	Méthode de programmation en C--	**Date rédaction :** **13 mars 2004.**
	Diffusion restreinte	**Date validation :**
Référence : UpComp-UpsVtm-000004-A Méthode de programmation en C--.doc		

Suivi des versions-révisions et des validations du document.			
Ce document annule et remplace tout document diffusé de version-révision antérieure. Dès réception de ce document, les destinataires ont pour obligation de détruire les versions-révisions antérieures, toutes les copies, et de les remplacer par cette version. Si les versions-révisions antérieures sont conservées pour mémoire, les destinataires doivent s'assurer qu'elles ne peuvent être confondues avec cette présente version-révision dans leur usage courant.			
Version.	**Date.**	**Auteurs.**	**Création, modification ou validation.**
A	23 nov. 2003	JPD.	Création.

1 Tables

1.1 Table des matières

1.2 Table des illustrations

2 Références

2.1 Glossaire

Liste des définitions des termes employés.	
Ce tableau recense tous les termes, les concepts particuliers ainsi que les abréviations employés dans ce document.	
Terme, concept, abré.	**Définition du terme, du concept ou de l'abréviation.**

2.2 Ressources

Liste des documents applicables et en référence.

Un document est **applicable** à partir du moment où son contenu est validé et que l'activité ou le projet fait partie de son périmètre d'application. Il est obligatoire d'appliquer son contenu.

Un document est en **référence** à partir du moment où son contenu n'est pas validé ou que l'activité ou le projet ne fait partie de son périmètre d'application. Il est recommandé d'appliquer son contenu mais cela n'est pas obligatoire.

Un document applicable est indicé par ***A1***, ***A2***, ***A3***, etc. Un document en référence est indicé par ***R1***, ***R2***, ***R3***, etc.

Index.	Nom du document.	Commentaire.
A1	UpComp-Plan Qualité-000005	Méthode documentaire.
A2	UpComp-UpsVm-000003	Plan de programmation.

3 Introduction

3.1 Objet du document

L'objet du document est de présenter la méthode de programmation en ***C--***. Cette méthode est applicable à tous les modules écrits manuellement pour fabriquer ***Up ! Application System***.

3.2 Audience

Ce document s'adresse à tout ingénieur chargé de :

- L'étude, de la réalisation ou de la maintenance d'un module natif.
- L'écriture du générateur de code pour ***Up ! P32***.

3.3 Pré-requis

Le pré-requis est la connaissance de la programmation en ***C / C++***.

4 Pourquoi C-- ?

4.1 Avantages de C--

La programmation en ***C--*** offre les avantages suivants :

- Etre totalement portable.
  Le langage ***C--*** est le sous-ensemble des langages ***C / C++*** portable à l'identique en terme de fonctionnement sur toutes les plates-formes matérielles connues à ce jour. Aucune modification n'est à apporter au fichier source, pas même une directive de pré processeur supplémentaire. Couplée aux ***Application Program Interfaces*** (**API**) d'***Up ! Virtual Technical Machine***, il permet d'écrire des programmes totalement portables sur toutes les plates-formes.
- Etre simple d'emploi.
  Le langage ***C--*** est fort simple en comparaison des concepts complexes de ***C++*** et ***Java***. Il permet à un informaticien ayant peu de compétences techniques de réaliser aisément des petits programmes.
- Etre fiable.
  Le langage ***C--*** ne tolère aucune astuce de programmation tel qu'il serait possible de les imaginer en langage ***C***. Il intègre automatiquement les normes de programmations décrites dans le document **Plan de programmation** [A1].
- Etre rapide à l'exécution.
  Le langage ***C--*** s'apparente plus à macro-assembleur portable i.e. un ***Langage de 2ème Génération*** (**L2G**) qu'à un ***Langage de 3ème Génération*** (**L3G**) comme les langages ***C++*** ou ***Java***. De ce fait, il est naturellement rapide à l'exécution, sans pour autant devoir réaliser des optimisations fines.
- Pouvoir être généré automatiquement.
  Le langage ***C--*** est généré automatiquement par ***Up ! Compiler*** à partir des fichiers sources ***Idl Corba***, ***Idl DCom***, ***Java*** ou ***Up ! 5GL***.
- Pouvoir être écrit manuellement.
  Le langage ***C--*** est utilisé pour écrire les modules constituant ***Up ! Virtual Technical Machine*** :
  - ***Up ! Kernel***.
  - ***Up ! Mathematical***.
  - ***Up ! Module***.
  - ***Up ! Natural Language Support***.
  - ***Up ! Network***.
  - ***Up ! Object Management System***.
  - ***Up ! Object Request Broker***.
  - ***Up ! Security Management System***.
  - ***Up ! Starter***.
  - ***Up ! System***.
- Pouvoir être compilé en natif.
  Comme le langage ***C--*** est un sous-ensemble des langages ***C / C++***, il peut être compilé à l'identique en terme de fonctionnement sur toutes les plates-formes matérielles disposant d'un

compilateur ***C / C++*** respectant la norme ***Ansi*** du standard ***C***. En particulier, il peut être compilé par le compilateur ***Gnu***.

- Pouvoir être compilé pour le processeur virtuel ***Up ! P32***.
  Up ! Compiler génère automatiquement des modules en pseudo-code binaire qui sont interprétés à l'exécution par ***Up ! Engine***.

4.2 Comparaison avec les langages C / C++ / Java

Le langage ***C--*** est un sous-ensemble du langage ***C*** mais utilisé en ***C++***. La raison est que les compilateurs ***C++*** sont beaucoup plus fiables que les compilateurs ***C*** parce qu'ils reposent sur des normes plus récentes et plus restrictives.

Ce sous-ensemble diffère cependant du langage ***Java***, du fait des concepts suivants :

- Classes obligatoires.
  Java est un langage « intégriste » dans le sens où il oblige d'utiliser la programmation objet même quand cela n'est pas nécessaire. Le procédural est banni !
  Le résultat est la profusion de centaines de classes au code éparse, rendant les programmes difficilement compréhensibles, ce qui nuit à leur fiabilité et à leur maintenance.
- Pointeurs absents.
  Java ne permet pas une manipulation aisée des pointeurs de données et n'accepte pas une manipulation des pointeurs de traitements. Si cette décision est louable dans le sens de simplifier la compréhension des programmes, elle est dommageable pour la souplesse de programmation nécessaire à l'écriture de modules efficaces.

Voici un diagramme présentant les relations entre ces langages :

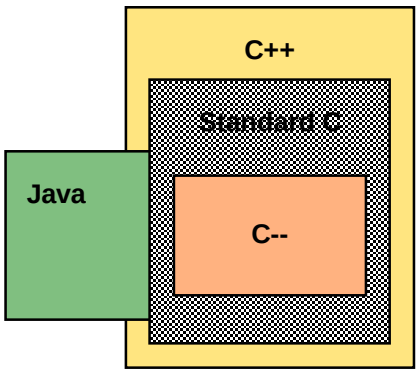

Diagramme 1 – Positionnement de C-- par rapport aux autres langages C / C++ / Java

5 Définition du langage C--

5.1 Définitions lexicales

5.1.1 Commentaire

Le début d'un bloc de commentaires est signalé par **/*** et sa fin est signalée par ***/**. Voici un exemple :

```
I=1;
/* Premiere ligne de commentaire.
Deuxieme ligne de commentaire. */
J=1;
```

Texte 2 – Exemple de commentaire

M Il n'est pas possible d'imbriquer les blocs de commentaires.

M Il n'est pas possible de mettre une ligne en commentaire avec la marque ***//***.

5.1.2 Constante entière

Une constante entière est composée par une succession de chiffres. Voici des exemples de constantes entières :

```
0
1
4567
'A'
```

Texte 3 – Exemple de constantes entières

Voici des contre-exemples de constantes entières :

```
-3
0.5
'abcd'
abcd
```

Texte 4 – Contre-exemple de constantes entières

M Les constantes entières sont implicitement positives ou nulles. En fait le signe moins est interprété comme l'opérateur unaire de changement de signe, ce qui revient sensiblement au même.

5.1.3 Constante réelle

Une constante réelle est composée d'une succession de chiffres, suivie du caractère **point .** , suivi éventuellement d'une succession de chiffres. Un exposant peut être précisé introduit par la lettre ***E***.

Voici des exemples de constantes réelles :

```
0.
0.0
2.3345
23.487
23E7
45E-6
```

Texte 5 – Exemple de constantes réelles

Voici des contre-exemples de constantes réelles :

```
8
-1
.5
-0.7
abcd
45E
```

Texte 6 – Contre-exemple de constantes réelles

M Les constantes réelles sont implicitement positives ou nulles. En fait le signe moins est interprété comme l'opérateur unaire de changement de signe, ce qui revient sensiblement au même.

5.1.4 Identificateur

Un identificateur est composé d'une lettre suivie éventuellement par une succession de lettres, de chiffres, ou du caractère **souligné** _.

Voici des exemples d'identificateur :

```
abcde
a123
def_ad167
```

Texte 7 – Exemple d'identificateurs

Voici des contre-exemples de constantes réelles :

```
18
3.14
123aa
_abcde
_aaa
```

Texte 8 – Contre-exemple d'identificateurs

M Un identificateur comporte au plus 50 caractères.

