μ-Πασχαλ

(μ-Pascal **Version 1.0**)

A propos ...

Le sous-ensemble du langage Pascal proposé
Un écran typique

A propos de "µ-Pascal".

L'apprentissage d'un nouveau langage est toujours difficile. Les concepts fondamentaux de la programmation tels que les boucles, les tableaux et surtout les pointeurs sont souvent abstrait pour le débutant.

Le meilleur moyen de se faire une idée est encore de **voir** les choses. C'est pourquoi une représentation graphique des variables pendant le déroulement du programme est un plus appréciable. De plus ce coté visuel est un atout irremplaçable pour comprendre ce que sont les pointeurs. On voit afin quelle est la différence entre un tableau de pointeurs et un pointeur sur un tableau ...

Ce programme a été réalisé par Laurent Riesterer. Si vous estimez qu'il vous est utile, envoyer votre contribution financière à l'auteur. Comme la majorité des utilisateurs seront des débutants, surement jeunes et ayant peu de ressources, je ne fixe pas de contributions minimale. Toutefois, plus les gens se montrerons intéressés, plus je pourrais réaliser rapidement des améliorations.

N'hésitez pas à m'écrire à l'adresse suivante pour me faire part de toutes vos remarques :

Laurent Riesterer
2, rue H.Urbain
52410 Roches/Marne
France

e-Mail : riestere@ensta.fr

Les possibles améliorations pour la future version sont :

- implémentations des fonctions de Couper-Coller dans l'éditeur.
- extensions du langage aux tableaux multi-dimensionnels.
- mises en place d'un tutorial.
- traduction en anglais pour une plus large diffusion.
- écriture du même programme mais pour l'apprentissage du C.

Le sous-ensemble du langage Pascal proposé.

Le langage µ-Pascal est un sous-ensemble simple, mais non trivial du langage Pascal. Les types de données reconnus sont :

- les entiers (**`integer`**) compris entre -32767 et 32768,
- les types composites (**`record`**) avec possibilité d'inclure à l'intérieur un type représentant une structure (mais on ne peut pas la déclarer directement), mais on ne peut pas inclure de tableaux dans une structure,
- les pointeurs, pouvant pointer vers n'importe quel type (possibilité de définir des pointeurs multiples ou "handle"), le pointeur nul étant défini par '**`nil`**',
- les tableaux unidimensionnels (**`array [`**`nb .. nb`**`] of`** `type_simple`) dont les éléments peuvent être soit des entiers, soit des pointeurs.

On peut définir des fonctions et des procédures, avec passage des paramètres par valeur ou par référence, selon la même syntaxe que le langage Pascal classique. Le type renvoyé par une fonction doit être soit un entier, soit un pointeur. Une sous-routine peut posséder des variables locales, mais on ne peut pas redéfinir de type locaux et définir des sous-routines locales. Les seules restrictions sont qu'on ne peut pas passer d'arguments fonctionnels (passage de procédure ou de fonction) et qu'on ne peut pas déclarer de procédure en '`forward`', d'où l'impossibilité de faire de la récurrence croisée.

Les instructions du langage sont :

- l'affectation **`:=`** ,
- le test **`if ... then ... else ...`** ,
- les boucles **`repeat ... until`** , **`while ... do ...`** , **`for ... do`** (**`downto`**) **`...`** ,
- la création d'une nouvelle cellule dans le tas (**`new(...)`**),
- la lecture et l'affichage des variables entières, et l'affichage de chaînes de texte littérales,
- la génération de nombres aléatoires (**`random(`**`borne supérieure`**`)`**) compris entre 0 et borne supérieure,
- les opérations mathématiques +,-,*,/ et les opérateurs **div** et **mod**, sur les nombres entiers.

Les deux dernières fonctions supportées sont les opérations d'entrée/sortie **`writeln`** et **`readln`**. Leur comportement est un peu "aménagé". Pour la procédure **`writeln`**, on peut afficher soit une chaine soit un entier; aucune fonction de formatage n'est supportée. Le résultat n'est pas affiché explicitement car la chaine est visible dans le texte source du programme et la valeur d'une variable se trouve dans la partie graphique (si on ne sait pas où la trouver car l'expression est complexe, utiliser l'avance détaillée qui vous amènera vers la valeur en question).
Pour la fonction **`readln`** qui lit uniquement un entier, une émulation est proposée grâce à une boite de dialogue. Il suffit de taper la valeur désirée. Le résultat est directement visualisé sur la représentation graphique.

