**S O M M A I R E**

1. **Présentation générale** (Page 5)

PrésentationIDH\_Present IDH\_Object

Menu

ITraitementsIDH\_trait

Langage ZIDH\_langagez

DocumentationIDH\_doc

1. **Etapes de réalisation d’un programme sous KHAWARIZM** (Page 7)

Familiarisation avec le langage Z

Edition de l'algorithme

Vérification syntaxique

Execution

Simulation

Trace

Passage vers un langage de programmation

Programmation PASCAL ou C

Exemple d'un Z-algorithme

Equivalent PASCAL

Equivalent C

1. **Langage Z (Base)** (Page 15)

GénéralitésIDH\_gen

Structure d'un Z-algorithmeIDH\_struct

Action composéeIDH\_act

FonctionIDH\_fonct

ScalairesIDH\_obj

PointeursIDH\_pointeur

ExpressionsIDH\_exp

Actions élémentaires

Structures de contrôleIDH\_str

CommentairesIDH\_com

Actions de haut niveauIDH\_acth

Fonctions standards Mod, Min, Max, ExpIDH\_foncts1

Fonctions de génération aléatoire Aleachaine, AleanombreIDH\_foncts2

Fonctions sur chaînes de caractères Caract, LongchaineIDH\_foncts3

Exemple d'un Z-algorithme

1. **Langage Z (Structures de données)** (Page 21)

TableauxIDH\_tableau

StructuresIDH\_structure

Listes linéaires chaînéesIDH\_liste

Listes linéaires chaînées bilatéralesIDH\_listebi

Files d'attenteIDH\_files

PilesIDH\_pile

Arbres de recherche binaireIDH\_arbreb

Arbres de recherche m-aireIDH\_arbrem

Fichiers

IDH\_Fichier

1. **Machines abstraites** (Page 26)

VecteursIDH\_Pvect

Structures

Listes linéaires chaînéesIDH\_Pliste

Listes bidirectionnellesIDH\_Plistebi IDH\_Pstruct

PilesIDH\_Ppile

Files d'attente

Arbres de recherche binaireIDH\_Parb

Arbres de recherche m-aireIDH\_Parm

FichiersIDH\_Pfich

1. **Passage de Z vers PASCAL** (Page 31)

DéclarationsIDH\_Pdec

AffectationIDH\_Paff

ExpressionsIDH\_Pexp

La boucle TantqueIDH\_Ptq

La boucle PourIDH\_Ppour

L'alternative SiIDH\_Psi

LectureIDH\_Plect

EcritureIDH\_Pecr

Action composéeIDH\_Pact

FonctionIDH\_Pfonct

Fonctions standardsIDH\_Pstand

ProgrammeIDH\_progpas

1. **Implémentation des machines Z en PASCAL** (Page 36)

VecteursIDH\_Pvect

Structures

Listes linéaires chaînéesIDH\_Pliste

Listes bidirectionnellesIDH\_Plistebi IDH\_Pstruct

PilesIDH\_Ppile

Files d'attente

Arbres de recherche binaireIDH\_Parb

Arbres de recherche m-aireIDH\_Parm

FichiersIDH\_Pfich

1. **Passage de Z vers C** (Page 49)

DéclarationsIDH\_Pdec

AffectationIDH\_Paff

ExpressionsIDH\_Pexp

La boucle TantqueIDH\_Ptq

La boucle PourIDH\_Ppour

L'alternative SiIDH\_Psi

LectureIDH\_Plect

EcritureIDH\_Pecr

Action composéeIDH\_Pact

FonctionIDH\_Pfonct

Fonctions standardsIDH\_Pstand

ProgrammeIDH\_progpas

1. **Implémentation des machines Z en C** (Page 52)

VecteursIDH\_Pvect

Structures

Listes linéaires chaînéesIDH\_Pliste

Listes bidirectionnellesIDH\_Plistebi IDH\_Pstruct

PilesIDH\_Ppile

Files d'attente

Arbres de recherche binaireIDH\_Parb

Arbres de recherche m-aireIDH\_Parm

FichiersIDH\_PfichIDH\_Pfich

1. **Index des mots-clés** (Page 66)
2. **Présentation générale**

**Présentation**

KHAWARIZM est un environnement pour apprendre et approfondir les principales structures de données et de fichiers.

KHAWARIZM offre la possibilité d'écrire des algorithmes dans un langage algorithmique (langage Z), de les arranger, de les dérouler ou les simuler et de les traduire vers les langages de programmation PASCAL et C.

KHAWARIZM vise la conception assistée des algorithmes.

KHAWARIZM fournit une importante documentation.

**Menus**

KHAWARIZM offre plusieurs fenêtres montrant :

- les données (lectures)

- les résultats de l'exécution (écritures)

- les résultats de la simulation  (trace complète)

- l'algorithme traduit en PASCAL ou en C

A tout moment dans KHAWARIZM, vous pouvez invoquer l'aide (F1) ou actionner les opérations à l'aide de boutons

**Traitements**

KHAWARIZM offre les services suivants

- Un éditeur pour écrire vos algorithmes fournissant toute la documentation sur le langage Z.

- Un indenteur pour arranger vos algorithmes.

Ses principales fonctions sont :

. Chaque instruction est écrite sur une ligne différente,

. Les mots-clé sont réécrits en majuscule (ou minuscule),

. Le premier caractère de tout identificateur est réécrit en majuscule,

. Les structures de contrôle sont mises en relief,

. Les instructions de même niveau commencent sur la même colonne.

. Le "pas d'aération" est variable.

- Un interpréteur pour exécuter vos algorithmes en donnant comme résultat l'ensemble des écritures émises (Fenêtre Résultats).

- Un simulateur pour donner le déroulement complet (fenêtre "Simulation") de vos algorithmes en montrant l'évolution de tous les objets manipulés. Ce qui vous aide à corriger, voir construire vos algorithmes.

- Un traducteur permettant de traduire vos algorithmes en PASCAL ou C.

- Une documentation importante pour montrer le passage d'un Z-algorithme vers un programme PASCAL ou C grâce un hypertexte intégré.

**Langage Z**

Dans KHAWARIZM, Les algorithmes sont exprimés dans un langage algorithmique (le langage Z).

La particularité du langage Z réside dans le fait de pouvoir écrire des algorithmes sur des machines abstraites simulant les principales structures de données.

Le langage Z est conçu principalement pour les objectifs suivants :

- l'expérimentation sur les principales structures de données, peu importe leurs implémentations, en développant des algorithmes sur

. Les vecteurs,

. Les structures,

. Les listes linéaires chaînées,

. Les listes bilatérales,

. Les files d'attente,

. Les piles,

. Les arbres de recherche binaire,

. Les arbres de recherche m-aire.

- la création et la manipulation de structures de données complexes telles que

. Liste de files d'attente,

. Liste de piles,

. Arbre de listes,

. Liste de piles de vecteurs,

. Etc.

- l'écriture d'algorithmes récursifs.

Grâce à sa machine abstraite définie sur les fichiers, le langage Z permet aussi l'utilisation des fichiers et la construction aussi bien de structures simples que complexes de fichiers.

**Documentation**

KHAWARIZM offre toute la documentation sur le langage Z.

KHAWARIZM fournit les équivalents Z --> PASCAL et Z --> C.

KHAWARIZM donne quelques implémentations possibles en PASCAL et en C des différentes machines abstraites considérées dans le langage Z.

1. **Etapes de réalisation d’un programme sous KHAWARIZM**

**Familiarisation avec le langage algorithmique Z**

Apprendre le langage algorithmique utilisé. Utiliser l'aide en ligne.

**Edition de l'algorithme**

Ecrire un algorithme ou corriger un algorithme existant.

**Vérification syntaxique**

Lancer le module Arranger.

Répéter tant qu'il y a des erreurs

. Corriger les erreurs

. Relancer le module Arranger

A ce stade, votre algorithme est bien écrit et il a été indenté pour vous.( Vous pouvez changer les modes de présentation de votre algorithme ( voir "Options" du menu )

**Exécution**

Lancer l'exécution de votre algorithme

Les fenêtres montrent alors

- les données lues par votre algorithme (Bouton Données)

- les écritures émises par votre algorithme (Bouton Résultats)

Ou bien votre algorithme donne les résultats attendus ou pas. Dans ce dernier cas, lancer la simulation pour essayer de déterminer les erreurs de logique.

**Simulation**

Lancer la simulation de votre algorithme. Il s'agit d'une exécution avec une trace.

Les fenêtres montrent alors

- les données lues par votre algorithme (Bouton Données)

- les écritures émises par votre algorithme (Bouton Résultats)

- tous les changements effectués sur les objets utilisés (Bouton Simulation)

Vous avez ainsi la trace complète de votre algorithme que vous pouvez imprimer et l'analyser pour détecter les erreurs.

Si vous désirez voir de plus près les différents pas de votre algorithme, demander une trace.

**Trace**

Redemander la simulation avec trace. Vous pouvez alors suivre pas à pas l'évolution de votre algorithme, sortir de la boucle courante ou même du module courant.

Afin d'éviter d'avoir une trace complète qui peut être longue il est possible de limiter la longueur des boucles utilisées dans votre algorithme. Vous pouvez changer les modes de simulation (voir "Options" du menu).

**Passage vers un langage de programmation**

Une fois que votre algorithme "tourne", il est possible de le traduire automatiquement en PASCAL ou en C. Il suffit de cliquer sur le bouton "Vers Pascal" ou "Vers C". Deux fenêtres organisées en "Tuile" apparaissent. L'une contient votre algorithme et l'autre le résultat de votre traduction. Utiliser alors l'aide concernant le passage vers PASCAL ou C pour faciliter la compréhension de la traduction.

Dans cette aide, vous trouverez

- les équivalents Z vers PASCAL et Z vers C.

- toutes les implémentations des machines Z.

La tâche de Khawarizm s'arrête à ce niveau là.

**Programmation PASCAL ou C**

Utiliser le compilateur PASCAL ou C, pour finaliser définitivement votre programme. En particuliers, vous devez rajouter tous les modules de saisie des données et de restitution des résultats. Le langage Z n'offre pas des facilités pour de telles tâches.

**Exemple d'un Z-algorithme**

{Inclusion d'une liste linéaire chaînée dans une autre ?}

**SOIENT**

L1 , L2 **DES LISTES** ;

Rech , Tous : **FONCTION** ( **BOOLEEN** ) ;

**DEBUT**

**CREER\_LISTE** ( L1 , [ 2 , 5 , 9 , 8 , 3 , 6 ] ) ;

**CREER\_LISTE** ( L2 , [ 12 , 5 , 19 , 8 , 3 , 6 , 2 , 9 ] ) ;

**ECRIRE** ( Tous ( L1 , L2 ) )

**FIN**

{Recherche d'une valeur dans une liste linéaire chaînée}

**FONCTION** Rech ( L , Val ) : **BOOLEEN**

**SOIENT**

L **UNE LISTE** ;

Val **UN ENTIER** ;

**DEBUT**

SI L = **NIL**

Rech := **FAUX**

**SINON**

**SI VALEUR** ( L ) = Val

Rech := **VRAI**

**SINON**

Rech := Rech ( **SUIVANT** ( L ) , Val )

**FSI**

**FSI**

**FIN**

{Inclusion d'une liste dans une autre}

**FONCTION** Tous ( L1 , L2 ) : **BOOLEEN**

**SOIENT**

L1 , L2 **DES LISTES** ;

**DEBUT**

**SI** L1 = **NIL**

Tous := **VRAI**

**SINON**

**SI NON** Rech ( L2 , **VALEUR** ( L1 ) )

Tous := **FAUX**

**SINON**

Tous := Tous ( **SUIVANT** ( L1 ) , L2 )

**FSI**

**FSI**

**FIN**

**Equivalent PASCAL**

/\*\*       C o n v e r s i o n  Z vers PASCAL (Standard)   \*\*/

  PROGRAM Mon\_programme;

  {/// Implémentation \\\: LISTE DE ENTIERS }

  { Listes linéaires chaînées }

  TYPE

    Typeelem\_LI = INTEGER;

    Pointeur\_LI = ^Maillon\_LI; { type du champ 'Adresse' }

    Maillon\_LI = RECORD

        Val : Typeelem\_LI;

        Suiv : Pointeur\_LI

    END;

  PROCEDURE Allouer\_LI ( VAR P : Pointeur\_LI ) ;

    BEGIN  NEW(P)  END;

  PROCEDURE Liberer\_LI ( P : Pointeur\_LI ) ;

    BEGIN  DISPOSE(P)  END;

  PROCEDURE Aff\_val\_LI(P : Pointeur\_LI; Val : Typeelem\_LI );

    BEGIN P^.Val := Val  END;

  FUNCTION Valeur\_LI (P : Pointeur\_LI) : Typeelem\_LI;

    BEGIN   Valeur\_LI := P^.Val  END;

  FUNCTION Suivant\_LI( P : Pointeur\_LI) : Pointeur\_LI;

    BEGIN  Suivant\_LI := P^.Suiv END;

  PROCEDURE Aff\_adr\_LI( P, Q : Pointeur\_LI ) ;

    BEGIN P^.Suiv := Q    END;

  { Macro opérations }

   { Création d'une liste }

  PROCEDURE CREER\_LISTE\_LI ( VAR L : Pointeur\_LI ; Tab : Typetab\_V6I ; N: INTEGER)  ;

    VAR

      I : INTEGER;

      P, Q : Pointeur\_LI ;

    BEGIN

      L:=NIL;

      FOR I := 1 TO N DO

        BEGIN

          ALLOUER\_LI( Q ) ;

          AFF\_VAL\_LI (Q, Tab[I]);

          AFF\_ADR\_LI (Q, NIL);

          IF L = NIL

          THEN L := Q

          ELSE AFF\_ADR\_LI (P, Q);