5.1.5 Chaîne de caractères constante

Une chaîne de caractères constante est composée d'une succession de caractères encadrée par deux caractères **guillemets** ".

Voici des exemples de chaînes de caractères :

```
"liste des employés"
"pi=3.14159"
```

Texte 9 – Exemple de chaînes de caractères constantes

Voici des contre-exemples de constantes réelles :

```
'abcd'
abcd
12
```

Texte 10 – Contre-exemple de chaînes de caractères constantes

Pour inclure dans une chaîne le caractère **guillemet "**, il faut le précéder par le caractère **division inversée **. Voici un exemple :

```
"citation :\"essai\"."
```

Texte 11 – Exemple d'emploi d'un caractère guillemet dans un chaîne de caractères constante

Pour inclure dans une chaîne le caractère **division inversée **, il faut le précéder par le caractère **division inversée** \. Voici un exemple :

```
"c:\\tmp\\exemple.txt"
```

Texte 12 – Exemple d'emploi d'un caractère division inversée dans un chaîne de caractères constante

Si une chaîne de caractère doit être coupée parce qu'elle est trop longue, il suffit d'ajouter le caractère **division inversée ** en fin des premières lignes composant la chaîne. Voici un exemple :

```
"ceci \
est un \
essai"
```

Texte 13 – Exemple de chaîne de caractères constante sur plusieurs lignes

5.2 Définitions syntaxiques

5.2.1 Syntaxe des instructions du pré processeur

M Une commande concernant le pré processeur commence toujours par le caractère **dièse #** mise en premier caractère de la ligne. Il ne peut donc y avoir qu'une commande par ligne de programme.

Instruction :

`define` ***NomDuSymbole*** *ParametresOptionnels* ***DefinitionDuSymbole***

| **`undef`** ***NomDuSymbole***

| **`include`** ***NomDuFichier***

| **`line`** ***NumeroDeLigne NomDuFichier***

| **`if`** *Expression*

| **`elif`** *Expression*

| **`else`**

| **`endif`**

| **`pragma`** ***Directive***

;

ParametresOptionnels :

| **(** *Parametres* **)**

;

Parametres :

NomDuSymbole

| *Parametres* **,** ***NomDuSymbole***

;

Expression :

| **`defined`** **(** ***NomDuSymbole***)

| **(** *Expression* **)**

| **!** *Expression*

| *Expression* **&&** *Expression*

| *Expression* **||** *Expression*

;

5.2.2 Syntaxe des instructions du langage C--

a Les types élémentaires sont ceux d'***Up ! Virtual Technical Machine***. Il y a :

Types standards	Nom littéral du module
TypUpsVmChar	Entier sur un octet de valeur appartenant à [-128, 127].
TypUpsVmUnsignedChar	Entier sur un octet de valeur appartenant à [0, 255].
TypUpsVmShort	Entier sur deux octets de valeur appartenant à [-32768, 32767].
TypUpsVmUnsignedShort	Entier sur deux octets de valeur appartenant à [0, 65536].
TypUpsVmLong	Entier sur quatre octets de valeur appartenant à [-2147483648, 2147483647].
TypUpsVmUnsignedLong	Entier sur quatre octets de valeur appartenant à [0, 4294967296].
TypUpsVmDouble	Réel sur huit octets.
TypUpsVmUnicode	Caractère sur deux octets.
TypUpsVmPointeurDonnees	Pointeur vers une interface de données indéfinie.
TypUpsVmPointeurTraitements	Pointeur vers une interface de traitements indéfinie.

Tableau 14 – Type de données d'Up ! Virtual Technical Machine

Programme :

DebutLangageC

DefinitionDesEnumeres

DefinitionDesTypes

DefinitionDesVariablesGlobales

DefinitionDesFonctions

FinLangageC

;

DebutLangageC :

| **extern** *"C"* **{**

;

FinLangageC :

| **}**

;

DefinitionDesEnumeres :

| *DefinitionDesEnumeres Enumere*

;

Enumere :

enum ***NomDeLEnumere***

{

DefinitionDesValeurEnumerees

};

;

DefinitionDesValeursEnumerees :

ValeurEnumeree

| *DefinitionDesValeursEnumerees* **,** *ValeurEnumeree*

;

ValeurEnumeree :

NomDeLaValeurEnumeree = ***ConstanteEntiere***

;

DefinitionDesTypes :

| *DefinitionDesTypes Type*

;

Type :

typedef *SuiteType* **;**

;

SuiteType :

TypeDeDonnees ListeDAlias

| *TypeDeTraitements*

;

TypeDeDonnees :

NomDUnType

| **struct** ***NomDeLaStructure***

{

ListeDesChamps

}

| **union** ***NomDeLUnion***

{

ListeDesChamps

}

;

ListeDeChamps :

Champ

| *ListeDeChamps Champ*

;

Champ :

Definition **;**

;

Definition :

NomDUnType *PointeurOption* ***NomDuParametre*** *TableauOption*

| ***NomDUnType*** *PointeurOption* **(*** ***NomDuParametre*** **) (** *ListeDeParametres* **)**

;

PointeurOption :

| *PointeurOption* *****

;

TableauOption :

| *TableauOption* **[** *TailleDuTableauOption* **]**

;

TailleDuTbleauOption :

| ***TailleDuTableau***

;

ListeDesParametres :

`void`

| *ListeDesParametres2*

;

ListeDesParametres2 :

Definition

| *ListeDesParametres2* **,** *Definition*

;

ListeDAlias :

Alias

| *ListeDAlias Alias*

;

Alias :

PointeurOption ***NomDeLAlias*** *TableauOption*

;

TypeDeTraitements :

NomDUnType *PointeurOption* **(*** ***NomDuType*** **) (** *ListeDeParametres* **)**

;

DefinitionDesVariablesGlobales :

| *DefinitionDesVariablesGlobales VariableGlobale*

;

VariableGlobale :

PrefixeDeclaration Definition ValeurParDefaut **;**

;

PrefixeDeclaration :

| **extern**

| **static**

;

ValeurParDefaut :

| = ***ConstanteCaractere***

| = ***ConstanteEntiere***

| = ***ConstanteReelle***

;

DefinitionDesFonctions :

| *DefinitionDesFonctions Fonction*

;

Fonction :

PrefixeDeclaration TypeResultat ***NomDeLaFonction*** **(** *ListeDeParametres* **)** *SuiteFonction*

;

SuiteFonction :

{

DefinitionDesVariablesLocales

Instructions

}

| **;**

;

TypeResultat :

void

| ***NomDUnType*** *PointeurOption*

;

DefinitionDesVariablesLocales :

| *DefinitionDesVariablesLocales VariableLocale*

;

VariableLocale :

Definition **;**

;

Instructions :

| *Instructions Instruction*

;

Instruction :

{

Instructions

}

| *Expression* **;**

| *Arreter* **;**

| *Continuer* **;**

| *Retourner* **;**

| *Boucler*

| *Tester*

| *Eclater*

;

Expression :

ConstanteCaractere

| ***ConstanteEntiere***

| ***ConstanteReelle***

| ***NomDUneVariable*** *SuiteIdentificateur*

| ***NomDUneFonction***

| **(** *Expression* **)**

| *ExpressionUnaire*

| *ExpressionBinaire*

| **(** ***NomDUnType*** *PointeurOption* **)** *Expression*

| **`sizeof`** **(** *NomDUnType* **)**

| ***NomDUneFonction*** **(** *ListeDExpressionsOption* **)**

;

SuiteIdentificateur :

| **.** ***NomDUnChamp*** *SuiteIdentificateur*

| **->** ***NomDUnChamp*** *SuiteIdentificateur*

| **[** *Expression* **]** *SuiteIdentificateur*

;

ExpressionUnaire :

- *Expression*

| **++** *Expression*

| **--** *Expression*

| *Expression* **++**

| *Expression* **--**

| **!** *Expression*

| **~** *Expression*

| **&** *Expression*

| ***** *Expression*

;

ExpressionBinaire :

Expression **+** *Expression*

| *Expression* **-** *Expression*

| *Expression* ***** *Expression*

| *Expression* **/** *Expression*

| *Expression* **%** *Expression*

| *Expression* **&** *Expression*

| *Expression* **|** *Expression*

| *Expression* **^** *Expression*

| *Expression* **&&** *Expression*

| *Expression* **||** *Expression*

| *Expression* **<<** *Expression*

| *Expression* **>>** *Expression*

| *Expression* **==** *Expression*

| *Expression* **!=** *Expression*

| *Expression* **<=** *Expression*

| *Expression* **>=** *Expression*

| *Expression* **<** *Expression*

| *Expression* **>** *Expression*

| *Expression* **=** *Expression*

| *Expression* **+=** *Expression*

| *Expression* **-=** *Expression*

| *Expression* ***=** *Expression*

| *Expression* **/=** *Expression*

| *Expression* **%=** *Expression*

| *Expression* **<<=** *Expression*

| *Expression* **>>=** *Expression*

| *Expression* **&=** *Expression*

| *Expression* **|=** *Expression*

| *Expression* **^=** *Expression*

| *Expression* **&&=** *Expression*

| *Expression* **||=** *Expression*

;

NomDUnType :

NomDUnType *PointeurOption*

| **`struct`** ***NomDUneStructure***

| **`union`** ***NomDUneUnion***

;

ListeDExpressionsOption :

| *ListeDExpressions*

;

ListeDExpressions :

Expression

| *ListeDExpressions* **,** *Expression*

;

Arreter :

`break`

;

Continuer :

`continue`

;

Retourner :

`return` *ExpressionOption*

;

ExpressionOption :

| *Expression*

;

Boucler :

`for` **(** *ExpressionOption* **;** *ExpressionOption* **;** *ExpressionOption* **)**

Instruction

| **`do`** *Instruction*

`while` **(** *Expression* **) ;**

| **`while`** **(** *Expression* **)**

Instruction

;

Tester :

if **(** *Expression* **)** *Instruction SuiteTester*

;

SuiteTester :

| **else if** **(** *Expression* **)** *Instruction SuiteTester*

| **else** *Instruction*

;

Eclater :

switch **(** *Expression* **)**

{

ListeDeCas CasParDefaut

}

;

ListeDeCas :

Cas

| *ListeDeCas Cas*

;

Cas :

case ***ConstanteEntiere*** **:** *Instructions*

;

CasParDefaut :

default **:** *Instructions*

;

6 Référence du pré processeur

6.1 Opérateurs du pré processeur

6.1.1 !