Pour les connaisseurs, voici la grammaire exacte :

<program> → **program** <ident>**;**
<type section>
<var section>
<sub-routine section>
begin
<instructions list>

end .

<type section> → Ø | **type** <ident> **=** <type> **;** <type declaration>
<type declaration> → Ø | <ident> **=** <type> **;** <type declaration>
<type> → <simple type> | <complex type>
<simple type> → <integer> | <ident>
<complex type> → **record** <var declarations> **end** |
array [<integer> **..** <integer> **]** **of** <ident>

<var section> → Ø | **var** <ident> <var declaration> <var declarations>
<var declarations> → Ø | <ident> <var declaration> <var declarations>
<var declaration> → <idents> **:** <ident> **;**
<idents> → Ø | **,** <ident> <idents>

<sub-routine section> → Ø
| <sub-routine header>
<var section>
begin
<instructions list>
end ;
<sub-routine section>
<sub-routine header> → **function** <ident> <parameters> **:** <simple type> **;**
| **procedure** <ident> <parameters> **;**
<parameters> → Ø | **(** <parameters list> **)**
<parameters list> → <reference> <ident> <idents> : <ident>
| <reference> <ident> <idents> : <ident> **;** <parameters list>
<reference> → Ø | **var**

<instructions list> → <instruction> <instructions>
<instructions> → Ø | **;** <instructions list>
<instruction> → Ø
| <variable> **:=** <expression>
| <sub-routine call>
| **if** <test> **then** <instruction> <alternative>
| **while** <test> **do** <instruction>
| **repeat** <instructions list> **until** <test>
| **for** <ident> **:=** <expression> <for list> **do** <instruction>
| **new (** <variable> **)**
| **writeln (** <writable> **)**
| **readln (** <variable> **)**
| **begin** <instructions list> **end**
<alternative> → Ø | **else** <instruction>
<for list> → **to** <expression>
| **donwto** <expression>
<sub-routine call> → <ident> | <ident> **(** <expressions list> **)**
<expressions list> → <expression> | <expressions>
<expressions> → Ø | **,** <expression> <expressions>
<writable> → <string> | <ident> <accessors>

<variable> → <ident> <accessors>
<accessors> → Ø | <accessors> | **.** <ident> <accessors>
| **[** <expression> **]** <accessors>

<expression> → <term> <terms>
<term> → <factor> <factors>
<terms> → **+** <term> <terms> | **-** <term> <terms>

<factor> → **nil** | <integer> | **random (** <expression> **)**
| <variable> | <sub-routine call>
| **-** <factor> | **+** <factor> | **(** <expression> **)**

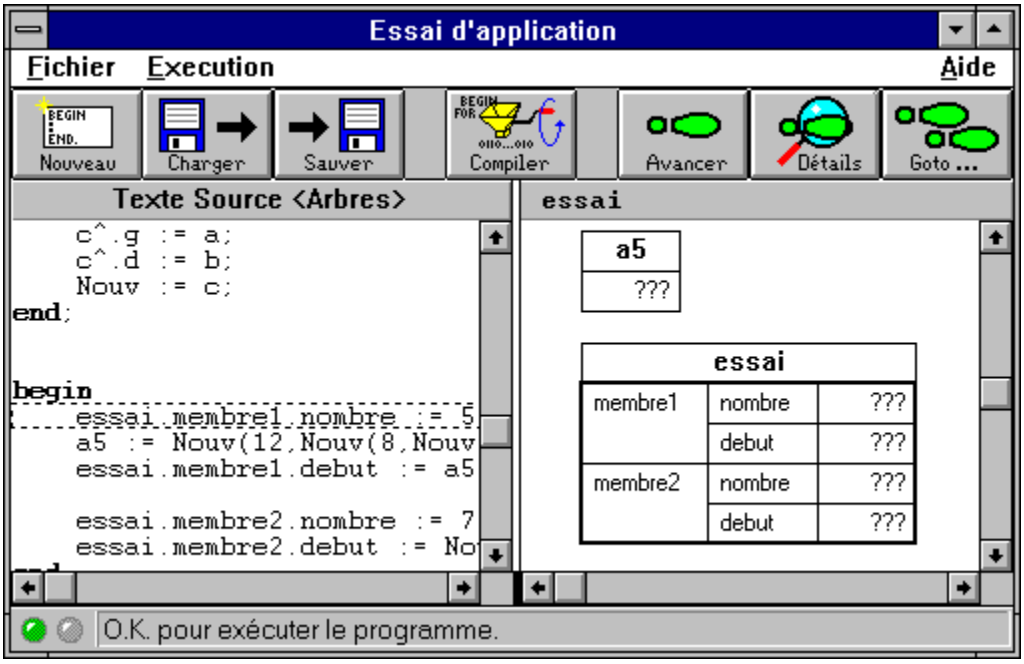
<factors> → Ø | ***** <factor> <factors> | */* <factor> <factors>
| **mod** <factor> <factors> | **div** <factor> <factors>