          P := Q

        END;

    END;

  TYPE

    Typeelem\_V6I = INTEGER ;

    Typetab\_V6I = ARRAY[1..6] of Typeelem\_V6I ;

  TYPE

    Typeelem\_V8I = INTEGER ;

    Typetab\_V8I = ARRAY[1..8] of Typeelem\_V8I ;

   {Partie déclaration de variables }

   VAR

      L1 : Pointeur\_LI;

      L2 : Pointeur\_LI;

      T\_L1 : Typetab\_V6I ;

      T\_L2 : Typetab\_V8I ;

   { Prototypes des procédures et/ou fonctions }

   FUNCTION Rech (VAR L : Pointeur\_LI ; VAR Val : INTEGER) : BOOLEAN; FORWARD;

   FUNCTION Tous (VAR L1 : Pointeur\_LI ; VAR L2 : Pointeur\_LI) : BOOLEAN; FORWARD;

  { Recherche d'une valeur dans une liste linéaire chaînée }

   FUNCTION Rech (VAR L : Pointeur\_LI ; VAR Val : INTEGER) : BOOLEAN;

   {Partie déclaration de variables }

   VAR

      Rech2 : BOOLEAN ;

      Px1 : Pointeur\_LI;

   BEGIN

      IF L = NIL THEN BEGIN

         Rech2 := FALSE END

      ELSE

         BEGIN

         IF VALEUR\_LI(L ) = Val THEN BEGIN

            Rech2 := TRUE END

         ELSE

            BEGIN

            Px1 :=  SUIVANT\_LI(L ) ;

            Rech2 := Rech2 (Px1, Val )

         END

      END

      ;Rech := Rech2 ;

   END;

  { Inclusion d'une liste dans une autre }

   FUNCTION Tous (VAR L1 : Pointeur\_LI ; VAR L2 : Pointeur\_LI) : BOOLEAN;

   {Partie déclaration de variables }

   VAR

      Tous2 : BOOLEAN ;

      Px1 : Pointeur\_LI;

      Px2 : Pointeur\_LI;

   BEGIN

      IF L1 = NIL THEN BEGIN

         Tous2 := TRUE END

      ELSE

         BEGIN

         Px1 :=  VALEUR\_LI(L1 ) ;

         IF NOT Rech ( L2 ,Px1) THEN BEGIN

            Tous2 := FALSE END

         ELSE

            BEGIN

            Px2 :=  SUIVANT\_LI(L1 ) ;

            Tous2 := Tous2 (Px2, L2 )

         END

      END

      ;Tous := Tous2 ;

   END;

   {Corps du programme principal }

   BEGIN

      T\_L1 [ 1 ] := 2 ;

      T\_L1 [ 2 ] := 5 ;

      T\_L1 [ 3 ] := 9 ;

      T\_L1 [ 4 ] := 8 ;

      T\_L1 [ 5 ] := 3 ;

      T\_L1 [ 6 ] := 6 ;

      NEW(L1);

      CREER\_LISTE\_LI ( L1 , T\_L1 , 6 )  ;

      T\_L2 [ 1 ] := 12 ;

      T\_L2 [ 2 ] := 5 ;

      T\_L2 [ 3 ] := 19 ;

      T\_L2 [ 4 ] := 8 ;

      T\_L2 [ 5 ] := 3 ;

      T\_L2 [ 6 ] := 6 ;

      T\_L2 [ 7 ] := 2 ;

      T\_L2 [ 8 ] := 9 ;

      NEW(L2);

      CREER\_LISTE\_LI ( L2 , T\_L2 , 8 )  ;

      WRITE ( Tous(L1,L2) )

   END.

**Equivalent C**

  /\*\* C o n v e r s i o n   Z vers C (Standard) \*\*/

  #include <stdio.h>

  #include <stdlib.h>

  typedef int bool ;

  #define True 1

  #define False 0

  /\*\* Implémentation \*\*\: LISTE DE ENTIERS\*\*/

  /\*\* Listes linéaires chaînées \*\*/

  typedef int Typeelem\_Li   ;

  typedef struct Maillon\_Li \* Pointeur\_Li ;

  struct Maillon\_Li

    {

      Typeelem\_Li  Val ;

      Pointeur\_Li Suiv ;

    } ;

  Pointeur\_Li Allouer\_Li (Pointeur\_Li \*P)

    {

      \*P = (struct Maillon\_Li \*) malloc( sizeof( struct Maillon\_Li)) ;

    }

  void Aff\_val\_Li(Pointeur\_Li P, Typeelem\_Li V)

    {

      P->Val = V ;

    }

  void Aff\_adr\_Li( Pointeur\_Li P,  Pointeur\_Li Q)

    {

      P->Suiv = Q ;

    }

  Pointeur\_Li Suivant\_Li(  Pointeur\_Li P)

    { return( P->Suiv ) ;  }

  Typeelem\_Li Valeur\_Li( Pointeur\_Li P)

    { return( P->Val) ; }

  void Liberer\_Li ( Pointeur\_Li P)

    { free (P);}

  /\*\* Prototypes des fonctions \*\*/

  bool  Rech (Pointeur\_Li \*L , int \*Val) ;

  bool  Tous (Pointeur\_Li \*L1 , Pointeur\_Li \*L2) ;

   /\*\* Macro opérations \*\*/

  /\*\* Création d'une liste \*\*/

  void Creer\_liste\_Li ( Pointeur\_Li \*L, Typetab\_V6i Tab, int N)

    {

      int I ;

      Pointeur\_Li P, Q ;

      \*L =NULL;

      for( I=1;I<=N;++I)

        {

          Allouer\_Li( &Q ) ;

          Aff\_val\_Li (Q, Tab[I-1]);

          Aff\_adr\_Li (Q, NULL);

          if (\*L == NULL)

            \*L = Q ;

          else Aff\_adr\_Li (P, Q);

          P = Q ;

        }

    }

  /\*\* Pour les variables temporaires \*\*/

  typedef int Typeelem\_V6i;

  typedef Typeelem\_V6i Typetab\_V6i[6];

  /\*\* Pour les variables temporaires \*\*/

  typedef int Typeelem\_V8i;

  typedef Typeelem\_V8i Typetab\_V8i[8];

/\*\* Partie déclaration de variables \*\*/

     Pointeur\_Li L1;

     Pointeur\_Li L2;

       Typetab\_V6i T\_L1;

       Typetab\_V8i T\_L2;

  /\* Recherche d'une valeur dans une liste linéaire chaînée \*/

  bool  Rech (Pointeur\_Li \*L , int \*Val)

    {

     /\*\* Partie déclaration de variables \*\*/

      bool  Rech2 ;

     Pointeur\_Li Px1;

     /\*\* Corps du module \*\*/

     if( \*L == NULL) {

       Rech2  =  False; }

     else

       {

       if( Valeur\_Li(\*L ) == \*Val) {

         Rech2  =  True; }

       else

         {

         Px1 =  Suivant\_Li(\*L ) ;

         Rech2  =  Rech2 ( &Px1, & \*Val );

       }

     }

     return Rech2 ;

    }

  /\* Inclusion d'une liste dans une autre \*/

  bool  Tous (Pointeur\_Li \*L1 , Pointeur\_Li \*L2)

    {

     /\*\* Partie déclaration de variables \*\*/

      bool  Tous2 ;

     Pointeur\_Li Px1;

     Pointeur\_Li Px2;

     /\*\* Corps du module \*\*/

     if( \*L1 == NULL) {

       Tous2  =  True; }

     else

       {

       Px1 =  Valeur\_Li(\*L1 ) ;

       if( ! Rech ( & \*L2 , &Px1)) {

         Tous2  =  False; }

       else

         {

         Px2 =  Suivant\_Li(\*L1 ) ;

         Tous2  =  Tous2 ( &Px2, & \*L2 );

       }

     }

     return Tous2 ;

    }

int main(int argc, char \*argv[])

    {

     T\_L1 [ 0 ] = 2 ;

     T\_L1 [ 1 ] = 5 ;

     T\_L1 [ 2 ] = 9 ;

     T\_L1 [ 3 ] = 8 ;

     T\_L1 [ 4 ] = 3 ;

     T\_L1 [ 5 ] = 6 ;

     L1 = malloc(sizeof(Typestr\_Li));

     Creer\_liste\_Li (&L1 , T\_L1 , 6 )  ;

     T\_L2 [ 0 ] = 12 ;

     T\_L2 [ 1 ] = 5 ;

     T\_L2 [ 2 ] = 19 ;

     T\_L2 [ 3 ] = 8 ;

     T\_L2 [ 4 ] = 3 ;

     T\_L2 [ 5 ] = 6 ;

     T\_L2 [ 6 ] = 2 ;

     T\_L2 [ 7 ] = 9 ;

     L2 = malloc(sizeof(Typestr\_Li));

     Creer\_liste\_Li (&L2 , T\_L2 , 8 )  ;

     printf ( " %d ", Tous(&L1,&L2) );

     system("PAUSE");

     return 0;

    }

1. **Langage Z (Base)**

**Généralités sur le langage Z**

> Un Z-algorithmeIDH\_struct est un ensemble de modules parallèles dont le premier est principal et les autres sont soient des actions composéesIDH\_act soient des fonctionsIDH\_fonct.

> La communication entre les modules se fait via les paramètres et/ou les variables globales.

> Les objets globaux sont définis dans le module principal.

> Le langage permet tout type de paramètres : scalaires, structures, file, vecteur, ..., et mêmes les types complexes.

> Le langage Z admet les modules récursifs.

> Le langage permet l'allocation dynamique de tableaux et de structures.

> Le langage permet l'affectation globale de tout type.

> Quatre types standards (scalaires) sont autorisés : **ENTIER**, **BOOLEEN**, **CARACTERE**, **CHAINE**.

> Certaines fonctions usuelles sont prédéfinies : **MOD**, **MIN**, **MAX** et **EXP**IDH\_foncts1.

> Le langage est l'ensemble des algorithmes abstraits, écrits à base de modèles ou machines abstraites.

> On définit ainsi des machines abstraites sur

- les structuresIDH\_mstruct ou objets composés

- les vecteursIDH\_mvect

- les listes mono directionnellesIDH\_mlistes

- les listes bidirectionnellesIDH\_mlistebi

- les pilesIDH\_mpile

- les files d'attenteIDH\_mfile

- les arbres de recherche binaireIDH\_marb

- les arbres de recherche m-aireIDH\_marm

- les fichiersIDH\_mfich

> Le langage permet les types composés du genre **PILE de FILES de LISTES** ... dont la dernière est de type scalaire ou structure simple.

> Le langage peut être étendu avec d'autres machines abstraites dans les futures versions.

> Le langage est doté des opérations de haut niveau permettant de construire des listes, des arbres, des files, ... à partir d'un ensemble de valeurs ( expressions entières )

> Le langage offre deux fonctions très utiles permettant de générer aléatoirement des chaînes de caractères (**ALEACHAINE**IDH\_foncts2) et des entiers (**ALEANOMBRE**IDH\_foncts2).

> Le langage offre également deux fonctions pour la manipulation des chaînes de caractères (**LONGCHAINE** et **CARACT**IDH\_foncts3).

> Le langage permet la lecture et l'écriture de scalaires, de vecteurs de n'importe quelle dimension et des structures simples ou complexes.

> Le format d'écriture est libre.

**Structure d'un Z-algorithme**

**SOIENT**

Objets locaux et globaux

Annonce des modules

**DEBUT**

Instructions

**FIN**

Module 1

Module 2

...

Modules n

Chaque module est soit une action composée soit une fonction.

**Structures de contrôle**

La boucle TANTQUE

**TANTQUE** Exp[ : ]

Instructions

**FINTANTQUE**

La boucle POUR

**POUR** V : = Exp1, Exp2 [,Exp3] [ : ]

Instructions

**FINPOUR**

Exp3 désigne l'incrément du pas. Par défaut, sa valeur est 1.

La conditionnelle

**SI** Exp [ : ]

Instructions

**FINSI**

L'alternative

**SI** Exp [ : ]

Instructions

**SINON**

Instructions

**FINSI**

**Actions élémentaires**

Affectation

V := Exp

Affecte la valeur d'une expression à une variable.

Lecture

**LIRE** ( V1, V2, .... )

Introduit des données dans les variables V1, V2,...

Ecriture

**ECRIRE** ( Exp1, Exp2, ... )

Restitue les expressions Exp1, Exp2,....

(Expressions entières ou chaînes de caractères)

**Opérations de haut niveau**

Elles permettent de remplir une structure de données (ou initialiser une machine) à partir d'un ensemble d'expressions.

**INIT\_VECT** ( T, [Exp1, Exp2, ....] )

**INIT\_STRUCT** ( S, [Exp1, Exp2, ....] )

**CREER\_LISTE** ( L, [Exp1, Exp2, ....] )

**CREER\_LISTEBI** ( LB, [Exp1, Exp2, ....] )

**CREER\_ARB** ( A, [Exp1, Exp2, ....] )

**CREER\_ARM** ( M, [Exp1, Exp2, ....] )

**CREER\_FILE** ( F, [Exp1, Exp2, ....] )

**CREER\_PILE** ( P, [Exp1, Exp2, ....] )

Exp1, Exp2, .... sont des expressions scalaires ou des structures simples.

Exemple

**CREER\_LISTE** ( L, [12, 34, I, I+J, 45] )

Crée la liste linéaire chaînée L avec les valeurs entre crochets dans l'ordre indiqué.

**Fonctions standards**

**MOD** (A, B)

Reste de la division entière de A par B.

**MAX**(A, B)

Maximum entre A et B.

**MIN**(A, B)

Minimum entre A et B.

**EXP**(A, B)

A exposant B

**Fonctions de génération aléatoire**

**ALEACHAINE** ( N )

Fournit une chaîne aléatoire de N caractères parmi les caractères alphabétiques majuscule et minuscule.