6.1.1.1 Prototypes

! *Expression*

6.1.1.2 Description

Inversion de la valeur de l'expression :

- Si la valeur de l'expression préfixée est évaluée à ***Vrai***, alors le résultat est ***Faux***.
- Si la valeur de l'expression préfixée est évaluée à ***Faux***, alors le résultat est ***Vrai***.

6.1.1.3 Exemple

```
/*****************************************************/
#if (defined(AIX43)||defined(SCO32))&&!defined(LINUX20)
/*****************************************************/
...
/****/
#endif
/****/
```

6.1.1.4 Avertissement

Néant.

6.1.1.5 Voir aussi

( ) pour isoler une expression.

6.1.2 ( )

6.1.2.1 Prototypes

(*Expression* **)**

6.1.2.2 Description

Isole une expression. Le résultat est la valeur de l'expression.

6.1.2.3 Exemple

```
/*****************************************************/
#if (defined(AIX43)||defined(SCO32))&&!defined(LINUX20)
/*****************************************************/
...
/****/
#endif
/****/
```

6.1.2.4 Avertissement

Néant.

6.1.2.5 Voir aussi

! pour inverser la valeur d'une expression.

6.1.3 ||

6.1.3.1 Prototypes

Expression **||** *Expression*

6.1.3.2 Description

Disjonction inclusive de deux expressions :

- Le résultat est ***Vrai*** si l'une des deux expressions vaut ***Vrai***.
- Le résultat est ***Faux*** si les deux expressions valent ***Faux***.

6.1.3.3 Exemple

```
/*************************************************/
#if (defined(AIX43)||defined(SCO32))&&!defined(LINUX20)
/*************************************************/
...
/****/
#endif
/****/
```

6.1.3.4 Avertissement

Il n'y a pas d'ordre de priorité par rapport à l'opérateur **&&**. Il pour cela isoler les expressions à l'aide de l'opérateur **( )**.

6.1.3.5 Voir aussi

&& pour réaliser une conjonction logique.

6.1.4 &&

6.1.4.1 Prototypes

Expression **&&** *Expression*

6.1.4.2 Description

Disjonction inclusive de deux expressions :

- Le résultat est ***Vrai*** si les deux expressions valent ***Vrai***.
- Le résultat est ***Faux*** si l'une des deux expressions vaut ***Faux***.

6.1.4.3 Exemple

```
/*************************************************/
#if (defined(AIX43)||defined(SCO32))&&!defined(LINUX20)
/*************************************************/
...
```

```
/****/
#endif
/****/
```

6.1.4.4 Avertissement

Il n'y a pas d'ordre de priorité par rapport à l'opérateur **||**. Il pour cela isoler les expressions à l'aide de l'opérateur **( )**.

6.1.4.5 Voir aussi

|| pour réaliser une disjonction logique inclusive.

6.2 Instructions du pré processeur

6.2.1 define

6.2.1.1 Prototypes

define ***NomDuSymbole*** *ParametresOptionnels* ***DefinitionDuSymbole***

6.2.1.2 Description

S'il n'y a pas de paramètres, cette instruction définit une constante portant le nom ***NomDuSymbole*** et de valeur ***DefinitionDuSymbole***.

S'il y a au moins un paramètre, cette instruction définit une macro-fonction portant le nom ***NomDuSymbole*** et de corps ***DefinitionDuSymbole***. A chaque fois que cette macro-fonction est utilisée, ses paramètres sont substitués.

La définition du symbole est constituée de l'ensemble des caractères jusqu'à la fin de la ligne à partir du premier caractère qui n'est pas un séparateur.

6.2.1.3 Exemple

```
#define CO_TailleBuffer 256
#define _T(X) (wchar_t *)(X)
...
A=CO_TailleBuffer;
B=_T("coucou");
```

6.2.1.4 Avertissement

Pour écrire une définition sur plusieurs lignes, il suffit d'ajouter le caractère **division inversée ** en fin des premières lignes composant la définition de la constante ou de la macro-fonction.

La définition est valable tant qu'elle n'est pas supprimée par l'instruction **undef**.

6.2.1.5 Voir aussi

undef pour supprimer une définition.

6.2.2 defined

6.2.2.1 Prototypes

defined (*NomDuSymbole*)

6.2.2.2 Description

Retourne ***Vrai*** si le symbole est défini sous forme d'une constante ou d'une macro-fonction.

6.2.2.3 Exemple

```
#if defined(CO_TailleBuffer)
...
#endif
```

6.2.2.4 Avertissement

Néant.

6.2.2.5 Voir aussi

`define` pour définir un symbole de constante ou de macro-fonction.

6.2.3 elif

6.2.3.1 Prototypes

`elif` *Expression*

6.2.3.2 Description

Continue une analyse conditionnelle selon la convention suivante :

- Si une des expressions des instructions **`if`** et **`elif`** précédentes du même bloc d'analyse conditionnelle a est évaluée à ***Vrai***, alors le contenu du fichier est ignoré jusqu'à l'instruction **`elif`** ou **`endif`** correspondant à l'instruction **`elif`**.
- Sinon, si l'expression est évaluée à ***Vrai***, alors le contenu du fichier est analysé jusqu'à l'instruction **`elif`**, **`else`** ou **`endif`** correspondant à l'instruction **`elif`**.
- Sinon, le contenu du fichier est ignoré jusqu'à l'instruction **`elif`**, **`else`** ou **`endif`** correspondant à l'instruction **`elif`**.

6.2.3.3 Exemple

```
/******************/
#if defined(LINUX20)
/******************/
#include <wchar.h>
#define _T(X) (wchar_t *)(X)
/******************/
#elif defined(OS400)
/******************/
#include <sys/tchar.h>
/****/
#else
/****/
#include <tchar.h>
/****/
#endif
/****/
```

6.2.3.4 Avertissement

L'instruction **elif** ne peut être employée qu'entre des instructions **if** et **endif**.

6.2.3.5 Voir aussi

if pour démarrer une analyse conditionnelle.

else pour continuer une analyse conditionnelle.

endif pour terminer une analyse conditionnelle.

6.2.4 else

6.2.4.1 Prototypes

```
else
```

6.2.4.2 Description

Continue une analyse conditionnelle selon la convention suivante :

- Si une des expressions des instructions **if** et **elif** précédentes du même bloc d'analyse conditionnelle a est évaluée à ***Vrai***, alors le contenu du fichier est ignoré jusqu'à l'instruction **endif** correspondant à l'instruction **else**.
- Sinon, le contenu du fichier est analysé jusqu'à l'instruction **endif** correspondant à l'instruction **else**.

6.2.4.3 Exemple

```
/******************/
#if defined(LINUX20)
/******************/
#include <wchar.h>
#define _T(X) (wchar_t *)(X)
/******************/
#elif defined(OS400)
/******************/
#include <sys/tchar.h>
/****/
#else
/****/
#include <tchar.h>
/****/
#endif
/****/
```

6.2.4.4 Avertissement

L'instruction **else** ne peut être employée qu'entre des instructions **if** et **endif**.

6.2.4.5 Voir aussi

if pour démarrer une analyse conditionnelle.

elif pour continuer une analyse conditionnelle.

endif pour terminer une analyse conditionnelle.

6.2.5 endif

6.2.5.1 Prototypes

```
endif
```

6.2.5.2 Description

Termine une analyse conditionnelle.

6.2.5.3 Exemple

```
/******************/
#if defined(LINUX20)
/******************/
#include <wchar.h>
#define _T(X) (wchar_t *)(X)
/******************/
#elif defined(OS400)
/******************/
#include <sys/tchar.h>
/****/
#else
/****/
#include <tchar.h>
/****/
#endif
/****/
```

6.2.5.4 Avertissement

Les instructions **if** et **endif** fonctionnent par paire comme des parenthèses.

6.2.5.5 Voir aussi

if pour démarrer une analyse conditionnelle.

elif et **else** pour continuer une analyse conditionnelle.