<test> → <expression> <relation> <expression>

<relation> → **=** | **<>** | **<** | **>** | **<=** | **>=**

Les différentes parties de l'écran.

Voici un écran typique. Pour avoir des renseignements complémentaires, cliquez sur la partie de l'écran qui vous intéresse (les parties intéressantes sont celles où le curseur de la souris se change en main).



Pour un accès direct, voici des liens vers les sujets essentiels :

La compilation.
La fenêtre d'édition.
La fenêtre de visualisation.

Le menu Fichier.

Le menu Fichier est composé des éléments suivants :

- Nouveau
- Charger
- Sauver
- Sauver Sous
- Quitter

Permet de créer un nouveau texte source.

ATTENTION: l'ancien est effacé sans demande de confirmation !

Permet de charger le texte source d'un programme.

ATTENTION: l'ancien texte présent est écrasé sans avertissement !

Permet de sauvegarder le texte du programme en cours d'édition.

Pour un fichier déjà chargé, la sauvegarde s'effectue sous le même nom.
Pour un nouveau programme, le nom est demandé.
Pour changer le nom lors de la sauvegarde, utiliser "Fichier | Sauver Sous".

Permet de sauvegarde le texte source du programme et de lui donner un nouveau nom.

Permet de quitter le programme.

Le menu Execution.

Le menu Execution est composé des éléments suivants :

- Compiler
- Avancer
- Détails
- Goto
- Sortir le code

La compilation des programmes.

Cette commande permet de compiler le texte source contenu dans la fenêtre d'édition.
Si au cours de cette phase, une erreur survient (erreur de syntaxe, erreur lexicale, ...) alors la compilation est arrêtée, la ligne fautive est mise en évidence par un cadre rouge et un message expliquant l'erreur s'affiche dans la ligne d'information en bas de la fenêtre.

La liste des messages d'erreurs possibles sont :

A la compilation :

- Il manque le nom du programme.
- Nom du type attendu.
- '=' attendu.
- ';' attendu.
- ':' attendu.
- 'end' attendu.
- '[' attendu.
- ']' attendu.
- '..' attendu.
- 'of' attendu.
- ')' attendu pour clore la liste des paramètres.
- '(' attendu pour la liste des paramètres.
- ':=' attendu.
- 'do' attendu.
- 'then' attendu.
- 'until' attendu.
- 'do' ou 'downto' attendu.
- '(' attendue.
- ')' attendue.
- 'program' attendu.
- 'begin' principal attendu.
- 'end' final attendu.
- Point final attendu.
- ';' attendu après l'entête de la procédure.
- ';' attendu après l'entête de la fonction.
- 'begin' attendu pour le corps de la procédure.
- 'begin' attendu pour le corps de la fonction.
- 'end' attendu pour le corps de la procédure.
- 'end' attendu pour le corps de la fonction.
- Identificateur attendu après ','.
- Identificateur attendu après 'var'.
- Identificateur attendu pour la variable locale.
- Identificateur pour l'argument attendu.
- Nom pour la procédure attendu.
- Nom pour la fonction attendu.
- Nom du compteur de boucle attendu.
- Nom de champ attendu après '.'.
- Nom de variable attendu.
- Nombre entier pour la borne de 'random' attendu.
- Opérateur de relation attendu dans le test.
- Type simple attendu après '^'.
- Variable pour stocker le résultat de la fonction attendu.
- Variable de type pointeur attendue.