**ALEAENTIER** ( N )

Fournit un nombre aléatoire entre 0 et N

**Fonctions sur chaînes de caractères**

**LONGCHAINE** ( C )

Donne la longueur de la chaîne C

**CARACT**(C, I)

Fournit le I-ième caractère de la chaîne C

**Définition d'une action composée**

**ACTION** Nom ( P1, P2, ...Pn )

Objets locaux et paramètres

**DEBUT**

Instructions

**FIN**

Les paramètres sont appelés par "référence", ce qui implique que les paramètres d'entrée ne sont pas protégés par l'action.

Une action composée doit être annoncée dans le module principal.

L'appel à une action se fait par

**APPEL** nom de l'action (paramètres réels)

**Définition d'une fonction**

**FONCTION** Nom ( P1, P2, ...Pn ) : type

Objets locaux et paramètres

**DEBUT**

Instructions

**FIN**

Les paramètres sont appelés par "référence", ce qui implique que les paramètres d'entrée ne sont pas protégés par l'action.

Une fonction utilisateur doit être annoncée dans le module principal en précisant son type.

Il doit y exister dans le corps d'une fonction une affectation du genre

Nom := Expression.

**Scalaires**

Quatre types standards sont autorisés : **ENTIER**, **CARACTERE**, **CHAINE**, **BOOLEEN**.

Définition des scalaires

[**SOIT**/**SOIENT**] <li> Sep Type

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { **:**, **UN**, **UNE**, **DES** }

Type est un type standard.

Exemples

A, B, Trouv : **BOOLEENS** ;

X, Y, Z **DES BOOLEENS** ;

A : **ENTIERS** ;

X, Y, Z **DES CHAINES** ;

**Objets "Pointeurs"**

On définit des variables de type "Pointeur" pour la manipulation des structures de données.

[**SOIT**/**SOIENT**] <li> Sep

**POINTEUR VERS** [sep] [ **DE** Typec **DE** Typec **DE** ....] [ **DE** Types]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { :, **UN**, **UNE**, **DES** }

Typec dans { **vecteur**, **pile**, **liste**, **file**, **pile**, **arb**, **listebi**, **arm** }

Types est un scalaire ou une structure simple.

Remarque

On peut s'en passer de ce type. En effet, les déclarations

"**Soit** L **une liste**" et

"**soit** L **un pointeur vers une liste**" sont équivalentes.

Exemples

P1 : **POINTEUR VERS LISTE DE PILE**;

P2, P2 **DES POINTEURS VERS DES ARB**;

**Expressions Z**

Comme dans les langages de programmation.

Expressions arithmétiques : + , - , / , \*

Expressions logiques : **ET**, **OU**, **NON**

Expressions sur chaînes de caractères : +

Expressions relationnelles : <, <=, >, >=, =, <> (ou #)

Constantes logiques : **VRAI**, **FAUX**

Constante pointeur : **NIL**

Exemples

B+C / F

**NON** Trouv

(X # 5) **ET NON** TROUV

F(X) <> 5

P = **Nil**

**Commentaires**

Les commentaires peuvent être insérés dans tout endroit oŚ on peut avoir un blanc.

Les commentaires sont entre { et } ou entre /\* et \*/.

**Exemple d'un Z-algorithme**

**SOIENT**

L1, L2 **DES LISTES**;

Rech, Tous : **FONCTION**(**BOOLEEN**);

**DEBUT**

**CREER\_LISTE**(L1, [2, 5, 9, 8, 3, 6 ]);

**CREER\_LISTE**(L2, [12, 5, 19, 8, 3, 6, 2,9]);

**ECRIRE**( Tous(L1, L2) )

**FIN**

/\* Recherche de la valeur Val dans la liste L \*/

**FONCTION** Rech ( L, Val ) : **BOOLEEN**

**SOIENT**

L **UNE LISTE**;

Val **UN ENTIER**;

**DEBUT**

**SI** L = **NIL** : Rech := **FAUX**

**SINON**

**SI VALEUR**(L) = Val

Rech := **VRAI**

**SINON**

Rech := Rech(**SUIVANT**(L), Val )

**FSI**

**FSI**

**FIN**

/\* Détermine si tous les éléments de L1 sont dans L2 \*/

**FONCTION** Tous ( L1, L2 ) : **BOOLEEN**

**SOIENT**

L1, L2 **DES LISTES**;

**DEBUT**

**SI** L1 = **NIL**

Tous := **VRAI**

**SINON**

**SI NON** Rech(L2, **VALEUR**(L1) )

Tous := **FAUX**

**SINON**

Tous := Tous(**SUIVANT**(L1), L2)

**FSI**

**FSI**

**FIN**

1. **Langage Z (Structures de données)**

**Structures**

Une structure est un ensemble d'éléments hétérogènes.

Une structure peut être simple, c'est à dire composée uniquement de scalaires.

Une structure peut être complexe, c'est à dire composée de scalaires et/ou de vecteurs à une dimension de scalaires.

Une structure peut être statique ou dynamique.

Définition des structures

[**SOIT**/**SOIENT**] <Li> Sep

[**STRUCTURE**] ( Type1, type2, ... ) [**DYNAMIQUE**]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { :, **Un**, **Une**, **Des** }

Typei est soit un type scalaire soit un tableau à une dimension de scalaires.

Exemples

S1 : (**ENTIER**, **CHAINE**) ;

S2 **UNE STRUCTURE** ( **CHAINE**, **ENTIER**, **BOOLEEN**) ;

S3 **UN** ( **ENTIER**, **VECTEUR**(5) **DE CHAINE**) **DYNAMIQUE**;

**Tableaux**

Un tableau est un ensemble d'éléments homogènes.

Un tableau ( ou vecteur) peut être simple, c'est à dire composé uniquement de scalaires.

Un tableau peut être complexe, c'est à dire composé de structures simples.

Un tableau peut être statique ou dynamique.

Définition des tableaux

[**SOIT**/**SOIENT**] <Li> sep

**VECTEUR**(Dim1, Dim2, ... )

**DE** Typec **DE** Typec **DE** .... **DE** Types [**DYNAMIQUE**]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans {:, **Un**, **Une**, **Des** }

Typec dans { **vecteur, pile, liste, file, pile, arb, listebi, arm** }

Types est un scalaire ou une structure simple.

Exemples

V1 **UN TABLEAU** (5) **DYNAMIQUE**;

V2, V3 **DES VECTEURS** (3, 8) **DE CHAINES** ;

**Listes linéaires chaînées**

Une liste linéaire chaînée est un ensemble de maillons alloués dynamiquement.

Un élément possède deux champs : Valeur et Adresse.

Le champ 'Valeur' peut être quelconque.

Définition des listes

[**SOIT**/**SOIENT**] <Li> Sep **LISTE** [ **DE** Typec **DE** Typec **DE** .... ] [ **DE** Types ]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans {:, **Un**, **Une**, **Des** }

Typec dans { **vecteur**, **pile**, **liste**, **file**, **pile**, **arb**, **listebi**, **arm** }

Types est un scalaire ou une structure simple.

Exemples

L1 **UNE LISTE DE** (**CHAINE**, **ENTIER**);

L2 **UNE LISTE DE PILE DE CHAINE** ;

L3 **UNE LISTE DE CHAINES**;

**Listes linéaires chaînées bilatérales**

Une liste linéaire chaînée bilatérale est un ensemble de maillons alloués dynamiquement qui peut être parcourue dans les deux sens.

Un élément possède trois champs : Valeur, adresse gauche, adresse droite.

Le champ 'Valeur' peut être quelconque.

Définition des listes bilatérales

[**SOIT**/**SOIENT**] <Li> Sep **LISTEBI** [**DE** Typec **DE** Typec **DE** .... ][ **DE** Types]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { **:**, **Un**, **Une**, **Des** }

Typec dans { **vecteur, pile, liste, file, pile, arb, listebi, arm** }

Types est un scalaire ou une structure simple.

Exemples

Lb1 **UNE LISTEBI DE** (**CHAINE**, **ENTIER**);

Lb2 **UNE LISTEBI DE PILE DE CHAINE** ;

Lb3 **UNE LISTEBI DE CHAINES**;

**Files d'attente**

Une file d'attente est une collection d'éléments dans laquelle tout nouvel élément est inséré à la fin et tout retrait se fait du début. C’est le principe FIFO : First In First Out

Un élément peut être quelconque

Définition des files d'attente

[**SOIT**/**SOIENT**] <Li> Sep **FILE** [**DE** Typec **DE** Typec **DE** .... ] [**DE** Types]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { **:**, **Un**, **Une**, **Des** }

Typec dans { **vecteur**, **pile**, **liste**, **file**, **pile**, **arb**, **listebi**, **arm** }

Types est un scalaire ou une structure simple.

Exemples

F1 **UNE FILE DE** (**CHAINE**, **ENTIER**);

F2 **UNE FILE DE PILE DE CHAINE** ;

F3 **UNE FILE DE CHAINES**;

**Piles**

Une pile est une collection d'éléments dans laquelle tout nouveau élément est inséré à la fin et tout retrait se fait également de la fin. C'est le principe LIFO : Last In First Out.

Un élément peut être quelconque.

Définition des piles

[**SOIT**/**SOIENT**] <Li> Sep **PILE** [**DE** Typec **DE** Typec **DE** .... ][ **DE** Types]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { **:**, **Un**, **Une**, **Des** }

Typec dans { **vecteur**, **pile**, **liste**, **file**, **pile**, **arb**, **listebi**, **arm** }

Types est un scalaire ou une structure simple.

Exemples

P1 **UNE PILE DE** (**CHAINE**, **ENTIER**);

P2 **UNE PILE DE PILE DE CHAINE** ;

P3 **UNE PILE DE CHAINES**;

**Arbres de recherche binaire**

Un arbre de recherche binaire est une structure de données généralement dynamique non linéaire.

Un nœud d'un arbre de recherche binaire contient une information et deux fils.

Structure d'un nœud : (a1, V1, a2)

Toutes les données rangées dans le sous arbre a1 sont strictement inférieures à V1.

Toutes les données rangées dans le sous arbre a2 sont strictement supérieures à V1.

Un élément (nœud) peut être quelconque.

Définition des arbres de recherche binaire

[**SOIT**/**SOIENT**] <Li> Sep **ARB** [**DE** Typec **DE** Typec **DE** .... ] [**DE** Types]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { **:**, **Un**, **Une**, **Des** }

Typec dans { **vecteur**, **pile**, **liste**, **file**, **pile**, **arb**, **listebi**, **arm** }

Types est un scalaire ou une structure simple.

Exemples

A1 **UN ARB DE** (**CHAINE**, **ENTIER**);

A2 **UN ARB DE PILE DE CHAINE** ;

A3 **UN ARB DE CHAINES**;

**Arbres de recherche m-aire**

Un arbre de recherche m-aire est une structure de données généralement dynamique non linéaire.

C'est une généralisation de l'arbre de recherche binaire.

Un nœud d'un arbre de recherche m-aire d'ordre p contient (p-1) informations et p fils.

Structure d'un nœud : (a1, V1, a2, V2, ......,Vp-1, ap)

Toutes les données rangées dans le sous arbre a1 sont strictement inférieures à V1.

Toutes les données rangées dans le sous arbre ap sont strictement supérieures à Vp-1.

Toutes les données rangées dans le sous arbre ai ( 1 < i < p ) sont strictement inférieures à Vi et strictement supérieures à Vi+1.

Un élément (nœud) peut être quelconque.

Définition des arbres de recherche m-aire

[**SOIT**/**SOIENT**] <Li> Sep **ARM** (degre) [ **DE** Typec **DE** Typec **DE** ....] [ **DE** Types]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { **:**, **Un**, **Une**, **Des** }

Typec dans { **vecteur, pile, liste, file, pile, arb, listebi, arm** }

Types est un scalaire ou une structure simple.

Exemples

M1 **UN ARM**(4) **DE** (**CHAINE**, **ENTIER**);

M2 **UN ARM**(2) **DE PILE DE CHAINE** ;

M3 **UN ARM**(3) **DE CHAINES**;

**Fichiers**

Un fichier est un ensemble d'enregistrements ( ou structures ) rangés généralement sur un disque.

Les enregistrements peuvent être des articles pour un utilisateur.

Les enregistrements peuvent être des blocs pour un concepteur.

Le fichier renferme une partie indispensable ( En-tête )pour la conception de structures de fichiers.

Définition des fichiers

[**SOIT**/**SOIENT**] <Li> sep **FICHIER DE** type

**BUFFER** <Li>

[**ENTETE** ( Type1, Type2, .... ) ]

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans { :, **Un**, **Une**, **Des** }

Une définition de fichier comporte 3 parties :

La première partie (**FICHIER**) précise la nature des éléments du fichier.

Un élément (article ou bloc) du fichier peut être

- un scalaire

- un vecteur à une dimension de scalaires

- une structure complexe ( pouvant contenir des scalaires et/ou des vecteurs à une dimension de scalaires)

La deuxième partie (**BUFFER**) définit les variables Tampon utilisées dans les opérations de lecture/écriture.

La troisième partie (**ENTETE**) définit les caractéristiques du fichier en précisant le type de chacune d'entre elles. Cette partie est facultative et est surtout utilisée pour la création de structures de fichiers. Elle sert à mémoriser toutes les informations utiles pour l'exploitation du fichier.

Exemples

F1 **UN FICHIER DE CHAINES BUFFER** V1, V2;

F2 **UN FICHIER DE VECTEUR**(5) **DE ENTIER BUFFER** V ;

F3 **UN FICHIER DE** (**ENTIER**, **VECTEUR**(3) **DE CAR**) **BUFFER** V **ENTETE**(**ENTIER**,**ENTIER**) ;

F4 **UN FICHIER DE CAR BUFFER** V **ENTETE** (**ENTIER**, **CHAINE**, **BOOLEEN**) ;

1. **Machines abstraites**

Sur chaque structure de données, une machine abstraite est définie avec son ensemble d'opérations.