6.2.6 if

6.2.6.1 Prototypes

if *Expression*

6.2.6.2 Description

Réalise une analyse conditionnelle selon la convention suivante :

- Si l'expression est évaluée à ***Vrai***, alors le contenu du fichier est analysé jusqu'à l'instruction **elif**, **else** ou **endif** correspondant à l'instruction **if**.
- Sinon, le contenu du fichier est ignoré jusqu'à l'instruction **elif**, **else** ou **endif** correspondant à l'instruction **if**.

6.2.6.3 Exemple

```
/******************/
#if defined(LINUX20)
```

```
/******************/
#include <wchar.h>
#define _T(X) (wchar_t *)(X)
/******************/
#elif defined(OS400)
/******************/
#include <sys/tchar.h>
/****/
#else
/****/
#include <tchar.h>
/****/
#endif
/****/
```

6.2.6.4 Avertissement

Les instructions **if** et **endif** fonctionnent par paire comme des parenthèses.

6.2.6.5 Voir aussi

elif et **else** pour continuer une analyse conditionnelle.

endif pour terminer une analyse conditionnelle.

6.2.7 include

6.2.7.1 Prototypes

include ***NomDuFichier***

6.2.7.2 Description

Inclut à la place de la ligne courante le contenu du fichier dont le nom est donné la chaîne de caractères ***NomDuFichier***. Celui-ci peut être exprimé :

- Entre caractères **guillemets "**.
  En ce cas, le fichier est cherché :
  - Dans le répertoire désigné si ***NomDuFichier*** contient un chemin d'accès absolu.
  - En relatif par rapport au répertoire courant si ***NomDuFichier*** contient un chemin d'accès relatif.
- Entre caractères **chevrons <** et **>**.
  En ce cas, le fichier est cherché :
  - Dans le répertoire désigné si ***NomDuFichier*** contient un chemin d'accès absolu.
  - En relatif par rapport au chemin des inclusions si ***NomDuFichier*** contient un chemin d'accès relatif.

6.2.7.3 Exemple

```
#include <stdio.h>
#include "C:\\tmp\\essai.h"
```

6.2.7.4 Avertissement

Le nom du fichier courant et le compteur de lignes sont changés le temps de l'inclusion du fichier avec deux instructions **`line`** :

- Une nouvelle ligne est alors ajoutée avant l'inclusion pour changer le nom et remettre le compteur de lignes.
- Une nouvelle ligne est alors ajoutée après l'inclusion pour remettre le nom et remettre le compteur de lignes.

6.2.7.5 Voir aussi

`line` pour changer le nom du fichier et le compteur de lignes.

6.2.8 line

6.2.8.1 Prototypes

`line` ***NumeroDeLigne NomDuFichier***

6.2.8.2 Description

Change la valeur du nom du fichier et du compteur de lignes en prenant respectivement la valeur de la chaîne de caractères ***NomDuFichier*** et la valeur entière ***NumeroDeLigne***.

6.2.8.3 Exemple

```
#line 10 "C:\\tmp\\essai.cpp"
```

6.2.8.4 Avertissement

Des instructions **`ligne`** sont automatiquement ajoutées de part et d'autre d'une instruction **`include`** par le pré processeur.

6.2.8.5 Voir aussi

Néant.

6.2.9 pragma

6.2.9.1 Prototypes

`pragma` ***Directive***

6.2.9.2 Description

Passe une directive au compilateur.

La définition de la directive est constituée de l'ensemble des caractères jusqu'à la fin de la ligne à partir du premier caractère qui n'est pas un séparateur.

6.2.9.3 Exemple

```
#pragma warning(pop)
#pragma warning(push,4)
#pragma warning(disable : 4055 4065 4115 4121 4190 4201 4213 4250)
```

6.2.9.4 Avertissement

La directive est ignorée en ***C--***. Cette instruction est uniquement conservée pour assurer la compatibilité avec les compilateurs ***C / C++***.

6.2.9.5 Voir aussi

Néant.

6.2.10 undef

6.2.10.1 Prototypes

```
undef NomDuSymbole
```

6.2.10.2 Description

Suppression de la définition du symbole identifié par ***NomDuSymbole***.

Il peut s'agit soit d'une constante, soit d'une macro-fonction.

6.2.10.3 Exemple

```
undef CO_TailleBuffer
undef _T
```

6.2.10.4 Avertissement

Néant.

6.2.10.5 Voir aussi

`defined` pour tester l'existence d'une définition.

7 Référence de C--

7.1 Opérateurs du langage C--

7.1.1 ( )

7.1.1.1 Prototypes

(*Expression* **)**

(***NomDUnType*** *PointeurOption* **)** *Expression*

NomDUneFonction **(** *ListeDExpressionsOption* **)**

7.1.1.2 Description

Opérateur d'isolation d'une expression. Le type du résultat est le type de l'expression isolée.

Opérateur de conversion de type. Le type du résultat est du type spécifié dans la conversion.

Opérateur d'appel de fonction. Le type du résultat est le type de la fonction.

7.1.1.3 Exemple

```
TypUpsVmShort F(TypUpsVmShort L);

TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmDouble K;

I=10;
J=2*(I+2);
K=(TypUpsVmDouble)J;
I=F(J);
```

7.1.1.4 Avertissement

M La conversion des types doit être compatible. Elle est utilisée pour :

- Etendre la taille d'un entier.
- Diminuer la taille d'un entier.
- Changer le type de pointeur de données.
- Changer le type de pointeur de traitements.

Il n'est pas possible de :

- Convertir un entier en pointeur.
- Convertir un pointeur de données en un pointeur de traitement.
- Convertir un pointeur de traitements en un pointeur de données.

M Le prototype de la fonction doit être respecté :

- Même nombre de paramètres.
- Même type entre les valeurs passées et les paramètres attendus.

7.1.1.5 Voir aussi

Néant.

7.1.2 .

7.1.2.1 Prototypes

. ***NomDUnChamp*** *SuiteIdentificateur*

7.1.2.2 Description

Opérateur de sélection d'un champ. Le type du résultat est le type du champ sélectionné.

7.1.2.3 Exemple

```
typedef struct matruct
{
TypUpsVmShort A;
TypUpsVmShort B;
} MaStruct;

MaStruct I;

I.A=10;
I.B=20;
```

7.1.2.4 Avertissement

L'expression précédant cet opérateur doit être de type structure ou union.

7.1.2.5 Voir aussi

-> pour interpréter une adresse et sélectionner un champ.

7.1.3 ->

7.1.3.1 Prototypes

-> ***NomDUnChamp*** *SuiteIdentificateur*

7.1.3.2 Description

Opérateur d'interprétation d'une adresse et de sélection d'un champ. Le type du résultat est le type du champ sélectionné.

7.1.3.3 Exemple

```
typedef struct matruct
{
TypUpsVmShort A;
TypUpsVmShort B;
} MaStruct;

MaStruct *I;

I->A=10;
I->B=20;
```

7.1.3.4 Avertissement

L'expression précédant cet opérateur doit être de type pointeur sur une structure ou sur une union.

7.1.3.5 Voir aussi

. pour sélectionner un champ.

7.1.4 [ ]

7.1.4.1 Prototypes

[*Expression* **]** *SuiteIdentificateur*

7.1.4.2 Description

Opérateur de sélection d'un élément dans un tableau. Le type du résultat est le type de l'élément du tableau.

7.1.4.3 Exemple

```
TypUpsVmShort I[10];

I[0]=10;
I[1]=20;
```

7.1.4.4 Avertissement

L'index du premier élément du tableau est ***0***.

7.1.4.5 Voir aussi

Néant.

7.1.5 +

7.1.5.1 Prototypes

Expression **+** *Expression*

7.1.5.2 Description

Opérateur binaire d'addition deux des expressions de type entier ou de type réel. Si l'une des deux expressions est de type réel, le résultat est de type réel, sinon le résultat est de type entier.

7.1.5.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmDouble K;

I=10;
J=I+2;
K=I+2.0;
K=3.0+I;
```

7.1.5.4 Avertissement

Pour une opération sur des nombres entiers, la taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.5.5 Voir aussi

- pour réaliser une soustraction.

***** pour réaliser une multiplication.

/ pour réaliser une division.

% pour réaliser le reste d'une division.

7.1.6 ++

7.1.6.1 Prototypes

++ *Expression*

Expression **++**

7.1.6.2 Description

Opérateur unaire d'incrémentation d'une expression de type entier ou de type réel. Le type du résultat est le type de l'opérande.

L'expression doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Si l'opérateur est préfixé, le résultat est la valeur après être incrémentée. Si l'opérateur est post fixé, le résultat est la valeur avant d'être incrémentée.

7.1.6.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
J=I++;
J=++I;
```

7.1.6.4 Avertissement

Pour une opération sur des nombres entiers, la taille de stockage du résultat en terme de nombre d'octets est la taille de l'opérande.

7.1.6.5 Voir aussi

-- pour réaliser une décrémentation.

7.1.7 -

7.1.7.1 Prototypes

- *Expression*

Expression **-** *Expression*

7.1.7.2 Description

Opérateur unaire d'inversion de signe pour une expression de type entier ou de type réel.

Opérateur binaire de soustraction entre deux expressions de type entier ou de type réel. Si l'une des deux expressions est de type réel, le résultat est de type réel, sinon le résultat est de type entier.