- Plus de mémoire ...
- Dépassement de la taille maximale de l'exécutable ...
- Redéclaration du type.
- Le type 'xxx' n'existe pas.
- Pas de tableau dans un record.
- Référence circulaire.
- Les éléments d'un tableau doivent être des entiers ou des pointeurs.
- Redéclaration de variable.
- Même nom pour une variable et une sous-routine impossible.
- Même nom pour deux sous-routines impossible.
- Type entier ou pointeur attendu pour le résultat d'une fonction.
- Types incompatibles dans le passage de paramètres.
- La variable 'xxx' n'existe pas.
- Variable de type entier attendue pour le compteur de boucle.
- Les bornes du compteur doivent être des expressions entières.
- La sous-routine 'xxx' n'existe pas.
- Types incompatibles dans l'affectation.
- Seules les variables entières peuvent être affichées.
- Mauvais argument pour 'writeln'.
- On ne peut accéder aux champs que pour un record.
- Nom de champ inconnu.
- On ne peut accéder aux éléments que pour un tableau.
- Expression de valeur entière attendue pour l'index d'un tableau.
- On ne peut additionner que des entiers.
- On ne peut soustraitre que des entiers.
- On ne peut multiplier que des entiers.
- On ne peut diviser que des entiers.
- On ne peut utiliser 'div' que sur des entiers.
- On ne peut utiliser 'mod' que sur des entiers.
- Types incompatibles.
- Appel d'une procédure dans une expression impossible.
- Erreur de syntaxe.
- On ne peut comparer que des valeurs du même type.
- On ne peut comparer que des entiers ou des pointeurs.

Permet d'avancer d'une ligne dans le programme.

Si la ligne qui va être executée contient un appel à une sous-routine (procédure ou fonction), alors on entre dans cell-ci pour suivre son déroulement.

Pour éviter d'executer une sous-routine, il faut utiliser la commande "Fichier | Goto" et selectionner la ligne suivant l'appel.

Voici une liste des erreurs pouvant apparaître lors de l'execution :

- Débordement de la pile d'exécution ...
- Essai de retirer un objet inexistant de la pile d'exécution ...
- Valeur du pointeur indéfinie.
- Impossible de prendre le contenu d'un pointeur valant NIL.
- Valeur de l'index en dehors des bornes. (pour une opération sur un tableau)
- Division par zéro.
- Contenu de la variable (entière) indéfini.
- Contenu de la varaible (pointeur) indéfini.
- Plus de mémoire ...

Permet de suivre pas-à-pas toutes les instructions de la ligne, en particulier de voir le cheminement pour acceder aux variables.

Pour plus de renseignements, cliquez dans la fenêtre où sont visualisées les variables.

Permet d'aller directement à une ligne donnée.

Le curseur se transforme en des petits pas comme le bouton de la barre d'outils avec une croix pour indiquer précisement le lieu du 'clic'.
L'execution se déroule alors pour arriver jusqu'à la ligne choisie. Si celle-ci n'est pas atteinte lors de la suite de l'execution, le programme se poursuit jusqu'à la fin.

ATTENTION: Toutes les lignes ne sont pas executées. Par exemple cliquer sur une ligne blanche ne produira pas d'arrêt sur la prochaine ligne contenant des instructions. Il faut cliquer SUR les instructions à atteindre.

SCROLLING: Une fois l'option "Execution | Goto" choisie ou le bouton cliqué, il n'est plus possible de scroller le texte. Il faut rendre la ligne où on veut aller visible AVANT de lancer une commande Goto.

Sortie du code.

Cette option réservée aux personnes curieuse de voir le pseudo-code machine généré par le compilateur. Ce code n'est pas optimisé, il est simple et a pour seul but de permettre une execution efficace dans le cadre donné, à savoir la représentation graphique des variables pendant le déroulement du programme.

La sortie est effectuée dans un fichier dont on choisit le nom et qui est ensuite visualisé avec le Bloc-Note de Windows.

