter que **else** et **elseif** doivent **oblagatoirement** être directement à la suite d'un **if.**

VIII. Exemples

Des programmes d'exemples (imprimé) sont donnés en annexe des documents. Ils sont également donnés en fichier avec le programme.