[**SOIT**/**SOIENT**] <li> Sep <Machine abstraite>

<Li> : Liste d'identificateurs séparés par des virgules.

Sep dans {:, **Un**, **Une**, **Des** }

Exemples

L1, L2 **DES LISTES**;

F **UNE FILE**;

V1 **UN VECTEUR**(10, 60);

Y **UNE LISTE DE PILES DE VECTEUR**(5);

**Machine abstraite sur les vecteurs**

**ELEMENT** ( T [i, j, ...] )

Accès à l'élément T[i, j, ...]du vecteur T.

**AFF\_ELEMENT** ( T [I, J, ...], Val )

Affecter à l'élément T[i, j, ...] la valeur Val.

**ALLOC\_TAB** ( T )

Allocation d'un tableau de taille spécifiée par la définition de T.L'adresse est rendue dans la variable T.

**LIBER\_TAB** ( T )

Libération de l'espace mémoire pointé par T.

**Machine abstraite sur les structures**

**STRUCT** ( S, i)

Accès au i-ème champ de la structure S

**AFF\_STRUCT** ( S, i, Exp )

Affecter à la structure S dans son i-ème champ l'expression Exp.

**ALLOC\_STRUCT** ( S )

Allocation d'un espace mémoire de taille spécifiée par la définition de S. L'adresse est rendue dans la variable S.

**LIBER\_STRUCT** ( S )

Libération de l'espace mémoire pointé par S.

**Machine abstraite sur les listes linéaires chaînées**

**ALLOUER** ( P )

Crée un maillon et retourne son adresse dans P.

**LIBERER** ( P )

Libère le nœud d'adresse P.

**SUIVANT** ( P )

Accès au champ 'Adresse' du nœud référencé par P.

**VALEUR** ( P )

Accès au champ 'Valeur' du nœud référencé par P.

**AFF\_ADR** ( P, Q )

Affecter au champ 'Adresse' du nœud référencé par P, l'adresse Q.

**AFF\_VAL**( P, Val )

Affecter au champ 'Valeur' du nœud référencé par P, la valeur Val.

**Machine abstraite sur les listes bidirectionnelles**

**ALLOUER** ( P )

Crée un maillon et retourne son adresse dans P.

**LIBERER** ( P )

Libère le nœud d'adresse P.

**SUIVANT** ( P )

Accès au champ 'Adresse droite' du nœud référencé par P.

**PRECEDENT** ( P )

Accès au champ 'Adresse gauche' du nœud référencé par P.

**VALEUR** ( P )

Accès au champ 'Valeur' du nœud référencé par P.

**AFF\_ADRD** ( P, Q )

Affecter au champ 'Adresse droite' du nœud référencé par P, l'adresse Q

**AFF\_ADRG** ( P, Q )

Affecter au champ 'Adresse gauche' du nœud référencé par P, l'adresse Q.

**AFF\_VAL**( P, Val )

Affecter au champ 'Valeur' du nœud référencé par P, la valeur Val.

**Machine abstraite sur les files d'attente**

**CREERFILE** ( F )

Crée une file d'attente vide.

**FILEVIDE** ( F)

Teste si une file d'attente est vide.

**ENFILER** ( F, Val )

Enfiler (rajouter en queue) la valeur Val dans la file d'attente F.

**DEFILER** ( F, Val )

Défiler ( récupérer de la tête ) une valeur pour la mettre dans Val.

**Machine abstraite sur les piles**

**CREERPILE** ( P )

Crée une pile vide.

**PILEVIDE** ( P )

Teste si une pile est vide.

**EMPILER** ( P, Val )

Empiler (rajouter au sommet) la valeur Val dans la pile P.

**DEPILER** ( P, Val )

Dépiler ( récupérer du sommet ) une valeur pour la mettre dans Val.

**Machine abstraite sur les arbres de recherche binaire**

**CREERNOEUD** ( Val )

Crée un nœud avec l'information Valet retourne l'adresse du nœud. Les autres champs sont à NIL.

**LIBERERNOEUD** ( P )

Libère le nœud d'adresse P.

**FG** ( P )

Accès au champ Fils gauche du nœud référencé par P.

**FD** ( P )

Accès au champ Fils droit du nœud référencé par P.

**PERE** ( P )

Accès au champ Père du nœud référencé par P.

**INFO** ( P )

Accès au champ Info du nœud référencé par P.

**AFF\_FG** ( P, Q )

Affecter au champ Fils gauche du nœud référencé par p, l'adresse Q

**AFF\_FD** ( P, Q )

Affecter au champ Fils droit du nœud référencé par p, l'adresse Q

**AFF\_PERE** ( P, Q )

Affecter au champ Père du nœud référencé par p, l'adresse Q

**AFF\_INFO**( P, Val )

Affecter au champ Info du nœud référencé par p, la valeur Val

**Machine abstraite sur les arbres de recherche m-aire**

**CREERNOEUD** ( Val )

Crée un nœud avec l'information Val et retourne l'adresse du nœud. Les autres champs sont à NIL.

**LIBERERNOEUD** ( P )

Libère le nœud d'adresse P.

**FILS** ( P, I )

Accès au champ I-ième fils du nœud référencé par P.

**PERE** ( P )

Accès au champ Père du nœud référencé par P.

**INFOR** ( P, I )

Accès au I-ième champ Info du nœud référencé par P.

**AFF\_FILS** ( P, I, Q )

Affecter au champ I-ième fils du nœud référencé par P, l'adresse Q.

**AFF\_PERE** ( P, Q )

Affecter au champ Père du nœud référencé par P, l'adresse Q.

**AFF\_INFOR**( P, I, Val )

Affecter au I-ième champ Information du nœud référencé par P, la valeur Val.

**Machine abstraite sur les fichiers**

**OUVRIR** (Fl, Fp, Mode)

Ouvrir le fichier logique Fl et l'associer au fichier physique Fp en précisant le mode( fichier nouveau('N') ou ancien 'A') )

**FERMER** (Fl)

Fermer le fichier Fl.

**LIRESEQ** (Fl, V)

Lire dans la variable tampon V le bloc (ou l'article)se trouvant à la position courante.

**ECRIRESEQ** (Fl, V)

Ecrire le contenu de la variable tampon V à la position courante du fichier Fl.

**LIREDIR** (Fl, V, N) :

Lire le N-ième bloc (ou article) du fichier Fl dans la variable tampon V.

**ECRIREDIR** (Fl, V, N)

Ecrire le contenu de la variable tampon V à la N-ième position du fichier Fl.

**RAJOUTER**(Fl, V)

Ecrire le contenu de la variable tampon à la fin du fichier Fl.

**FINFICH**(Fl)

Prédicat égal à vrai si la fin du fichier Fl est rencontrée, faux sinon.

**ALLOC\_BLOC**(Fl)

Fournit un bloc (ou article) du fichier dans lequel on pourra écrire.

**ENTETE**(Fl, I)

Récupérer la I-ième caractéristique du fichier Fl.

**AFF\_ENTETE**(Fl, I, Exp)

Affecter Exp comme la I-ème caractéristique du fichier.

1. **Passage de Z vers PASCAL**

**Déclaration des variables**

Une déclaration de variables PASCAL se fait par

VAR <Li> : Type;

où <Li> désigne une liste d'identificateurs.

**SOIT** (**SOIENT**) se traduit par VAR.

Les objets simples

Equivalents des objets Z --> PASCAL

Z PASCAL

**ENTIER** INTEGER

**BOOLEEN** BOOLEAN

**Boucle "TANTQUE"**

-------Z---------

**TANTQUE** Cond :

instructions

**FINTANTQUE**

-------PASCAL------

WHILE ( Cond ) DO

BEGIN

Instructions

END

**Boucle "POUR"**

--------------Z---------------

**POUR** V:= Exp1, Exp2 [, Exp3] :

Instructions

**FINPOUR**

Si Exp3 est absent ou égale à 1

Se traduit par :

--------PASCAL---------

FOR V:= Exp1 TO Exp2 DO

BEGIN

Instructions

END

Si Exp3 <> 1 :

----------PASCAL------

V := Exp1;

WHILE ( V <= Exp2 ) DO

BEGIN

Instructions ;

V := V + Exp3

END;

**L'alternative "SI"**

----------Z----------

**SI** Cond :

Instructions

[**SINON**

Instruction ]

**FSI**

Se traduit par :

-------PASCAL---------

IF Cond

THEN

BEGIN

Instructions

END

[ELSE

BEGIN

Instructions

END]

**Z-expressions**

La grammaire des Z-expressions est incluse dans la grammaire PASCAL.

**Lecture**

--------Z--------

**LIRE**(V1, V2, ...)

Se traduit par :

------PASCAL------

READLN(V1, V2, ...)

**Ecriture**

-----------Z-----------

**ECRIRE**(Exp1, Exp2, ...)

Se traduit par

--------PASCAL----------

WRITELN(Exp1, Exp2, ...)

**Affectation**

Même syntaxe

**Action composée**

------------Z-----------------

**ACTION** Nom ( P1, P2, ...)

**SOIENT**

Définition des objets locaux

et des paramètres

**DEBUT**

Instructions

**FIN**

Se traduit par :

--------------------PASCAL--------------------

PROCEDURE Nom ( VAR P1: typ; VAR P2:typ, ...);

VAR

Définition des objets locaux

BEGIN

Instructions

END

**Fonctions**

----------------Z-----------------

**FONCTION** Nom ( P1, P2, ...) : Type

**SOIENT**

Définition des objets locaux

et des paramètres

**DEBUT**

Instructions

**FIN**

Se traduit par :

---------------------PASCAL------------------------

FUNCTION Nom ( VAR P1: typ; VAR P2:typ, ...) : Type;

VAR

Définition des objets locaux

BEGIN

Instructions

END

**Fonctions prédéfinies**

**MOD** (a, b)

FUNCTION Mod (a, b : INTEGER) : INTEGER;

BEGIN

Mod := a Mod b

END;

**MIN** (a, b)

FUNCTION Min (a, b: INTEGER) : INTEGER;

BEGIN

Min := a; IF b < a THEN Min := b;

END;

**MAX** ( a, b )

FUNCTION Max (a, b: INTEGER) : INTEGER;

BEGIN

Max := a; IF b > a THEN Max := b;

END;

**EXP** ( a, b )

FUNCTION Exp (a, b: INTEGER) : INTEGER;

VAR I : INTEGER;

BEGIN

Exp := 1;

FOR I:= 1 TO b DO Exp := Exp \* a

END;

**ALEANOMBRE** ( N )

FUNCTION Aleanombre (N: INTEGER) : INTEGER;

BEGIN

Aleanombre := Random( N );

END;

**ALEACHAINE** ( N )

FUNCTION Aleachaine(N: INTEGER) : STRING;

VAR

K : BYTE;

Chaine : STRING;

BEGIN

Chaine := '';

FOR K:=1 TO N DO

CASE Random(2) OF

0 : Chaine := Chaine + CHR(97+Random(26) ) ;

1 : Chaine := Chaine + CHR(65+Random(26) )

END;

Aleachaine := Chaine;

END;

**LONGCHAINE** ( C )

FUNCTION Longchaine(C : STRING): INTEGER;

BEGIN

Min := a; IF b < a THEN Min := b;

END;

**Algorithme**

--------------------Z-----------------

**SOIENT**

Objets locaux et globaux

Annonce des modules

**DEBUT**

Instructions

**FIN**

Module 1

Module 2

...

Modules n

Se traduit par :

------------------PASCAL---------------

PROGRAM Pascal;

VAR

Objets locaux et globaux

{ Définition des modules }

Module 1

Module 2

...

Module n

BEGIN

Instructions

END.

1. **Implémentation des machines Z en PASCAL**

**Implémentation des vecteurs en PASCAL**

**soit Tab un TABLEAU (5, 10);**

{ Tableaux }

  TYPE

    Typeelem\_V5\_10I = INTEGER;

     Typetab\_V5\_10I = ARRAY[1..5,1..10] OF Typeelem\_V5\_10I;

    Typevect\_V5\_10I = ^ Typetab\_V5\_10I;

  FUNCTION Element\_V5\_10I ( V:Typevect\_V5\_10I; I1 , I2 : INTEGER ) : Typeelem\_V5\_10I ;

    BEGIN

      Element\_V5\_10I :=  V^[I1  ,I2];

    END;

  PROCEDURE Aff\_element\_V5\_10I ( V :Typevect\_V5\_10I; I1 , I2 :INTEGER; Val : Typeelem\_V5\_10I );

    BEGIN

      V^[I1  ,I2] := Val;

    END;

   {Partie déclaration de variables }

   VAR

      Tab : Typevect\_V5\_10I;

   {Corps du programme principal }

   BEGIN

      NEW(Tab);

   END.

**Implémentation des structures en PASCAL**

**soit S une structure (chaine, entier);**

TYPE Typestring = STRING[255];

  { Structures }

  TYPE

   Type1\_TSI  =  Typestring;

   Type2\_TSI  =  INTEGER;

   Typestr\_TSI  = ^ Type\_TSI ;

   Type\_TSI = record

      Champ1 : Type1\_TSI ;

      Champ2 : Type2\_TSI ;

    END;

  FUNCTION STRUCT1\_TSI ( S: Typestr\_TSI) : Type1\_TSI;

    BEGIN

      STRUCT1\_TSI :=  S^.champ1;

    END;

  FUNCTION STRUCT2\_TSI ( S: Typestr\_TSI) : Type2\_TSI;

    BEGIN

      STRUCT2\_TSI :=  S^.champ2;

    END;

  PROCEDURE AFF\_STRUCT1\_TSI ( S: Typestr\_TSI; Val :Type1\_TSI  );

    BEGIN

      S^.champ1 := Val;

    END;

  PROCEDURE AFF\_STRUCT2\_TSI ( S: Typestr\_TSI; Val :Type2\_TSI  );

    BEGIN

      S^.champ2 := Val;

    END;

   {Partie déclaration de variables }

   VAR

      S : Typestr\_TSI;

   {Corps du programme principal }

   BEGIN

      NEW(S);

   END.