7.1.7.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmDouble K;

I=10;
J=-I;
J=I-2;
K=I-2.0;
K=3.0-I;
```

7.1.7.4 Avertissement

Pour une opération sur des nombres entiers, la taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.7.5 Voir aussi

+ pour réaliser une addition.

***** pour réaliser une multiplication.

/ pour réaliser une division.

% pour réaliser le reste d'une division.

7.1.8 --

7.1.8.1 Prototypes

-- *Expression*

Expression **--**

7.1.8.2 Description

Opérateur unaire de décrémentation d'une expression de type entier ou de type réel. Le type du résultat est le type de l'opérande.

L'expression doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.

- Un champ d'une structure.
- Un champ d'une union.

Si l'opérateur est préfixé, le résultat est la valeur après être incrémentée. Si l'opérateur est post fixé, le résultat est la valeur avant d'être incrémentée.

7.1.8.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
J=I--;
J=--I;
```

7.1.8.4 Avertissement

Pour une opération sur des nombres entiers, la taille de stockage du résultat en terme de nombre d'octets est la taille de l'opérande.

7.1.8.5 Voir aussi

++ pour réaliser une incrémentation.

7.1.9 *

7.1.9.1 Prototypes

Expression ***** *Expression*

7.1.9.2 Description

Opérateur binaire de multiplication entre deux expressions de type entier ou de type réel. Si l'une des deux expressions est de type réel, le résultat est de type réel, sinon le résultat est de type entier.

Opérateur unaire d'interprétation de l'adresse d'une expression. Le résultat est du type de l'expression pointée. L'expression doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

7.1.9.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmDouble K;
TypUpsVmShort *L;

I=10;
J=I*2;
K=I*2.0;
K=3.0*I;
```

```
L=&I;
*L=1;
```

7.1.9.4 Avertissement

Pour une opération sur des nombres entiers, la taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.9.5 Voir aussi

+ pour réaliser une addition.

- pour réaliser une soustraction.

/ pour réaliser une division.

% pour réaliser le reste d'une division.

& pour lire l'adresse d'une expression.

7.1.10 /

7.1.10.1 Prototypes

Expression **/** *Expression*

7.1.10.2 Description

Opérateur binaire de division entre deux expressions de type entier ou de type réel. Si l'une des deux expressions est de type réel, le résultat est de type réel, sinon le résultat est de type entier.

7.1.10.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmDouble K;

I=10;
J=I/2;
K=I/2.0;
K=3.0/(I+1);
```

7.1.10.4 Avertissement

Pour une opération sur des nombres entiers, la taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes. De plus, seule la partie entière du résultat de la division est conservée.

7.1.10.5 Voir aussi

+ pour réaliser une addition.

- pour réaliser une soustraction.

***** pour réaliser une multiplication.

% pour réaliser le reste d'une division.

7.1.11 %

7.1.11.1 Prototypes

Expression **%** *Expression*

7.1.11.2 Description

Opérateur binaire de reste de la division entière entre deux expressions de type entier ou de type réel. Le résultat est de type entier même si l'une des deux expressions est de type réel.

7.1.11.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmDouble K;

I=10;
J=I%2;
K=I%2.0;
K=3.0%(I+1);
```

7.1.11.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.11.5 Voir aussi

+ pour réaliser une addition.

- pour réaliser une soustraction.

***** pour réaliser une multiplication.

/ pour réaliser une division.

7.1.12 &

7.1.12.1 Prototypes

Expression **&** *Expression*

& *Expression*

7.1.12.2 Description

Opérateur binaire de conjonction bit à bit entre deux expressions de type entier. Le résultat est de type entier.

- Un bit vaut ***1*** si ce même bit vaut ***1*** dans les deux valeurs.
- Un bit vaut ***0*** si ce même bit vaut ***0*** dans l'une des deux valeurs.

Opérateur unaire de lecture de l'adresse d'une expression. Le résultat est de type pointeur. L'expression doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.

- Un champ d'une structure.
- Un champ d'une union.

7.1.12.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmShort *K;

I=10;
J=I&2;
K=&J;
```

7.1.12.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.12.5 Voir aussi

| pour réaliser une disjonction inclusive bit à bit.

^ pour réaliser une disjonction exclusive bit à bit.

***** pour réaliser l'interprétation d'un pointeur.

7.1.13 |

7.1.13.1 Prototypes

Expression **|** *Expression*

7.1.13.2 Description

Opérateur binaire de disjonction inclusive bit à bit entre deux expressions de type entier. Le résultat est de type entier.

- Un bit vaut ***1*** si ce même bit vaut ***1*** dans l'une des deux valeurs.
- Un bit vaut ***0*** si ce même bit vaut ***0*** dans les deux valeurs.

7.1.13.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
J=I|2;
```

7.1.13.4 Avertissement

Pour une opération sur des nombres entiers, la taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.13.5 Voir aussi

& pour réaliser une conjonction bit à bit.

^ pour réaliser une disjonction exclusive bit à bit.

7.1.14 ^

7.1.14.1 Prototypes

Expression **^** *Expression*

7.1.14.2 Description

Opérateur binaire de disjonction exclusive bit à bit entre deux expressions de type entier. Le résultat est de type entier.

- Un bit vaut ***1*** si ce même bit vaut ***1*** dans l'une deux valeurs et ***0*** dans l'autre.
- Un bit vaut ***0*** si ce même bit vaut ***0*** dans les deux valeurs ou ***1*** dans les deux valeurs.

7.1.14.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
J=I^2;
```

7.1.14.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.14.5 Voir aussi

& pour réaliser une conjonction bit à bit.

| pour réaliser une disjonction inclusive bit à bit.

7.1.15 ~

7.1.15.1 Prototypes

Expression **~** *Expression*

7.1.15.2 Description

Opérateur unaire de négation bit à bit entre deux expressions de type entier. Le résultat est de type entier.

- Un bit vaut ***1*** si ce même bit vaut ***0*** dans la valeur.
- Un bit vaut ***0*** si ce même bit vaut ***1*** dans la valeur.

7.1.15.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
J=~I;
```

7.1.15.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la taille de l'opérande.

7.1.15.5 Voir aussi

& pour réaliser une conjonction bit à bit.

| pour réaliser une disjonction inclusive bit à bit.

^ pour réaliser une disjonction inclusive bit à bit.

7.1.16 &&

7.1.16.1 Prototypes

Expression **&&** *Expression*

7.1.16.2 Description

Opérateur binaire de conjonction logique entre deux expressions de type entier. Le résultat est de type entier.

- Le résultat est ***1*** si les deux expressions ne valent pas ***0***.
- Le résultat est ***0*** si l'une des deux expressions vaut ***0***.

7.1.16.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
J=I&&2;
```

7.1.16.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.16.5 Voir aussi

|| pour réaliser une disjonction logique inclusive.

7.1.17 ||

7.1.17.1 Prototypes

Expression **||** *Expression*

7.1.17.2 Description

Opérateur binaire de conjonction logique entre deux expressions de type entier. Le résultat est de type entier.

- Le résultat est ***1*** si l'une des deux expressions ne vaut pas ***0***.
- Le résultat est ***0*** si les deux expressions valent ***0***.

7.1.17.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
```

```
J=I||2;
```

7.1.17.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.17.5 Voir aussi

&& pour réaliser une conjonction logique.

7.1.18 !

7.1.18.1 Prototypes

! *Expression*

7.1.18.2 Description

Opérateur unaire de négation logique d'une expression de type entier. Le résultat est de type entier.

- Le résultat vaut ***1*** si la valeur vaut ***0***.
- Le résultat vaut ***0*** si la valeur ne vaut pas ***0***.

7.1.18.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
J=!I;
```

7.1.18.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la taille de l'opérande.

7.1.18.5 Voir aussi

&& pour réaliser une conjonction logique.

|| pour réaliser une disjonction inclusive logique.

7.1.19 <<

7.1.19.1 Prototypes

Expression **<<** *Expression*

7.1.19.2 Description

Opérateur binaire de décalage des bits vers la gauche entre deux expressions de type entier. Le résultat est de type entier.

La valeur de l'expression de gauche est décalée d'autant de bits que le spécifie la valeur de l'expression de droite. Les bits insérés valent ***0***. Cela équivaut à autant de multiplication par deux.

7.1.19.3 Exemple

```
TypUpsVmShort I;
```

```
TypUpsVmShort J;

I=10;
J=I<<2;
```

7.1.19.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.19.5 Voir aussi

>> pour réaliser décalage de bit vers la droite.

7.1.20 >>

7.1.20.1 Prototypes

Expression **>>** *Expression*

7.1.20.2 Description

Opérateur binaire de décalage des bits vers la droite entre des expressions de type entier. Le résultat est de type entier.

La valeur de l'expression de gauche est décalée d'autant de bits que le spécifie la valeur de l'expression de droite. Les bits insérés valent ***0***. Cela équivaut à autant de divisions par deux.