Le compilateur genère du pseudo-code pour une machine ayant des intructions des la forme :
Instruction [Paramètre1 [, Paramètre2]]

La signification de ces diverses instructions élémentaires est la suivante, les paramètres utilisés sont donnés entre parenthèse :

Stop(Ligne)	Fixe la ligne d'arrêt pendant l'exécution du programme.
Fin(Ligne)	Marque la fin du programme et la ligne où elle se situe.
Aller(NewPC) **AllerVrai(NewPC)** **AllerFaux(NewPC)**	Détermine le nouveau compteur ordinal en tenant compte éventuellement d'une condition se trouvant au sommet de la pile d'exécution.
Pointage()	Accède au contenu du pointeur dont l'adresse se trouve au sommet de la pile d'exécution
Champ(VarRecord)	Accède au champ d'une structure décrit par '`VarRecord`' (offset, type, taille).
Element(Bornes,Size)	Accède à l'élément d'un tableau dont l'index est situé sur le sommet de la pile. La routine teste si l'index est bien compris entre les bornes définies pour le tableau en question. 'Size' est la taille d'un élément.
DebutElement()	Signale qu'on calcule un index d'un tableau, utilisé pour interrompre la décomposition des variables.
Add() , Sub() **Mul() , Div()** **Neg() , Mod()**	Routines mathématiques de base sur les entiers.
Randomise()	Routine renvoyant un nombre entier aléatoire compris entre 0 et la valeur située sur le sommet de la pile.
Incrementer(Sens)	Routine additionnant ou soustrayant un à la variable dont l'adresse est située sur le sommet de la pile. Cette routine est utilisé dans la gestion des boucles '`for...do`' pour gérer le compteur de boucle.
PrepareFonc(Memoire,Ident)	Alloue '`Memoire`' octets de mémoire pour les variables locales de la sous-routine que l'on va appeler d'identificateur 'Ident'. Sauvegarde le contexte d'exécution.
Parametre(Locale,Size)	Transmet le paramètre décrit par '`Locale`' (passage par valeur ou par référence, emplacement mémoire où le stocker, type) à la sous-routine que l'on va appeler. La valeur du paramètre se trouve au sommet de la pile pour les types simples; on effectue une copie de l'objet dont l'adresse se trouve au sommet de la pile pour les types complexes, la taille de cet objet étant '`Size`' octets.
AppelFonc(PCFonc)	Appel de la sous-routine en se branchant à la valeur du compteur

	ordinal '`PCFonc`'.
Retour()	Sort d'une sous-routine en restaurant le contexte d'exécution précédent (adresse des variables locales, adresse de la variable en cours d'affectation pour une fonction, compteur ordinal suivant l'appel) et libère la mémoire occupée par les variables locales de la sous-routine qui vient de se terminer.
Empiler(Nombre,Ident)	Empile le nombre '`Nombre`' au sommet de la pile. Si '`Ident`' vaut -1, il s'agit d'une valeur numérique, sinon il s'agit de l'adresse de la variable de nom référencé par '`Ident`'.
EmpilerLoc(Nombre,Ident)	Empile l'adresse de la variable locale dont l'offset vaut '`Nombre`' par rapport au début de la zone des variables locales dont l'adresse est donnée par '`MemoireLocale`'.
EmpilerLoc(Nombre,Ident)	Dans le cas d'une variable locale qui est un argument passé par référence, empile l'adresse réelle de la variable pour permettre des modifications de sa valeur de manière globale.
Contenu(Genre)	Empile la valeur de la variable dont l'adresse se trouve au sommet de la pile. Ceci permet d'obtenir une valeur-*d* afin de pouvoir réaliser une affectation.
FinGValue()	Indique que l'on a fini de calculer la valeur-*g* de l'affectation qui va venir et qu'il faut garder en mémoire l'adresse calculée afin de positionner correctement le champ '`Init`' de la variable pour signaler qu'elle a été initialisée.
Affecter(Genre,Size)	Affecte la valeur située au sommet de la pile à la variable dont l'adresse se situe immédiatement en-dessous pour les types simples; effectue une copie de l'objet pour les types complexes. On positionne ensuite le champ '`Init`' de la variable pour signaler qu'elle a été initialisée.
New(Type)	Crée dans le tas une nouvelle variable de type '`Type`' qui sera pointée par la variable de type pointeur dont l'adresse se trouve au sommet de la pile.