**Implémentation des listes linéaires chaînées en PASCAL**

**SOIT L UNE LISTE ;**

{ Listes linéaires chaînées }

  TYPE

    Typeelem\_LI = INTEGER;

    Pointeur\_LI = ^Maillon\_LI; { type du champ 'Adresse' }

    Maillon\_LI = RECORD

        Val : Typeelem\_LI;

        Suiv : Pointeur\_LI

    END;

  PROCEDURE Allouer\_LI ( VAR P : Pointeur\_LI ) ;

    BEGIN  NEW(P)  END;

  PROCEDURE Liberer\_LI ( P : Pointeur\_LI ) ;

    BEGIN  DISPOSE(P)  END;

  PROCEDURE Aff\_val\_LI(P : Pointeur\_LI; Val : Typeelem\_LI );

    BEGIN P^.Val := Val  END;

  FUNCTION Valeur\_LI (P : Pointeur\_LI) : Typeelem\_LI;

    BEGIN   Valeur\_LI := P^.Val  END;

  FUNCTION Suivant\_LI( P : Pointeur\_LI) : Pointeur\_LI;

    BEGIN  Suivant\_LI := P^.Suiv END;

  PROCEDURE Aff\_adr\_LI( P, Q : Pointeur\_LI ) ;

    BEGIN P^.Suiv := Q    END;

   {Partie déclaration de variables }

   VAR

      L : Pointeur\_LI;

**Implémentation des listes bilatérales en PASCAL**

**soit Lb une listebi;**

{ Listes bidirectionnelles  }

  TYPE

    Typeelem\_RI = INTEGER;

    Pointeur\_RI = ^Maillon\_RI; { type du champ 'Adresse' }

    Maillon\_RI = RECORD

        Val : Typeelem\_RI;

        Suiv : Pointeur\_RI;

        Prec : Pointeur\_RI

    END;

  PROCEDURE Allouer\_RI ( VAR P : Pointeur\_RI ) ;

    BEGIN  NEW(P)  END;

  PROCEDURE Liberer\_RI ( P : Pointeur\_RI ) ;

    BEGIN  DISPOSE(P)  END;

  PROCEDURE Aff\_val\_RI (P : Pointeur\_RI; Val : Typeelem\_RI );

    BEGIN P^.Val := Val  END;

  FUNCTION Valeur\_RI (P : Pointeur\_RI) : Typeelem\_RI;

    BEGIN   Valeur\_RI := P^.Val  END;

  FUNCTION Suivant\_RI( P : Pointeur\_RI) : Pointeur\_RI;

    BEGIN  Suivant\_RI := P^.Suiv END;

  FUNCTION Precedent\_RI( P : Pointeur\_RI) : Pointeur\_RI;

    BEGIN  Precedent\_RI := P^.Prec END;

  PROCEDURE Aff\_adrd\_RI( P, Q : Pointeur\_RI ) ;

    BEGIN P^.Suiv := Q    END;

  PROCEDURE Aff\_adrg\_RI( P, Q : Pointeur\_RI ) ;

    BEGIN P^.Prec := Q    END;

   {Partie déclaration de variables }

   VAR

      Lb : Pointeur\_RI;

**Implémentation des arbres de recherche binaire en PASCAL**

**soit A un arb;**

{ Arbres de recherche binaire }

  TYPE

    Typeelem\_AI = INTEGER;

    Pointeur\_AI = ^Noeud;

    Noeud = RECORD

      Element : Typeelem\_AI;

      Fg, Fd, Pere  : Pointeur\_AI ;

    END;

  FUNCTION Info\_AI(P : Pointeur\_AI) : Typeelem\_AI;

    BEGIN  Info\_AI := P^.Element   END;

  FUNCTION Fg\_AI( P : Pointeur\_AI) : Pointeur\_AI;

    BEGIN  Fg\_AI := P^.Fg  END;

  FUNCTION Fd\_AI( P : Pointeur\_AI) : Pointeur\_AI;

    BEGIN  Fd\_AI := P^.Fd  END;

  FUNCTION Pere\_AI( P : Pointeur\_AI) : Pointeur\_AI;

    BEGIN  Pere\_AI := P^.Pere  END;

  PROCEDURE Aff\_info\_AI ( VAR P : Pointeur\_AI; Val : Typeelem\_AI);

    BEGIN   P^.Element := Val  END;

  PROCEDURE Aff\_fg\_AI( VAR P : Pointeur\_AI; Q : Pointeur\_AI);

    BEGIN   P^.Fg :=  Q  END;

  PROCEDURE Aff\_fd\_AI( VAR P : Pointeur\_AI; Q : Pointeur\_AI);

    BEGIN   P^.Fd :=  Q  END;

  PROCEDURE Aff\_pere\_AI( VAR P : Pointeur\_AI; Q : Pointeur\_AI);

    BEGIN   P^.pere :=  Q  END;

  PROCEDURE Creernoeud\_AI( VAR P : Pointeur\_AI) ;

    BEGIN

      NEW ( P ) ;

      P^.Fg := Nil;

      P^.fd := Nil

    END;

  PROCEDURE Liberernoeud\_AI( P : Pointeur\_AI);

    BEGIN

      DISPOSE ( P )

    END;

   {Partie déclaration de variables }

   VAR

      A : Pointeur\_AI;

**Implémentation des arbres de recherche m-aire en PASCAL**

**soit M un arm(4);**

{ Arbres de recherche m-aire }

  TYPE

    Typeelem\_M4I = INTEGER;

    Pointeur\_M4I = ^Noeud;

    Noeud = RECORD

      Infor : ARRAY[1..4] of Typeelem\_M4I;

      Fils  : ARRAY[1..4] of Pointeur\_M4I;

      Degre : Byte ;

      Parent : Pointeur\_M4I

    END;

  FUNCTION Infor\_M4I(P : Pointeur\_M4I; I: INTEGER) : Typeelem\_M4I;

    BEGIN  Infor\_M4I := P^.Infor[I]   END;

  FUNCTION Fils\_M4I( P : Pointeur\_M4I; I : INTEGER) : Pointeur\_M4I;

    BEGIN  Fils\_M4I := P^.Fils[I]  END;

  FUNCTION Parent\_M4I( P : Pointeur\_M4I) : Pointeur\_M4I;

    BEGIN  Parent\_M4I := P^.Parent  END;

  PROCEDURE Aff\_infor\_M4I ( P : Pointeur\_M4I; I:INTEGER; Val : Typeelem\_M4I);

    BEGIN   P^.Infor[I] := Val  END;

  PROCEDURE Aff\_fils\_M4I( P : Pointeur\_M4I; I:INTEGER; Q : Pointeur\_M4I);

    BEGIN   P^.Fils[I] :=  Q  END;

  PROCEDURE Aff\_parent\_M4I( P : Pointeur\_M4I; Q : Pointeur\_M4I);

    BEGIN   P^.parent :=  Q  END;

  PROCEDURE Creernoeud\_M4I(  VAR P : Pointeur\_M4I ) ;

    VAR

      I : BYTE;

    BEGIN

      NEW ( P ) ;

      For I:=1 TO 4 Do P^.Fils[I] := NIL;

      P.degre := 0

    END;

  FUNCTION Degre\_M4I ( P : Pointeur\_M4I ) : BYTE;

    BEGIN

      Degre\_M4I := P^.Degre

    END;

  PROCEDURE Aff\_Degre\_M4I ( VAR P : Pointeur\_M4I; N : BYTE);

    BEGIN

      P^.Degre := N

    END;

  PROCEDURE Liberernoeud\_M4I( P : Pointeur\_M4I);

    BEGIN

      DISPOSE ( P )

    END;

   {Partie déclaration de variables }

   VAR

      M : Pointeur\_M4I;

**Implémentation des piles en PASCAL**

**Soit P une pile ;**

     { Piles }

  TYPE

    Typeelem\_PI = INTEGER;  { type quelconque }

    Pointeur\_PI = ^Maillon\_PI ;

    Maillon\_PI = RECORD

      Valeur : Typeelem\_PI;

      Suivant : Pointeur\_PI

    END;

  PROCEDURE Creerpile\_PI( VAR P : Pointeur\_PI );

    BEGIN

      P := NIL;

    END;

  FUNCTION Pilevide\_PI ( P : Pointeur\_PI ) : BOOLEAN;

    BEGIN

      Pilevide\_PI := ( P = NIL )

    END;

  PROCEDURE Empiler\_PI ( VAR P : Pointeur\_PI; Val : Typeelem\_PI );

    VAR

      Q : Pointeur\_PI;

    BEGIN

      NEW(Q);

      Q^.Valeur := Val;

      Q^.Suivant := P;

      P := Q;

    END;

  PROCEDURE Depiler\_PI ( VAR P : Pointeur\_PI; VAR V :Typeelem\_PI );

    VAR Sauv : Pointeur\_PI;

    BEGIN

     IF NOT Pilevide\_PI (P)

     THEN

       BEGIN

         V := P^.Valeur;

         Sauv := P;

         P := P^.Suivant;

         DISPOSE(Sauv);

       END

     ELSE WRITELN('Pile Vide');

    END;

  {Partie déclaration de variables }

   VAR

      P : Pointeur\_PI;

**Implémentation des files d'attente en PASCAL**

**Soit F une File ;**

    { Files d'attente }

  TYPE

    Typeelem\_FI = INTEGER;

    Ptliste\_FI = ^Maillon\_FI;

    Maillon\_FI = RECORD

      Val  : Typeelem\_FI;

      Suiv : Ptliste\_FI

    END;

    Pointeur\_FI = ^ Filedattente\_FI;

    Filedattente\_FI = RECORD

      Tete, Queue : Ptliste\_FI

    END;

  PROCEDURE Creerfile\_FI (VAR Fil : Pointeur\_FI );

    BEGIN

      New (Fil);

      Fil^.Tete := NIL ;

      Fil^.Queue := Nil

    END;

  FUNCTION Filevide\_FI (Fil : Pointeur\_FI) : BOOLEAN;

    BEGIN  Filevide\_FI := Fil^.Tete = NIL  END;

  PROCEDURE Enfiler\_FI (VAR Fil : Pointeur\_FI; Val : Typeelem\_FI );

    VAR

      P : Ptliste\_FI;

    BEGIN

      NEW(P);

      P^.Val := Val;

      P^.Suiv := NIL;

      IF NOT Filevide\_FI(Fil)

      THEN Fil^.Queue^.Suiv := P

      ELSE Fil^.Tete := P;

      Fil^.Queue := P;

    END;

  PROCEDURE Defiler\_FI (VAR Fil : Pointeur\_FI ; VAR Val : Typeelem\_FI );

    BEGIN

      IF NOT Filevide\_FI(Fil)

      THEN

        BEGIN

          Val := Fil^.Tete^.Val;

          Fil^.Tete := Fil^.Tete^.Suiv;

        END

      ELSE WRITELN(' File Vide ');

    END;

 {Partie déclaration de variables }

   VAR

      F : Pointeur\_FI;

**Implémentation des fichiers en PASCAL**

**soit F un fichier de (chaines, entier) entete (ENTIER, entier)  buffer B1;**

PROGRAM Mon\_programme;

  Uses Sysutils;

  TYPE Typestring = STRING[255];

  { Implémentation : FICHIER }

  { Traitement des fichiers ouverts }

  TYPE

    \_Ptr\_Noeud = ^\_Noeud;

    \_Noeud = RECORD

       Var\_fich : Thandle;

       Nom\_fich : string;

       Sauv\_pos : Longint;

       Suiv : \_Ptr\_Noeud

    END;

  VAR

    \_Pile\_ouverts : \_Ptr\_Noeud = NIL;

  FUNCTION \_Ouvert (Fp : String) : \_Ptr\_Noeud;

    VAR

      P : \_Ptr\_Noeud;

      Trouv : boolean;

    BEGIN

      P := \_Pile\_ouverts; Trouv := False ;

      WHILE (P <> NIL) AND NOT Trouv DO

        IF P^.Nom\_Fich = Fp

        THEN Trouv := True

        ELSE P := P^.Suiv;

      \_Ouvert := P;

    END;

  PROCEDURE \_Empiler\_ouvert ( Fp : string; VAR Fl: Thandle);

    VAR

      P : \_Ptr\_Noeud ;

    BEGIN

      New(P);

      P^.Nom\_fich := Fp;

      P^.Var\_fich := Fl;

      P^.Suiv := \_Pile\_ouverts;

      \_Pile\_ouverts := P

    END ;

  FUNCTION \_Depiler\_ouvert ( Fl : Thandle) : String;

    VAR

      P, Prec : \_Ptr\_Noeud  ;

    BEGIN

      P:= \_Pile\_ouverts;

      Prec := Nil;

      WHILE P^.Var\_fich <> Fl DO

        BEGIN Prec := P ; P := P^.Suiv END;

      \_Depiler\_ouvert := P^.Nom\_fich ;

      IF Prec <> NIL

      THEN Prec^.Suiv := P^.Suiv

      ELSE \_Pile\_ouverts := P^.Suiv;

      Dispose (P);

    END;

  { Fichiers }

  TYPE

    { Types des champs du bloc}

    Typechamp1\_SIEII =  Typestring;

    Typechamp2\_SIEII =  INTEGER;