7.1.20.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;

I=10;
J=I>>2;
```

7.1.20.4 Avertissement

La taille de stockage du résultat en terme de nombre d'octets est la plus grande taille des opérandes.

7.1.20.5 Voir aussi

<< pour réaliser décalage de bit vers la droite.

7.1.21 ==

7.1.21.1 Prototypes

Expression **==** *Expression*

7.1.21.2 Description

Opérateur binaire de comparaison entre deux expressions de type entier ou de type réel. Le résultat est de type entier.

Opérateur binaire de comparaison entre des expressions de type pointeur. Le résultat est de type entier.

- Le résultat est ***1*** si les valeurs des deux expressions sont identiques.

- Le résultat est ***0*** si les valeurs des deux expressions ne sont pas identiques.

7.1.21.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmShort K;

I=10;
J=2;
K=(I==J);
```

7.1.21.4 Avertissement

La taille de stockage du résultat celle de ***TypUpsVmShort***.

Une adresse est considérée comme un nombre entier de type ***TypUpsVmUnsignedLong***.

7.1.21.5 Voir aussi

!=, **<**, **>**, **<=**, **>=** pour réaliser une autre sorte de comparaison.

7.1.22 !=

7.1.22.1 Prototypes

Expression **!=** *Expression*

7.1.22.2 Description

Opérateur binaire de comparaison entre deux expressions de type entier ou de type réel. Le résultat est de type entier.

Opérateur binaire de comparaison entre des expressions de type pointeur. Le résultat est de type entier.

- Le résultat est ***1*** si les valeurs des deux expressions ne sont pas identiques.
- Le résultat est ***0*** si les valeurs des deux expressions sont identiques.

7.1.22.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmShort K;

I=10;
J=2;
K=(I!=J);
```

7.1.22.4 Avertissement

La taille de stockage du résultat celle de ***TypUpsVmShort***.

Une adresse est considérée comme un nombre entier de type ***TypUpsVmUnsignedLong***.

7.1.22.5 Voir aussi

==, **<**, **>**, **<=**, **>=** pour réaliser une autre sorte de comparaison.

7.1.23 <

7.1.23.1 Prototypes

Expression **<** *Expression*

7.1.23.2 Description

Opérateur binaire de comparaison entre deux expressions de type entier ou de type réel. Le résultat est de type entier.

Opérateur binaire de comparaison entre deux expressions de type pointeur. Le résultat est de type entier.

- Le résultat est ***1*** si la valeur de l'expression de gauche est strictement inférieure à la valeur de l'expression de droite.
- Le résultat est ***0*** si la valeur de l'expression de gauche est identique ou supérieure à la valeur de l'expression de droite.

7.1.23.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmShort K;

I=10;
J=2;
K=(I<J);
```

7.1.23.4 Avertissement

La taille de stockage du résultat celle de ***TypUpsVmShort***.

Une adresse est considérée comme un nombre entier de type ***TypUpsVmUnsignedLong***.

7.1.23.5 Voir aussi

==, **!=**, **>**, **<=**, **>=** pour réaliser une autre sorte de comparaison.

7.1.24 >

7.1.24.1 Prototypes

Expression **>** *Expression*

7.1.24.2 Description

Opérateur binaire de comparaison entre deux expressions de type entier ou de type réel. Le résultat est de type entier.

Opérateur binaire de comparaison entre deux expressions de type pointeur. Le résultat est de type entier.

- Le résultat est ***1*** si la valeur de l'expression de gauche est strictement supérieure à la valeur de l'expression de droite.
- Le résultat est ***0*** si la valeur de l'expression de gauche est identique ou inférieure à la valeur de l'expression de droite.

7.1.24.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmShort K;

I=10;
J=2;
K=(I<J);
```

7.1.24.4 Avertissement

La taille de stockage du résultat celle de ***TypUpsVmShort***.

Une adresse est considérée comme un nombre entier de type ***TypUpsVmUnsignedLong***.

7.1.24.5 Voir aussi

==, **!=**, **<**, **<=**, **>=** pour réaliser une autre sorte de comparaison.

7.1.25 <=

7.1.25.1 Prototypes

Expression **<=** *Expression*

7.1.25.2 Description

Opérateur binaire de comparaison entre deux expressions de type entier ou de type réel. Le résultat est de type entier.

Opérateur binaire de comparaison entre deux expressions de type pointeur. Le résultat est de type entier.

- Le résultat est ***1*** si la valeur de l'expression de gauche est identique ou inférieure à la valeur de l'expression de droite.
- Le résultat est ***0*** si la valeur de l'expression de gauche est strictement supérieure à la valeur de l'expression de droite.

7.1.25.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmShort K;

I=10;
J=2;
K=(I<=J);
```

7.1.25.4 Avertissement

La taille de stockage du résultat celle de ***TypUpsVmShort***.

Une adresse est considérée comme un nombre entier de type ***TypUpsVmUnsignedLong***.

7.1.25.5 Voir aussi

==, **!=**, **<**, **>**, **>=** pour réaliser une autre sorte de comparaison.

7.1.26 >=

7.1.26.1 Prototypes

Expression **>=** *Expression*

7.1.26.2 Description

Opérateur binaire de comparaison entre deux expressions de type entier ou de type réel. Le résultat est de type entier.

Opérateur binaire de comparaison entre deux expressions de type pointeur. Le résultat est de type entier.

- Le résultat est ***1*** si la valeur de l'expression de gauche est identique ou supérieure à la valeur de l'expression de droite.
- Le résultat est ***0*** si la valeur de l'expression de gauche est strictement inférieure à la valeur de l'expression de droite.

7.1.26.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmShort K;

I=10;
J=2;
K=(I>=J);
```

7.1.26.4 Avertissement

La taille de stockage du résultat celle de ***TypUpsVmShort***.

Une adresse est considérée comme un nombre entier de type ***TypUpsVmUnsignedLong***.

7.1.26.5 Voir aussi

==, **!=**, **<**, **>**, **<=** pour réaliser une autre sorte de comparaison.

7.1.27 =

7.1.27.1 Prototypes

Expression **=** *Expression*

7.1.27.2 Description

Opérateur binaire d'affectation entre deux expressions. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

7.1.27.3 Exemple

```
TypUpsVmShort I;
TypUpsVmShort J;
TypUpsVmShort K;

I=10;
J=2;
K=I;
```

7.1.27.4 Avertissement

Les types de données doivent être identiques, sinon il faut utiliser l'opérateur de conversion **( )**.

7.1.27.5 Voir aussi

+=, **-=**, ***=**, **/***, **%=**, **<<=**, **>>=**, **&=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

7.1.28 +=

7.1.28.1 Prototypes

Expression **+=** *Expression*

7.1.28.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une addition. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Une addition est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur d'addition.

7.1.28.3 Exemple

```
TypUpsVmShort I;

I=10;
I+=2;
```

7.1.28.4 Avertissement

Néant.

7.1.28.5 Voir aussi

=, **-=**, ***=**, **/***, **%=**, **<<=**, **>>=**, **&=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

+ pour réaliser une addition.

7.1.29 -=

7.1.29.1 Prototypes

Expression **-=** *Expression*

7.1.29.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une soustraction. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Une soustraction est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de soustraction.

7.1.29.3 Exemple

```
TypUpsVmShort I;

I=10;
I-=2;
```

7.1.29.4 Avertissement

Néant.

7.1.29.5 Voir aussi

=, **+=**, ***=**, **/***, **%=**, **<<=**, **>>=**, **&=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

- pour réaliser une soustraction.

7.1.30 *=

7.1.30.1 Prototypes

Expression ***=** *Expression*

7.1.30.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une multiplication. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.

- Un champ d'une union.

Une multiplication est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de multiplication.

7.1.30.3 Exemple

```
TypUpsVmShort I;

I=10;
I*=2;
```

7.1.30.4 Avertissement

Néant.

7.1.30.5 Voir aussi

=, **+=**, **-=**, **/=**, **%=**, **<<=**, **>>=**, **&=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

***** pour réaliser une multiplication.

7.1.31 /=

7.1.31.1 Prototypes

Expression /= Expression

7.1.31.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une division. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Une division est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de division.

7.1.31.3 Exemple

```
TypUpsVmShort I;

I=10;
I/=2;
```

7.1.31.4 Avertissement

Néant.

7.1.31.5 Voir aussi

=, **+=**, **-=**, ***=**, **%=**, **<<=**, **>>=**, **&=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

/ pour réaliser une division.

7.1.32 %=

7.1.32.1 Prototypes

Expression %= Expression

7.1.32.2 Description

Opérateur binaire d'affectation entre deux expressions combinant un reste de division. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Le reste de la division entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur modulo.

7.1.32.3 Exemple

```
TypUpsVmShort I;

I=10;
I%=2;
```

7.1.32.4 Avertissement

Néant.

7.1.32.5 Voir aussi

=, **+=**, **-=**, ***=**, **/=**, **<<=**, **>>=**, **&=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

% pour réaliser un reste de division.

7.1.33 <<=

7.1.33.1 Prototypes

Expression <<= Expression

7.1.33.2 Description

Opérateur binaire d'affectation entre deux expressions combinant un décalage à gauche des bits. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.

- Un champ d'une structure.
- Un champ d'une union.

Un décalage à gauche des bits est réalisé entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de décalage à gauche des bits.

7.1.33.3 Exemple

```
TypUpsVmShort I;