La fenêtre d'édition.

Cette fenêtre permet d'éditer le texte source du programme et de suivre le déroulement du programme lors de l'execution.

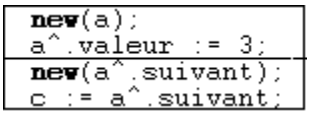
En phase d'édition.

La fonction d'édition proposée dans cette version sont simple. On peut se déplacer avec les touches du curseur, insérer des caractères, en supprimer soit avec `Backspace` soit avec `Delete`, aller au début d'une ligne avec `Home` ou à la fin avec `Fin`.

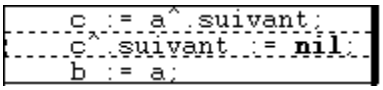
Ces fonctionnalités ne sont pas très performantes, mais le but principal du programme est l'execution de petit programmes simples pour comprendre des concepts de programmation, pas de faire un éditeur de texte... Une prochaine version devrait remédier à ces petits inconvénients.

En phase d'execution.

Le texte de la fenêtre est ajusté de manière à montrer le point d'arrêt actuel.
On peut avoir deux formes de point d'arrêt :



La ligne est placée après la ligne qui vient d'être executée et avant la ligne qui va être executée.



Le cadre pointillé montre la ligne en cours d'execution. Ce cadre apparaît lorsque l'on demande le détails d'execution d'un ligne.

La fenêtre de visualisation des variables.

La fenêtre se compose de deux parties : la visualisation graphique proprement dite et la barre de titre de la fenêtre.

La partie graphique.

Les variables sont représentées de manière graphique et le sens de cette représentation se comprend aisement. Les quelques point de détails à préciser sont les suivants :

- Les valeurs (entiers ou pointeurs) non initialisées sont représentées par '`???`'.
- Les entiers sont représentés par leur valeur numérique.
- Les pointeurs valant '`nil`' sont représentés par une case barrée.
- Les autres pointeur possèdent une flèche vers la variable pointée.

- Lors de l'appel d'une sous-routine, l'ensemble des variables (arguments, locales, et pour les fonctions valeur retournée qui est en fait une variable du nom de la fonction) appartenant à celle-ci est regroupé dans une cadre pointillé surmonté du nom de la sous-routine appelée. Dans l'ordre sont représentés : la valeur retournée dans le cas des fonctions, l'ensemble des arguments, et enfin les variables locales.

- Les variables représentant les arguments seront initialisées lors de l'appel, même si l'utilisateur passe en argument une variable non entièrement définie (à ses risques et périls). Les variables passées par référence sont symbolisées par un trait épais en pointillé vers les 'vraies' variables, celles dont les modifications survenues pendant la sous-routine perdureront après la fin de celle-ci.

- Les tableaux sont représentés horizontalement pour les tableaux d'entiers et verticalement pour les tableaux de pointeurs. Le nom, s'il existe, est affiché en haut, puis on marque les indices et enfin les valeurs.

- le **cadre gras** qui s'affiche lors de l'execution en détails indique la partie de la variable à laquelle on accède à un moment donné.

La barre de titre.

Elle est utilisée pour deux choses :

- la décomposition des variables lorsque l'on execute le programme avec la commande détails. Au fur et à mesure, on affiche les différentes étapes pour acceder à la valeur souhaitée.
- pour le retour des fonctions, on affiche la valeur renvoyée par la fonction qui est soit un entier soit un pointeur.

Le curseur change de forme en arrivant sur cette barre qui permet de choisir le partage de l'écran et la proportion accordée à chaque fenêtre. Une fois que le curseur a changé de forme, il faut appuyer sur le bouton gauche et le maintenir appuyer pour le réglage.

Indique les messages générés par le programme. Un bip sonore est émis chaque fois qu'un nouveau message arrive afin d'attirer l'attention de l'utilisateur.

Les messages peuvent êtres des erreurs de compilation (ou un succès !), des erreurs lors de la manipulation des fichiers, des erreurs ou des avertissements lors de l'executions.

Une diode rouge ou verte indique si le message provoque une erreur fatale empéchant la suite de l'opération en cours (diode rouge) ou alors une simple mise en garde ou le succès d'une compilation (diode verte).