    { Définition de la structure du bloc du fichier }

    Typestruct\_SIEII = ^ Typestruct\_SIEII\_ ;

    Typestruct\_SIEII\_ = RECORD

      Champ1 : Typechamp1\_SIEII ;

      Champ2 : Typechamp2\_SIEII ;

    END;

    { Définition du bloc du fichier }

    Typestruct\_SIEII\_Buf = RECORD

      Champ1 : Typechamp1\_SIEII ;

      Champ2 : Typechamp2\_SIEII ;

    END;

    { Types des champs de l'en-tête}

    Typeentete1\_SIEII =  INTEGER;

    Typeentete2\_SIEII =  INTEGER;

    { Définition du bloc d'entete }

    Typestruct\_SIEII\_entete = RECORD

      Entete1 : Typeentete1\_SIEII ;

      Entete2 : Typeentete2\_SIEII ;

    END;

    Typestr\_TSI = Typestruct\_SIEII;

    Type\_TSI = Typestruct\_SIEII\_;    { Utilisation probable }

  { Manipulation de la structure (Buffer) }

  FUNCTION STRUCT1\_TSI ( Buf : Typestruct\_SIEII) : Typechamp1\_SIEII;

    BEGIN

      STRUCT1\_TSI :=  Buf^.champ1;

    END;

  FUNCTION STRUCT2\_TSI ( Buf : Typestruct\_SIEII) : Typechamp2\_SIEII;

    BEGIN

      STRUCT2\_TSI :=  Buf^.champ2;

    END;

  PROCEDURE AFF\_STRUCT1\_TSI ( Buf : Typestruct\_SIEII; Val :Typechamp1\_SIEII );

    BEGIN

      Buf^.champ1 := Val;

    END;

  PROCEDURE AFF\_STRUCT2\_TSI ( Buf : Typestruct\_SIEII; Val :Typechamp2\_SIEII );

    BEGIN

      Buf^.champ2 := Val;

    END;

  { Déclaration du buffer de l'en-tête }

  VAR

     Buf\_caract\_SIEII :  Typestruct\_SIEII\_entete ;

  { Opérations sur les fichiers }

  PROCEDURE Ouvrir\_SIEII (VAR Fl : Thandle ; Fp, Mode : STRING );

    VAR

      P : \_Ptr\_Noeud;

    BEGIN

      P :=  \_Ouvert (Fp);

      IF P <> NIL

      THEN

        BEGIN

          { Sauvegarder la position courante du fichier ouvert et le fermer }

          P^.Sauv\_pos :=FILESEEK(P^.Var\_fich, 0, 1);

          FILESEEK(P^.Var\_fich,0,0);

          FILEWRITE(P^.Var\_fich, Buf\_caract\_SIEII, sizeof(Buf\_caract\_SIEII) );

          FILECLOSE (P^.Var\_fich);

        END;

      { Ouvrir ou Ré ouvrir le fichier }

      IF Mode = 'A'

      THEN

        BEGIN

          Fl:=FILEOPEN(Fp,fmOpenReadWrite);

          FILEREAD(Fl, Buf\_caract\_SIEII, sizeof(Buf\_caract\_SIEII) )

        END

      ELSE

        BEGIN

          Fl:=FILECREATE(Fp);

          FILEWRITE(Fl, Buf\_caract\_SIEII, sizeof(Buf\_caract\_SIEII) )

        END ;

      \_Empiler\_ouvert(Fp, Fl);

    END;

  PROCEDURE Fermer\_SIEII ( VAR Fl : Thandle);

    VAR

      P : \_Ptr\_Noeud;

      Fp : String;

    BEGIN

      Fp := \_Depiler\_ouvert(Fl);

      FILESEEK(Fl,0, 0);

      FILEWRITE(Fl, Buf\_caract\_SIEII, sizeof(Buf\_caract\_SIEII) );

      FILECLOSE(Fl);

      { Ya-til un fichier ouvert avec le même nom ? }

      { Si Oui, le Réouvrir à la position sauvegardée }

      P :=  \_Ouvert (Fp);

      IF P <> NIL

      THEN

        BEGIN

          Fl:=FILEOPEN(P^.Nom\_fich,fmOpenReadWrite);

          FILEREAD(Fl, Buf\_caract\_SIEII, sizeof(Buf\_caract\_SIEII) );

          FILESEEK(Fl, P^.Sauv\_pos, 0)

        END;

    END;

  FUNCTION Entete1\_SIEII(  VAR Fl : Thandle): Typeentete1\_SIEII;

    BEGIN

      Entete1\_SIEII := Buf\_caract\_SIEII.Entete1;

    END;

  FUNCTION Entete2\_SIEII(  VAR Fl : Thandle): Typeentete2\_SIEII;

    BEGIN

      Entete2\_SIEII := Buf\_caract\_SIEII.Entete2;

    END;

  PROCEDURE Aff\_entete1\_SIEII ( VAR Fl: Thandle; VAL : Typeentete1\_SIEII);

    BEGIN

      Buf\_caract\_SIEII.Entete1 := VAL

    END;

  PROCEDURE Aff\_entete2\_SIEII ( VAR Fl: Thandle; VAL : Typeentete2\_SIEII);

    BEGIN

      Buf\_caract\_SIEII.Entete2 := VAL

    END;

  PROCEDURE Ecrireseq\_SIEII ( VAR Fl: Thandle;  Buf : Typestruct\_SIEII );

    VAR

      Buffer : Typestruct\_SIEII\_Buf ;

      I : Integer;

    BEGIN

      Buffer.Champ1:= Buf^.Champ1;

      Buffer.Champ2:= Buf^.Champ2;

      FILEWRITE(Fl, Buffer, Sizeof(Buffer))

    END;

PROCEDURE Ecriredir\_SIEII ( VAR Fl: Thandle;  Buf : Typestruct\_SIEII;  N: INTEGER );

    VAR

      Buffer : Typestruct\_SIEII\_Buf ;

      I : Integer;

    BEGIN

      Buffer.Champ1:= Buf^.Champ1;

      Buffer.Champ2:= Buf^.Champ2;

      FILESEEK(Fl, Sizeof(Buf\_caract\_SIEII) + (N-1)\*Sizeof(Buffer ), 0);

      FILEWRITE(Fl, Buffer, Sizeof(Buffer))

    END;

  PROCEDURE Lireseq\_SIEII ( VAR Fl: Thandle; VAR Buf : Typestruct\_SIEII );

    VAR

      Buffer : Typestruct\_SIEII\_Buf ;

      I : Integer ;

    BEGIN

      FILEREAD(Fl, Buffer, Sizeof(Buffer));

      Buf^.Champ1:= Buffer.Champ1;

      Buf^.Champ2:= Buffer.Champ2;

    END;

  PROCEDURE Liredir\_SIEII ( VAR Fl: Thandle; VAR Buf : Typestruct\_SIEII;  N: INTEGER );

    VAR

      Buffer : Typestruct\_SIEII\_Buf ;

      I : Integer ;

    BEGIN

      FILESEEK(Fl, Sizeof(Buf\_caract\_SIEII) + (N-1)\*Sizeof(Buffer ), 0);

      FILEREAD(Fl, Buffer, Sizeof(Buffer));

      Buf^.Champ1:= Buffer.Champ1;

      Buf^.Champ2:= Buffer.Champ2;

    END;

  PROCEDURE Rajouter\_SIEII ( VAR Fl: Thandle;  Buf : Typestruct\_SIEII);

    VAR

      Buffer : Typestruct\_SIEII\_Buf ;

      I : Integer;

    BEGIN

      Buffer.Champ1:= Buf^.Champ1;

      Buffer.Champ2:= Buf^.Champ2;

      FILESEEK(Fl, 0, 2);

      FILEWRITE(Fl, Buffer, Sizeof(Buffer))

    END;

  FUNCTION Finfich\_SIEII ( VAR Fl : Thandle): BOOLEAN;

    VAR

      K, K2 : Longint;

    BEGIN

      K := FILESEEK(Fl, 0, 1);  { Position courante }

      K2 :=FILESEEK(Fl, 0, 2);  { Dernière position }

      IF K = K2

      THEN

        Finfich\_SIEII := true

      ELSE

        BEGIN

          FILESEEK(Fl, K, 0);

          Finfich\_SIEII := False

        END;

    END;

  FUNCTION Alloc\_bloc\_SIEII ( VAR Fl : Thandle) : INTEGER;

    VAR

      K : Longint;

    BEGIN

      K := FILESEEK(Fl, 0, 2); { Fin du fichier }

      K := K - Sizeof( Typestruct\_SIEII\_entete); { Ignorer l'en\_tête }

      K := K DIV Sizeof (Typestruct\_SIEII\_Buf);

      K := K + 1;

      Alloc\_bloc\_SIEII := K;

    END;

  {Partie déclaration de variables }

  VAR

    F :  Thandle;

    B1 : Typestruct\_SIEII ;

   {Corps du programme principal }

   BEGIN

     NEW(B1);

     ;READLN;

   END.

1. **Passage de Z vers C**

**Déclaration des variables**IDH\_cvoir1

Une déclaration de variables en C se fait par

Type <Li>;

où <Li> désigne une liste d'identificateurs.

Les objets simples

Equivalents des objets Z --> C

**ENTIER** se traduit par INT.

**CAR** se traduit par CHAR.

Au type "**CHAINE**" on associe le type construit Chaine que l'on définit par

typedef char Chaine[256]

En C le type Booléen n'existe pas.

Pour continuer à travailler toujours avec ce type, il suffit de rajouter au début du programme

typdef int Booleen

et de définir les valeurs True et False comme suit :

#define true 1

#define false 0

Les objet "structures"

Pour définir une structure S en C il faut choisir une implémentation.

Implémenter, c'est choisir une représentation mémoire ( généralement statique ou dynamique ) et traduire les opérations de la machine abstraite dans cette représentation.

Il suffit de remplacer la structure S par Pointeur et rajouter au niveau de l'en-tête du programme C l'implémentation désirée où l'on définira le type Pointeur.

**Boucle "TANTQUE"**

-----------Z----------

**TANTQUE** Cond :

Instructions

**FINTANTQUE**

se traduit par :

-------------C-----------

WHILE ( Cond )

{

Instructions

}

**Boucle "POUR"**

--------------Z-----------------

**POUR** V:= Exp1, Exp2 [, Exp3] [:]

Instructions

**FINPOUR**

se traduit par ║

---------------C---------------

for (V=Exp1; V<=Exp2; V=V+Exp3)

{

Instructions

}

**L'alternative "SI"**

-----------Z------------

**SI** Cond :

Instructions

[**SINON**

Instruction ]

**FSI**

se traduit par :

----------------C--------------

IF ( Cond )

{

Instructions

}

[ELSE

{Instructions } ]

**Z-expressions**

La grammaire des Z-expressions est incluse dans la grammaire C.

**Lecture**

--------Z--------

**LIRE**(V1, V2, ...)

se traduit par :

-----------C---------------

scanf("...", &V1, &V2, ...)

**Ecriture**

--------Z----------

**ECRIRE**(E1, E2, ...)

se traduit par :

--------C---------

printf(E1, E2, ...)

**Affectation**

Même syntaxe avec le '=' à la place de ':='.

**Action composée**

------------Z-------------

**ACTION** Nom ( P1, P2, ...);

**SOIENT**

Définition des objets locaux

et des paramètres

**DEBUT**

Instructions

**FIN**

se traduit par :

-----------------C----------------

void Nom ( typ1 P1 , typ2 P2, ...)

{

Définition des objets locaux

Instructions

}

**Fonctions**

------------------Z------------------

**FONCTION** Nom ( P1, P2, ...) : type;

**SOIENT**

Définition des objets locaux

et des paramètres

**DEBUT**

Instructions

**FIN**

se traduit par

------------------C------------------

type Nom ( typ1 P1, typ2 P2, ...)

{

Définition des objets locaux

Instructions

}

**Algorithme**

--------------------Z--------------------

**SOIENT**

Objets locaux et globaux

Annonce des modules

**DEBUT**

Instructions

**FIN**

Module 1

Module 2

...

Modules n

se traduit par :

-------------------C----------------------

Objets locaux et globaux

Définition des modules

Module 1

Module 2

...