I=10;
I<<=2;
```

7.1.33.4 Avertissement

Néant.

7.1.33.5 Voir aussi

=, **+=**, **-=**, ***=**, **/=**, **%=**, **>>=**, **&=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

<< pour réaliser un décalage à gauche des bits.

7.1.34 >>=

7.1.34.1 Prototypes

Expression **>>=** *Expression*

7.1.34.2 Description

Opérateur binaire d'affectation entre deux expressions combinant un décalage à droite des bits. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Un décalage à droite des bits est réalisé entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de décalage à droite des bits.

7.1.34.3 Exemple

```
TypUpsVmShort I;

I=10;
I>>=2;
```

7.1.34.4 Avertissement

Néant.

7.1.34.5 Voir aussi

=, **+=**, **-=**, ***=**, **/=**, **%=**, **<<=**, **&=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

>> pour réaliser un décalage à droite des bits.

7.1.35 &=

7.1.35.1 Prototypes

Expression **&=** *Expression*

7.1.35.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une conjonction bit à bit. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Une conjonction bit à bit est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de conjonction bit à bit.

7.1.35.3 Exemple

```
TypUpsVmShort I;

I=10;
I&=2;
```

7.1.35.4 Avertissement

Néant.

7.1.35.5 Voir aussi

=, **+=**, **-=**, ***=**, **/=**, **%=**, **<<=**, **>>=**, **|=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

& pour réaliser une conjonction bit à bit.

7.1.36 |=

7.1.36.1 Prototypes

Expression **|=** *Expression*

7.1.36.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une disjonction inclusive bit à bit. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.

- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Une disjonction inclusive bit à bit est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de disjonction inclusive bit à bit.

7.1.36.3 Exemple

```
TypUpsVmShort I;

I=10;
I|=2;
```

7.1.36.4 Avertissement

Néant.

7.1.36.5 Voir aussi

=, **+=**, **-=**, ***=**, **/=**, **%=**, **<<=**, **>>=**, **&=**, **^=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

| pour réaliser une disjonction inclusive bit à bit.

7.1.37 ^=

7.1.37.1 Prototypes

Expression **^=** *Expression*

7.1.37.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une disjonction exclusive bit à bit. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Une disjonction exclusive bit à bit est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de disjonction exclusive bit à bit.

7.1.37.3 Exemple

```
TypUpsVmShort I;

I=10;
I^=2;
```

7.1.37.4 Avertissement

Néant.

7.1.37.5 Voir aussi

=, **+=**, **-=**, ***=**, **/=**, **%=**, **<<=**, **>>=**, **&=**, **|=**, **&&=**, **||=** pour réaliser une autre sorte d'affectation.

^ pour réaliser une disjonction exclusive bit à bit.

7.1.38 &&=

7.1.38.1 Prototypes

Expression **&&=** *Expression*

7.1.38.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une conjonction logique. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Une conjonction logique est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de conjonction logique.

7.1.38.3 Exemple

```
TypUpsVmShort I;

I=10;
I&&=2;
```

7.1.38.4 Avertissement

Néant.

7.1.38.5 Voir aussi

=, **+=**, **-=**, ***=**, **/=**, **%=**, **<<=**, **>>=**, **&=**, **|=**, **^=**, **||=** pour réaliser une autre sorte d'affectation.

&& pour réaliser une conjonction logique.

7.1.39 ||=

7.1.39.1 Prototypes

Expression **||=** *Expression*

7.1.39.2 Description

Opérateur binaire d'affectation entre deux expressions combinant une disjonction inclusive logique. Il n'y a pas de résultat.

L'expression de gauche doit être :

- Une variable globale.
- Une variable locale.
- Un élément d'un tableau.
- Un champ d'une structure.
- Un champ d'une union.

Une disjonction inclusive logique est réalisée entre la valeur de l'expression de gauche et la valeur de l'expression de droite conformément à la sémantique de l'opérateur de disjonction inclusive logique.

7.1.39.3 Exemple

```
TypUpsVmShort I;

I=10;
I||=2;
```

7.1.39.4 Avertissement

Néant.

7.1.39.5 Voir aussi

=, **+=**, **-=**, ***=**, **/=**, **%=**, **<<=**, **>>=**, **&=**, **|=**, **^=**, **&&=** pour réaliser une autre sorte d'affectation.

|| pour réaliser une disjonction inclusive logique.

7.2 Instructions du langage C--

7.2.1 break

7.2.1.1 Prototypes

```
break
```

7.2.1.2 Description

Interromps l'exécution d'une liste d'instructions et continue à l'instruction suivante du bloc.

Cette instruction **break** est notamment employée pour sortir d'une instruction **for**, **while**, **do** ou **case**.

7.2.1.3 Exemple

```
for (...)
   {
   ...
   break;
   ...
   }

while (...)
   {
   ...
   break;
   ...
   }

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

7.2.1.4 Avertissement

L'instruction **break** ne peut pas être employée en dehors d'une liste d'instructions.

Les instructions qui suivent l'instruction **break** ne sont pas exécutées.

7.2.1.5 Voir aussi

for, **while**, **do** pour définir une boucle.

switch pour définir un éclatement.

7.2.2 case

7.2.2.1 Prototypes

case ***ConstanteEntiere*** **:** *Instructions*

7.2.2.2 Description

Exécute un test de la manière suivante :

1) Exécute l'expression de test.
2) Compare le résultat de l'expression à chaque cas de test.
3) Au premier cas de test dont la valeur correspond, exécute la liste des instructions du cas.
4) Continue sur les instructions du cas suivant.

7.2.2.3 Exemple

```
TypUpsVmShort I;

switch (I)
   {
   case 0 :
      ...
      break;

   case 1 :
      ...
      break;

   default :
      ...
      break;
   }
```

7.2.2.4 Avertissement

M L'expression doit être de type entier.

7.2.2.5 Voir aussi

switch pour définir un éclatement.

default pour définir le cas par défaut.

break pour sortir d'un cas de test.

7.2.3 continue

7.2.3.1 Prototypes

```
continue
```

7.2.3.2 Description

Interromps l'exécution d'une liste d'instructions d'une boucle et continue à l'itération suivante.

Cette instruction **continue** est employée dans le corps d'une instruction **for**, **while** ou **do**.

7.2.3.3 Exemple

```
for (...)
   {
   ...
   continue;
```

```
   ...
   }

while (...)
   {
   ...
   continue;
   ...
   }

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

7.2.3.4 Avertissement

L'instruction **continue** ne peut pas être employée en dehors d'une liste d'instructions d'une boucle.

Les instructions qui suivent l'instruction **continue** ne sont pas exécutées.

7.2.3.5 Voir aussi

for, **while**, **do** pour définir une boucle.

7.2.4 default

7.2.4.1 Prototypes

default :

7.2.4.2 Description

Alternative exécutée si aucun cas de test précédent ne concorde à la valeur de l'expression.

7.2.4.3 Exemple

```
TypUpsVmShort I;

switch (I)
   {
   case 0 :
      ...
      break;

   case 1 :
      ...
      break;

   default :
      ...
      break;
   }
```

7.2.4.4 Avertissement

Néant.

7.2.4.5 Voir aussi

switch pour définir un éclatement.

case pour définir un cas particulier.

break pour sortir d'un cas de test.

7.2.5 do

7.2.5.1 Prototypes

do *Instruction*

while **(** *Expression* **) ;**

7.2.5.2 Description

Exécute une boucle de la manière suivante :

1) Exécute l'instruction du corps de la boucle.
2) Exécute l'expression d'arrêt.
3) Si le résultat de l'expression vaut ***0*** alors on sort de la boucle.
4) Boucler à l'étape 2.

7.2.5.3 Exemple

```
TypUpsVmShort I;

I=0;
do
   {
   ...
   I++;
   } while (I<10) ;
```

7.2.5.4 Avertissement

M L'expression doit être de type entier.

7.2.5.5 Voir aussi

for et **while** pour définir une boucle autrement.

continue pour passer à l'itération suivante.

break pour sortir de la boucle.

7.2.6 else

7.2.6.1 Prototypes

else if **(** *Expression* **)** *Instruction SuiteTester*

else *Instruction*

7.2.6.2 Description

Exécute une alternative de test de la manière suivante si toutes les alternatives précédentes ont échoué :

1) Exécute l'expression de test.
2) Si le résultat de l'expression vaut ***0*** alors passe à l'alternative suivante ou sort de la cascade de tests.
3) Exécute l'instruction du corps du test.
4) Sort de la cascade de tests.

7.2.6.3 Exemple

```
TypUpsVmShort I;

if (I==0)
   {
   ...
   }
else if (I==1)
   {
   ...
   }
else
   {
   ...
   }
```

7.2.6.4 Avertissement

M L'expression doit être de type entier.

7.2.6.5 Voir aussi

`if` pour démarrer une cascade de tests.

7.2.7 enum

7.2.7.1 Prototypes

`enum` ***NomDeLEnumere***

{

DefinitionDesValeurEnumerees

};

7.2.7.2 Description

Déclare l'énuméré de nom ***NomDeLEnumere***.

7.2.7.3 Exemple

```
enum MonEnumere
   {
   MaValeurA=1,
   MaValeurA=2,
```

```
   MaValeurA=3
   };

enum MonEnumere A;

A=MaValeurA;
A=(enum MonEnumere)1;
```

7.2.7.4 Avertissement

Toutes les valeurs de l'énuméré doivent comporter une valeur entière correspondant à une séquence s'incrémentant.

7.2.7.5 Voir aussi

Néant.

7.2.8 extern

7.2.8.1 Prototypes