Module n

main()

{

Instructions

}

1. **Implémentation des machines Z en C**

**Implémentation des vecteurs en C**

**SOIT TAB UN TABLEAU (5, 10) ;**

/\*\* Implémentation \*\*\: TABLEAU DE ENTIERS\*\*/

  /\*\* Tableaux \*\*/

  typedef int Typeelem\_V5\_10i ;

  typedef Typeelem\_V5\_10i \* Typevect\_V5\_10i ;

  Typeelem\_V5\_10i Element\_V5\_10i ( Typevect\_V5\_10i V , int I1 , int I2 )

    {

      return  \*(V + I2-1 + (I1-1) \*5 ) ;

    }

  void Aff\_element\_V5\_10i ( Typevect\_V5\_10i V  , int I1 , int I2,  Typeelem\_V5\_10i Val )

    {

      \*(V + I2-1 + (I1-1) \*5 ) = Val ;

    }

  /\*\* Partie déclaration de variables \*\*/

  Typevect\_V5\_10i Tab;

  int main(int argc, char \*argv[])

   {

     Tab = malloc(5\*10 \* sizeof(int));

   }

**Implémentation des structures en C**

**soit S une structure (Chaine, entier);**

typedef char \* string256 ;

  /\*\* Structures statiques \*\*/

  typedef struct Tsi Type\_Tsi  ;

  typedef Type\_Tsi \* Typestr\_Tsi ;

  typedef string256 Type1\_Tsi  ;

  typedef int Type2\_Tsi  ;

  struct Tsi

     {

      Type1\_Tsi Champ1 ;

      Type2\_Tsi Champ2 ;

    };

  Type1\_Tsi Struct1\_Tsi ( Typestr\_Tsi S)

    {

      return  S->Champ1 ;

    }

  Type2\_Tsi Struct2\_Tsi ( Typestr\_Tsi S)

    {

      return  S->Champ2 ;

    }

void Aff\_struct1\_Tsi ( Typestr\_Tsi S, Type1\_Tsi Val )

    {

      strcpy( S->Champ1 , Val );

    }

  void Aff\_struct2\_Tsi ( Typestr\_Tsi S, Type2\_Tsi Val )

    {

       S->Champ2 = Val ;

    }

  /\*\* Partie déclaration de variables \*\*/

  Typestr\_Tsi S;

  int main(int argc, char \*argv[])

    {

     S = malloc(sizeof(Typestr\_Tsi));

     S->Champ1 = malloc(sizeof(string256));

    }

**Implémentation des listes linéaires chaînées en C**

**SOIT L UNE LISTE ;**

     /\*\* Implémentation \*\*\: LISTE DE ENTIERS\*\*/

  /\*\* Listes linéaires chaînées \*\*/

  typedef int Typeelem\_Li   ;

  typedef struct Maillon\_Li \* Pointeur\_Li ;

  struct Maillon\_Li

    {

      Typeelem\_Li  Val ;

      Pointeur\_Li Suiv ;

    } ;

  Pointeur\_Li Allouer\_Li (Pointeur\_Li \*P)

    {

      \*P = (struct Maillon\_Li \*) malloc( sizeof( struct Maillon\_Li)) ;

    }

  void Aff\_val\_Li(Pointeur\_Li P, Typeelem\_Li V)

    {

      P->Val = V ;

    }

  void Aff\_adr\_Li( Pointeur\_Li P,  Pointeur\_Li Q)

    {

      P->Suiv = Q ;

**}**

Pointeur\_Li Suivant\_Li(  Pointeur\_Li P)

    { return( P->Suiv ) ;  }

  Typeelem\_Li Valeur\_Li( Pointeur\_Li P)

    { return( P->Val) ; }

  void Liberer\_Li ( Pointeur\_Li P)

    { free (P);}

  /\*\* Partie déclaration de variables \*\*/

  Pointeur\_Li L;

**Implémentation des listes bidirectionnelles en C**

**SOIT L UNE LISTEBI ;**

/\*\* Implémentation \*\*\: LISTE BIDIRECTIONNELLE DE ENTIERS\*\*/

  /\*\* Listes bidirectionnelles \*\*/

  typedef int Typeelem\_Ri   ;

  typedef struct Maillon\_Ri \* Pointeur\_Ri ;

  struct Maillon\_Ri

    {

      Typeelem\_Ri  Val ;

      Pointeur\_Ri Suiv ;

      Pointeur\_Ri Prec ;

    } ;

  Pointeur\_Ri Allouer\_Ri (Pointeur\_Ri \*P)

    { \*P = (struct Maillon\_Ri \*) malloc( sizeof( struct Maillon\_Ri))   ; }

  void Aff\_val\_Ri(Pointeur\_Ri P, Typeelem\_Ri V)

    { P->Val = V ; }

  void Aff\_adrd\_Ri( Pointeur\_Ri P,  Pointeur\_Ri Q)

    { P->Suiv = Q; }

  void Aff\_adrg\_Ri( Pointeur\_Ri P,  Pointeur\_Ri Q)

    { P->Prec = Q; }

  Pointeur\_Ri Suivant\_Ri(  Pointeur\_Ri P)

    { return( P->Suiv );  }

Pointeur\_Ri Precedent\_Ri(  Pointeur\_Ri P)

    { return( P->Prec );  }

  Typeelem\_Ri Valeur\_Ri( Pointeur\_Ri P)

    { return( P->Val) ; }

  void Liberer\_Ri ( Pointeur\_Ri P)

   { free (P) ; }

  /\*\* Partie déclaration de variables \*\*/

  Pointeur\_Ri Lb;

**Implémentation des arbres de recherche binaire en C**

**Soit A un arb ;**

      /\*\* Implémentation \*\*\: ARBRE BINAIRE DE ENTIERS\*\*/

  /\*\* Arbres de recherche binaire \*\*/

  typedef int Typeelem\_Ai   ;

  typedef struct Noeud\_Ai \* Pointeur\_Ai ;

  struct Noeud\_Ai

    {

      Typeelem\_Ai  Val ;

      Pointeur\_Ai Fg ;

      Pointeur\_Ai Fd ;

      Pointeur\_Ai Pere ;

     } ;

  Typeelem\_Ai Info\_Ai( Pointeur\_Ai P )

    { return P->Val;   }

  Pointeur\_Ai Fg\_Ai( Pointeur\_Ai P)

    { return P->Fg ; }

  Pointeur\_Ai Fd\_Ai( Pointeur\_Ai P)

    { return P->Fd ; }

Pointeur\_Ai Pere\_Ai( Pointeur\_Ai P)

    { return P->Pere ; }

  void Aff\_info\_Ai ( Pointeur\_Ai P, Typeelem\_Ai Val)

    { P->Val = Val ; }

  void Aff\_fg\_Ai( Pointeur\_Ai P, Pointeur\_Ai Q)

    { P->Fg =  Q;  }

  void Aff\_fd\_Ai( Pointeur\_Ai P, Pointeur\_Ai Q)

    { P->Fd =  Q ; }

  void Aff\_pere\_Ai( Pointeur\_Ai P, Pointeur\_Ai Q)

    { P->Pere =  Q ; }

  void Creernoeud\_Ai( Pointeur\_Ai \*P)

    {

      \*P = (struct Noeud\_Ai \*) malloc( sizeof( struct Noeud\_Ai))   ;

      (\*P)->Fg = NULL;

      (\*P)->Fd = NULL;

      (\*P)->Pere = NULL;

    }

  void Liberernoeud\_Ai( Pointeur\_Ai P)

    { free( P ) ; }

 /\*\* Partie déclaration de variables \*\*/

 Pointeur\_Ai A;

**Implémentation des arbres de recherche m-aire en C**

**Soit M un arM(4) ;**

     /\*\* Implémentation \*\*\: ARBRE M-AIRE DE ENTIERS\*\*/

  /\*\* Arbres de recherche m-aire \*\*/

  typedef int Typeelem\_M4i   ;

  typedef struct Noeud\_M4i \* Pointeur\_M4i ;

  struct Noeud\_M4i

    {

      int Degre ;

      Pointeur\_M4i Fils[4] ;

      Typeelem\_M4i Infor[4] ;

      Pointeur\_M4i Parent ;

     } ;

  Typeelem\_M4i Infor\_M4i(Pointeur\_M4i P, int I)

    { return P->Infor[I-1] ;  }

  Pointeur\_M4i Fils\_M4i( Pointeur\_M4i P, int I)

    { return P->Fils[I-1] ; }

  Pointeur\_M4i Parent\_M4i( Pointeur\_M4i P)

    { return P->Parent ; }

  void Aff\_infor\_M4i ( Pointeur\_M4i P, int I, Typeelem\_M4i Val)

    { P->Infor[I-1] = Val ; }

  void Aff\_fils\_M4i( Pointeur\_M4i P, int I, Pointeur\_M4i Q)

    { P->Fils[I-1] =  Q ; }

void Aff\_parent\_M4i( Pointeur\_M4i P, Pointeur\_M4i Q)

    { P->Parent =  Q ; }

  void Creernoeud\_M4i(  Pointeur\_M4i \*P )

    {

      int I ;

      \*P = (struct Noeud\_M4i \*) malloc( sizeof( struct Noeud\_M4i))   ;

      for (I=0; I< 4; ++I) (\*P)->Fils[I] = NULL;

      (\*P)->Degre = 0 ;

    }

  int Degre\_M4i ( Pointeur\_M4i P )

    { return P->Degre ; }

  void Aff\_degre\_M4i ( Pointeur\_M4i P, int N)

    { P->Degre = N ; }

  void Liberernoeud\_M4i(Pointeur\_M4i P)

    { free  ( P );}

   /\*\* Partie déclaration de variables \*\*/

  Pointeur\_M4i M;

**Implémentation des piles en C**

**Soit P une pile ;**

typedef int bool ;

  #define True 1

  #define False 0

  /\*\* Implémentation \*\*\: PILE DE ENTIERS\*\*/

  /\*\* Piles \*\*/

  typedef int Typeelem\_Pi ;

  typedef struct Maillon\_Pi \* Pointeur\_Pi ;

  typedef   Pointeur\_Pi  Typepile\_Pi  ;

  struct Maillon\_Pi

    {

      Typeelem\_Pi  Val ;

      Pointeur\_Pi Suiv ;

    } ;

  void Creerpile\_Pi( Pointeur\_Pi \*P )

    { \*P = NULL ; }

  bool Pilevide\_Pi ( Pointeur\_Pi P )

    { return  (P == NULL ); }

  void Empiler\_Pi ( Pointeur\_Pi \*P,  Typeelem\_Pi Val )

    {

      Pointeur\_Pi Q;

      Q = (struct Maillon\_Pi \*) malloc( sizeof( struct Maillon\_Pi))   ;

      Q->Val = Val ;

      Q->Suiv = \*P;

      \*P = Q;

    }

void Depiler\_Pi ( Pointeur\_Pi \*P,  Typeelem\_Pi \*Val )

    {

      Pointeur\_Pi Sauv;

      if (! Pilevide\_Pi (\*P) )

        {

          \*Val = (\*P)->Val ;

          Sauv = \*P;

          \*P = (\*P)->Suiv;

          free(Sauv);

        }

      else printf ("%s \n", "Pile vide");

    }

  /\*\* Partie déclaration de variables \*\*/

  Pointeur\_Pi P;

**Implémentation des files d'attente en C**

**Soit F une file ;**

typedef int bool ;

  #define True 1

  #define False 0

  /\*\* Implémentation \*\*\: FILE DE ENTIERS\*\*/

  /\*\* Files d'attente \*\*/

  typedef int Typeelem\_Fi ;

  typedef  struct Filedattente\_Fi \* Pointeur\_Fi ;

  typedef  struct Maillon\_Fi \* Ptliste\_Fi ;

  struct Maillon\_Fi

    {

      Typeelem\_Fi Val ;

      Ptliste\_Fi Suiv  ;

    };

  struct Filedattente\_Fi

    {

      Ptliste\_Fi Tete, Queue ;

    };

  void Creerfile\_Fi ( Pointeur\_Fi \*Fil   )

    {

      \*Fil = (struct Filedattente\_Fi \*) malloc( sizeof( struct Filedattente\_Fi)) ;

      (\*Fil)->Tete = NULL ;

      (\*Fil)->Queue = NULL ;

    }

bool Filevide\_Fi (Pointeur\_Fi Fil  )

    { return  Fil->Tete == NULL ;}

  void Enfiler\_Fi ( Pointeur\_Fi Fil , Typeelem\_Fi Val   )

    {

      Ptliste\_Fi P;

      P = (struct Maillon\_Fi \*) malloc( sizeof( struct Maillon\_Fi))   ;

      P->Val = Val ;

      P->Suiv = NULL;

      if ( ! Filevide\_Fi(Fil) )

        Fil->Queue->Suiv = P ;

      else Fil->Tete = P;

        Fil->Queue = P;

    }

  void Defiler\_Fi (Pointeur\_Fi Fil, Typeelem\_Fi \*Val )

    {

      if (! Filevide\_Fi(Fil) )

        {

          \*Val = Fil->Tete->Val ;

          Fil->Tete =  Fil->Tete->Suiv;

        }

      else printf ("%s \n", "File d'attente vide");

    }

 /\*\* Partie déclaration de variables \*\*/

 Pointeur\_Fi F;

**Implémentation des fichiers en C**

**soit F un fichier de (chaines, entier) entete (ENTIER,   entier)  buffer B1;**

  #include <stdio.h>

  #include <stdlib.h>

  typedef int bool ;

  typedef char \* string255 ;

  #define True 1

  #define False 0

  /\*\* Implémentation \*\*\: FICHIER

  /\* Traitement des fichiers ouverts \*/

  /\* Traitement des fichiers ouverts \*/

  struct \_Noeud

    {

      FILE \* Var\_fich ;

      char \* Nom\_fich ;

      int Sauv\_pos;

      struct \_Noeud \*Suiv ;

    } ;

  typedef struct \_Noeud \* \_Ptr\_Noeud;

  \_Ptr\_Noeud  \_Pile\_ouverts  = NULL;

  /\* Teste si un fichier est ouvert \*/

  \_Ptr\_Noeud \_Ouvert ( char \* Fp)

    {

      \_Ptr\_Noeud P;

      bool Trouv ;

      P = \_Pile\_ouverts; Trouv = False ;

      while ((P != NULL) && ! Trouv )

        if ( strcmp(P->Nom\_fich, Fp) == 0)

        Trouv = True;

        else P = P->Suiv;

      return P;

    }

  /\* Ajouter un fichier ouvert \*/

  void \_Empiler\_ouvert ( char \*Fp, FILE \*Fl)

    {

      \_Ptr\_Noeud  P ;

      P = (\_Ptr\_Noeud) malloc( sizeof( struct \_Noeud)) ;

      P->Nom\_fich = Fp;

      P->Var\_fich = Fl;

      P->Suiv = \_Pile\_ouverts;

      \_Pile\_ouverts = P;

    }

  /\* Supprimer un fichier ouvert et rendre son nom\*/

  char \* \_Depiler\_ouvert ( FILE \*Fl)

    {

      char \* Fp = malloc (100);

      \_Ptr\_Noeud P,  Prec  ;

      P= \_Pile\_ouverts;

      Prec = NULL;

      while (P->Var\_fich != Fl )

        { Prec = P ; P = P->Suiv ;}

      strcpy(Fp, P->Nom\_fich);

      if (Prec != NULL)

        Prec->Suiv = P->Suiv;

      else \_Pile\_ouverts = P->Suiv;

      free (P);

      return Fp ;

    }

  /\*\* Fichiers \*\*/

  typedef char \_Tx[255];

  /\*\* Types des champs du bloc \*\*/

  typedef string255 Typechamp1\_siEii;

  typedef \_Tx Typechamp1\_siEii\_Buf ;

  typedef int Typechamp2\_siEii;

  /\*\* Types des champs de l'en-tête \*/

  typedef int Typeentete1\_siEii ;

  typedef int Typeentete2\_siEii ;

  /\*\* Type du bloc de données du fichier \*\*/

  typedef struct

    {

      Typechamp1\_siEii\_Buf Champ1 ;

      Typechamp2\_siEii Champ2 ;

    }  Typestruct1\_siEii\_Buf;

  /\*\* Type de la structure du bloc de données du fichier \*\*/

  typedef struct

    {

      Typechamp1\_siEii Champ1 ;

      Typechamp2\_siEii Champ2 ;

    }  Typestruct1\_siEii\_;

  typedef Typestruct1\_siEii\_  \* Typestruct1\_siEii ;

  typedef Typestruct1\_siEii Typestr\_Tsi;

  typedef Typestruct1\_siEii\_ Type\_Tsi;

  /\*\* Type du bloc des caractéristiques du fichier \*\*/

  typedef struct

    {

      Typeentete1\_siEii Entete1   ;

      Typeentete2\_siEii Entete2   ;

    } Typestruct2\_siEii ;

 /\*\* Manipulation de la structure \*\*/

 Typechamp1\_siEii Struct1\_Tsi ( Typestruct1\_siEii Buf )

    {

       return  Buf->Champ1;

    }

 Typechamp2\_siEii Struct2\_Tsi ( Typestruct1\_siEii Buf )

    {

       return  Buf->Champ2;

    }

 void Aff\_struct1\_Tsi ( Typestruct1\_siEii Buf, Typechamp1\_siEii Val )

    {

      strcpy( Buf->Champ1 , Val );

    }

 void Aff\_struct2\_Tsi ( Typestruct1\_siEii Buf, Typechamp2\_siEii Val )

    {

       Buf->Champ2 = Val ;

    }

  /\*\* Déclaration du buffer de l'en-tête \*\*/

  Typestruct2\_siEii Bloc\_caract\_siEii;

  /\*\* Opérations sur les fichiers \*\*/

  void Ouvrir\_siEii ( FILE \*\*siEii , char \*Fp , char \* Mode )

    {

       \_Ptr\_Noeud P = \_Ouvert(Fp);

       if ( P != NULL )

       /\* Le fichier est déjà ouvert \*/

         {

          P->Sauv\_pos = ftell (P->Var\_fich);

          fseek( P->Var\_fich, 0, 0);

          fwrite(&Bloc\_caract\_siEii, sizeof(Typestruct2\_siEii), 1, P->Var\_fich);

          fclose(P->Var\_fich);

         }

       /\* Le fichier est non ouvert \*/

       if ( strcmp(Mode,"A") == 0)

         {

           \*siEii = fopen(Fp, "r+b");

           fread(&Bloc\_caract\_siEii, sizeof(Typestruct2\_siEii), 1, \*siEii);

         }

         else

         {

           \*siEii = fopen(Fp, "w+b");

           fwrite(&Bloc\_caract\_siEii, sizeof(Typestruct2\_siEii), 1, \*siEii)  ;

         }

         \_Empiler\_ouvert( Fp, \*siEii);

    }

  void Fermer\_siEii ( FILE \* siEii )

    {

      char \* Fp = malloc(100);

      \_Ptr\_Noeud P  ;

      strcpy(Fp, \_Depiler\_ouvert(siEii));

      fseek( siEii, 0, 0);

      fwrite(&Bloc\_caract\_siEii, sizeof(Typestruct2\_siEii), 1, siEii);

      fclose(siEii) ;

      /\* Ya-til un fichier ouvert avec le même nom ?  \*/

      /\* Si Oui, le Réouvrir à la position sauvegardée \*/

      P =  \_Ouvert (Fp);

      if ( P != NULL)

      {

        siEii = fopen(P->Nom\_fich, "r+b");

        fread(&Bloc\_caract\_siEii, sizeof(Typestruct2\_siEii), 1, siEii);

        fseek(siEii, P->Sauv\_pos, 0);

      }

    }

  Typeentete1\_siEii Entete1\_siEii(  FILE \* siEii)

    {

      return  Bloc\_caract\_siEii.Entete1;

    }

  Typeentete2\_siEii Entete2\_siEii(  FILE \* siEii)

    {

      return  Bloc\_caract\_siEii.Entete2;

    }

  void Aff\_entete1\_siEii ( FILE \* siEii, Typeentete1\_siEii VAL)

    {

       Bloc\_caract\_siEii.Entete1 = VAL ;

    }

  void Aff\_entete2\_siEii ( FILE \* siEii, Typeentete2\_siEii VAL)

    {

       Bloc\_caract\_siEii.Entete2 = VAL ;

    }

  void Ecrireseq\_siEii ( FILE \* siEii, Typestruct1\_siEii Buf  )

    {

      Typestruct1\_siEii\_Buf Buffer ;

      int I, J;

      for(J=0; J<= strlen(Buf->Champ1); ++J)

        Buffer.Champ1[J] = Buf->Champ1[J];

      Buffer.Champ2 = Buf->Champ2;

      fwrite(&Buffer, sizeof( Typestruct1\_siEii\_Buf), 1, siEii)  ;

    }

  void Ecriredir\_siEii ( FILE \* siEii, Typestruct1\_siEii Buf, int N )

    {

      Typestruct1\_siEii\_Buf Buffer ;

      int I, J;

      for(J=0; J<= strlen(Buf->Champ1); ++J)

        Buffer.Champ1[J] = Buf->Champ1[J];

      Buffer.Champ2 = Buf->Champ2;

      fseek(siEii, (long) ((N-1)\* sizeof( Typestruct1\_siEii\_Buf) +

             sizeof( Typestruct2\_siEii)), 0 );

      fwrite(&Buffer, sizeof( Typestruct1\_siEii\_Buf), 1, siEii)  ;

    }

  void Lireseq\_siEii ( FILE \* siEii, Typestruct1\_siEii Buf )

    {

      Typestruct1\_siEii\_Buf Buffer ;

      int I, J;

      if (fread(&Buffer, sizeof( Typestruct1\_siEii\_Buf), 1, siEii)!=0){

      for(J=0; J<= strlen(Buffer.Champ1); ++J)

        Buf->Champ1[J] = Buffer.Champ1[J];

      Buf->Champ2= Buffer.Champ2;

      }

    }

  void Liredir\_siEii ( FILE \* siEii, Typestruct1\_siEii Buf, int N)

    {

      Typestruct1\_siEii\_Buf Buffer ;

      int I, J;

      fseek(siEii, (long) ((N-1)\* sizeof( Typestruct1\_siEii\_Buf) +

             sizeof( Typestruct2\_siEii)), 0 );

      fread(&Buffer, sizeof( Typestruct1\_siEii\_Buf), 1, siEii);

      for(J=0; J<= strlen(Buffer.Champ1); ++J)

        Buf->Champ1[J] = Buffer.Champ1[J];

      Buf->Champ2= Buffer.Champ2;

    }

  void Rajouter\_siEii ( FILE \* siEii, Typestruct1\_siEii Buf )

    {

      Typestruct1\_siEii\_Buf Buffer ;

      int I, J;

      for(J=0; J<= strlen(Buf->Champ1); ++J)

        Buffer.Champ1[J] = Buf->Champ1[J];

      Buffer.Champ2 = Buf->Champ2;

      fseek(siEii, 0, 2); /\* Fin du fichier \*/

      fwrite(&Buffer, sizeof( Typestruct1\_siEii\_Buf), 1, siEii)  ;

    }

  bool Finfich\_siEii (FILE \* siEii)

    {

      long K = ftell(siEii);

      fseek(siEii, 0, 2); /\* Fin du fichier \*/

      long K2 = ftell(siEii);   /\* position à partir du debut \*/

      if  (K==K2)

        { fseek(siEii, K, 0); return 1;}

      else

        { fseek(siEii, K, 0); return 0;}

    }

  int Alloc\_bloc\_siEii (FILE \* siEii)

    {

      long K;

      fseek(siEii, 0, 2); /\* Fin du fichier \*/

      K = ftell(siEii);   /\* position à partir du debut \*/

      K = K - sizeof( Typestruct2\_siEii); /\* Ignorer l'en\_tête \*/

      K = K / sizeof (Typestruct1\_siEii\_Buf);

      K ++;

      return(K);

    }

  /\*\* Variables du programme principal \*\*/

  FILE \*F;

  Typestruct1\_siEii B1 ;

  int main(int argc, char \*argv[])

    {

      B1 = malloc(sizeof(Typestruct1\_siEii));

      B1->Champ1 = malloc(255 \* sizeof(string255));

      system("PAUSE");

      return 0;

    }

1. **Index des mots-clés Z**

**A**

ACTIONIDH\_act

ACTIONSIDH\_act

AFF\_ADRIDH\_mlistes

AFF\_ADRDIDH\_mlistebi

AFF\_ADRGIDH\_mlistebi

AFF\_DEGREIDH\_marm

AFF\_ELEMENTIDH\_mvect

AFF\_ENTETEIDH\_mfich

AFF\_FDIDH\_marb

AFF\_FGIDH\_marb

AFF\_FILSIDH\_marm

AFF\_INFOIDH\_marb

AFF\_INFORIDH\_marm

AFF\_PEREIDH\_marb

AFF\_STRUCTIDH\_mstruct

AFF\_VALIDH\_mlistes

ALEACHAINEIDH\_foncts2

ALEANOMBREIDH\_foncts2

ALLOC\_BLOCIDH\_mfich

ALLOC\_STRUCTIDH\_mstruct

ALLOC\_TABIDH\_mvect

ALLOUERIDH\_mlistes

APPELIDH\_act

ARBIDH\_arbreb

ARMIDH\_arbrem

**B**

BOOLEENIDH\_obj

BOOLEENSIDH\_obj

BUFFERIDH\_fichier

**C**

CARIDH\_obj

CARACTIDH\_foncts3

CARACTEREIDH\_obj

CARACTERESIDH\_obj

CHAINEIDH\_obj

CHAINESIDH\_obj

CREERFILEIDH\_mfile

CREERNOEUDIDH\_marb

CREERPILEIDH\_mpile

CREER\_ARBIDH\_acth

CREER\_ARMIDH\_acth

CREER\_FILEIDH\_acth

CREER\_LISTEIDH\_acth

CREER\_LISTEBIIDH\_acth

CREER\_PILEIDH\_acth

**D**

DEIDH\_mach

DEBUTIDH\_struct

DEFILERIDH\_mfile

DEGREIDH\_marm

DEPILERIDH\_mpile

DESIDH\_obj

DYNAMIQUEIDH\_tableau

DYNAMIQUESIDH\_tableau

**E**

ECRIREIDH\_acte

ECRIREDIRIDH\_mfich

ECRIRESEQIDH\_mfich

ELEMENTIDH\_mvect

EMPILERIDH\_mpile

ENFILERIDH\_mfile

ENTETEIDH\_mfich

ENTIERIDH\_obj

ENTIERSIDH\_obj

ETIDH\_exp

EXPIDH\_foncts1

**F**

FAUXIDH\_exp

FDIDH\_marb

FERMERIDH\_mfich

FGIDH\_marb

FICHIERIDH\_fichier

FILEIDH\_files

FILESIDH\_files

FILEVIDEIDH\_mfile

FILSIDH\_marm

FINIDH\_struct

FINFICHIDH\_mfich

FINPOURIDH\_str

FINSIIDH\_str

FINTANTQUEIDH\_str

FONCTIONIDH\_fonct

FONCTIONSIDH\_fonct

FPOURIDH\_str

FSIIDH\_str

FTQIDH\_str

**I**

INFOIDH\_marb

INFORIDH\_marm

INIT\_STRUCTIDH\_acth

INIT\_TABIDH\_acth

INIT\_VECTIDH\_acth

**L**

LIBERERIDH\_mlistes

LIBER\_STRUCTIDH\_mstruct

LIBER\_TABIDH\_mvect

LIBERERNOEUDIDH\_marb

LIREIDH\_acte

LIREDIRIDH\_mfich

LIRESEQIDH\_mfich

LISTEIDH\_liste

LISTEBIIDH\_listebi

LISTESIDH\_liste

LONGCHAINEIDH\_foncts3

**M**

MAXIDH\_foncts1

MINIDH\_foncts1

MODIDH\_foncts1

**N**

NILIDH\_exp

NONIDH\_exp

**O**

OUIDH\_exp

OUVRIRIDH\_mfich

**P**

PEREIDH\_marb

PILEIDH\_pile

PILESIDH\_pile

PILEVIDEIDH\_mpile

POINTEURIDH\_mach

POINTEURSIDH\_mach

POURIDH\_str

PRECEDENTIDH\_mlistebi

**R**

RAJOUTERIDH\_mfich

**S**

SIIDH\_str

SINONIDH\_str

SOIENTIDH\_struct

SOITIDH\_struct

STRUCTIDH\_mstruct

STRUCTUREIDH\_structure

STRUCTURESIDH\_structure

SUIVANTIDH\_mlistes

**T**

TABLEAUIDH\_tableau

TABLEAUXIDH\_tableau

TANTQUEIDH\_str

TQIDH\_str

**U**

UNIDH\_obj

UNEIDH\_obj

**V**

VALEURIDH\_mlistes

VECTEURIDH\_mach

VECTEURSIDH\_mach

VERSIDH\_mach

VRAIIDH\_exp