```
extern
extern "C" {
```

7.2.8.2 Description

Spécifie qu'une déclaration est externe i.e. qu'elle est définie dans un autre composant.

Spécifie que les définitions de la liste sont écrites en langage ***C*** et non en ***C++***.

7.2.8.3 Exemple

```
extern "C"
{
TypUpsVmShort A;

TypUpsVmShort F(TypUpsVmShort P);

extern TypUpsVmShort B;

extern TypUpsVmShort G(TypUpsVmShort P);
}
```

7.2.8.4 Avertissement

Seule la spécification du langage ***C*** est possible.

7.2.8.5 Voir aussi

`static` pour définir une déclaration propre au composant.

7.2.9 for

7.2.9.1 Prototypes

for (*ExpressionOption* ; *ExpressionOption* ; *ExpressionOption*)

Instruction

7.2.9.2 Description

Exécute une boucle de la manière suivante :

1) Exécute la première expression d'initialisation.
2) Exécute la seconde expression d'arrêt.
3) Si le résultat de l'expression deux vaut ***0*** alors sort de la boucle.
4) Exécute l'instruction du corps de la boucle.
5) Exécute la troisième expression d'incrémentation.
6) Boucler à l'étape 2.

7.2.9.3 Exemple

```
TypUpsVmShort I;

for (I=0;I<10;I++)
   {
   ...
   }
```

7.2.9.4 Avertissement

MM

La seconde expression doit être de type entier.

Si la condition d'arrêt n'est pas précisée, la boucle est sans fin.

7.2.9.5 Voir aussi

do et **while** pour définir une boucle autrement.

continue pour passer à l'itération suivante.

break pour sortir de la boucle.

7.2.10 if

7.2.10.1 Prototypes

if (*Expression*) *Instruction SuiteTester*

7.2.10.2 Description

Exécute un test de la manière suivante :

1) Exécute l'expression de test.
2) Si le résultat de l'expression deux vaut ***0*** alors passe à l'alternative suivante ou sort de la cascade de tests.
3) Exécute l'instruction du corps du test.
4) Sort de la cascade de tests.

7.2.10.3 Exemple

```
TypUpsVmShort I;

if (I==0)
   {
```

```
   ...
   }
else if (I==1)
   {
   ...
   }
else
   {
   ...
   }
```

7.2.10.4 Avertissement

M L'expression doit être de type entier.

7.2.10.5 Voir aussi

else pour définir l'alternative d'un test.

switch pour définir un test autrement.

7.2.11 return

7.2.11.1 Prototypes

return *ExpressionOption*

7.2.11.2 Description

Interromps l'exécution d'une liste d'instructions d'une fonction en retournant la valeur de l'expression.

Cette instruction **return** est employée dans le corps d'une fonction.

7.2.11.3 Exemple

```
TypUpsVmShort MonAppel(...)
{
...
return 0;
...
}

void MonAppel2(...)
{
...
return;
...
}
```

7.2.11.4 Avertissement

M Le type de la valeur retournée doit être identique au type du résultat de la fonction.

Si la fonction n'a pas de résultat, l'instruction **return** ne doit pas retourner d'expression.

7.2.11.5 Voir aussi

Néant.

7.2.12 sizeof

7.2.12.1 Prototypes

```
sizeof ( NomDUnType )
```

7.2.12.2 Description

Retourne la taille exprimée en octets nécessaire pour le stockage d'une information de ce type.

7.2.12.3 Exemple

```
sizeof(TypUpsVmShort)

sizeof(TypUpsVmLong *)
```

7.2.12.4 Avertissement

Néant.

7.2.12.5 Voir aussi

`extern` pour définir une déclaration externe au composant.

7.2.13 static

7.2.13.1 Prototypes

```
static
```

7.2.13.2 Description

Spécifie qu'une déclaration est statique i.e. qu'elle est définie dans ce composant et qu'elle ne peut être utilisée par un autre composant.

7.2.13.3 Exemple

```
extern "C"
{
TypUpsVmShort A;

TypUpsVmShort F(TypUpsVmShort P);

static TypUpsVmShort B;

static TypUpsVmShort G(TypUpsVmShort P);
}
```

7.2.13.4 Avertissement

Néant.

7.2.13.5 Voir aussi

`extern` pour définir une déclaration externe au composant.

7.2.14 struct

7.2.14.1 Prototypes

struct *NomDeLaStructure*

struct *NomDeLaStructure*

{

ListeDesChamps

}

7.2.14.2 Description

Identifie une structure.

Définit les champs d'une structure. Tous les champs sont simultanément présents à l'inverse de l'union.

7.2.14.3 Exemple

```
typedef struct montypec
   {
   TypUpsVmShort MonChampA;
   TypUpsVmDouble MonChampB;
   } MonTypeC, *PMonTypeC;

MonTypeC C;
PMonTypeC PC;
```

7.2.14.4 Avertissement

M Une structure ne peut être employée sans alias.

7.2.14.5 Voir aussi

typedef pour déclarer l'alias d'une structure.

union pour déclarer une union.

sizeof pour mesurer la taille d'un type.

7.2.15 switch

7.2.15.1 Prototypes

switch **(** *Expression* **)**

{

ListeDeCas CasParDefaut

}

7.2.15.2 Description

Exécute un test de la manière suivante :

1) Exécute l'expression de test.

2) Compare le résultat de l'expression à chaque cas de test.
3) Au premier cas de test dont la valeur correspond, exécute la liste des instructions du cas
4) Continue sur les instructions du cas suivant.

7.2.15.3 Exemple

```
TypUpsVmShort I;

switch (I)
   {
   case 0 :
      ...
      break;

   case 1 :
      ...
      break;

   default :
      ...
      break;
   }
```

7.2.15.4 Avertissement

M L'expression doit être de type entier.

7.2.15.5 Voir aussi

case pour définir un cas de test.

if pour définir un test autrement.

break pour sortir d'un cas de test.

7.2.16 typedef

7.2.16.1 Prototypes

typedef *SuiteType* **;**

7.2.16.2 Description

Déclare des alias sur un type.

7.2.16.3 Exemple

```
typedef TypUpsVmShort MonTypeA;

typedef TypUpsVmShort (*MonTypeB)(TypUpsVmVoid);

typedef struct montypec
   {
   TypUpsVmShort MonChampA;
   TypUpsVmDouble MonChampB;
   } MonTypeC, *PMonTypeC;
```

```
typedef union montyped
   {
   TypUpsVmShort MonChampA;
   TypUpsVmDouble MonChampB;
   } MonTypeD, *PMonTypeD;

MonTypeA A;
MonTypeB B;
MonTypeC C;
PMonTypeC PC;
MonTypeD D;
PMonTypeD PD;
```

7.2.16.4 Avertissement

M Une structure ou une union ne peut être employée sans alias.

7.2.16.5 Voir aussi

struct pour déclarer une structure.

union pour déclarer une union.

sizeof pour mesurer la taille d'un type.

7.2.17 union

7.2.17.1 Prototypes

union ***NomDeLUnion***

union ***NomDeLUnion***

{

ListeDesChamps

}

7.2.17.2 Description

Identifie une union.

Définit les champs d'une union. Seul un des champs est simultanément présent à l'inverse de la structure.

7.2.17.3 Exemple

```
typedef union montyped
   {
   TypUpsVmShort MonChampA;
   TypUpsVmDouble MonChampB;
   } MonTypeD, *PMonTypeD;

MonTypeD D;
PMonTypeD PD;
```

7.2.17.4 Avertissement

Une union ne peut être employée sans alias.

M

7.2.17.5 Voir aussi

typedef pour déclarer l'alias d'une union.

struct pour déclarer une structure.

sizeof pour mesurer la taille d'un type.

7.2.18 void

7.2.18.1 Prototypes

void

7.2.18.2 Description

Définit le type vide.

Il est utilisé soit pour :

- Spécifier qu'une fonction n'a pas de paramètre.
- Spécifier qu'une fonction n'a pas de résultat.
- Construire un type pointeur vers un type inconnu.

7.2.18.3 Exemple

```
typedef void *Pointeur;

void MaProcedure(void)
{
...
}
```

7.2.18.4 Avertissement

M Il est préférable d'employer le type ***TypUpsVmVoid*** d'***Up ! Virtual Technical Machine***.

7.2.18.5 Voir aussi

typedef pour déclarer l'alias d'un type.

7.2.19 while

7.2.19.1 Prototypes

while **(** *Expression* **)**

Instruction

7.2.19.2 Description

Exécute une boucle de la manière suivante :

1) Exécute l'expression d'arrêt.
2) Si le résultat de l'expression vaut ***0*** alors on sort de la boucle.

3) Exécute l'instruction du corps de la boucle.
4) Boucler à l'étape 2.

7.2.19.3 Exemple

```
TypUpsVmShort I;

I=0;
while (I<10)
   {
   ...
   I++;
   }
```

7.2.19.4 Avertissement

M L'expression doit être de type entier.

7.2.19.5 Voir aussi

`do` et `for` pour définir une boucle autrement.

`continue` pour passer à l'itération suivante.

`break` pour sortir de la boucle.

Fin de document