

Khawarizm

Environnement d'écriture d'algorithmes abstraits

et de passage vers les langages PASCAL et C

Pr D.E ZEGOUR

Ecole Supérieure d'Informatique

SOMMAIRE

1. Présentation générale (Page 5)

Présentation
Menu
Traitements
Langage Z
Documentation

2. 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

3. Langage Z (Base) (Page 15)

Généralités
Structure d'un Z-algorithme
Action composée
Fonction
Scalaires
Pointeurs
Expressions
Actions élémentaires
Structures de contrôle
Commentaires
Actions de haut niveau
Fonctions standards Mod, Min, Max, Exp
Fonctions de génération aléatoire Aleachaine, Aleanombre
Fonctions sur chaînes de caractères Caract, Longchaîne
Exemple d'un Z-algorithme

4. Langage Z (Structures de données) (Page 21)

Tableaux
Structures
Listes linéaires chaînées
Listes linéaires chaînées bilatérales
Files d'attente
Piles
Arbres de recherche binaire

Arbres de recherche m-aire
Fichiers

5. Machines abstraites (Page 26)

Vecteurs
Structures
Listes linéaires chaînées
Listes bidirectionnelles
Piles
Files d'attente
Arbres de recherche binaire
Arbres de recherche m-aire
Fichiers

6. Passage de Z vers PASCAL (Page 31)

Déclarations
Affectation
Expressions
La boucle Tantque
La boucle Pour
L'alternative Si
Lecture
Ecriture
Action composée
Fonction
Fonctions standards
Programme

7. Implémentation des machines Z en PASCAL (Page 36)

Vecteurs
Structures
Listes linéaires chaînées
Listes bidirectionnelles
Piles
Files d'attente
Arbres de recherche binaire
Arbres de recherche m-aire
Fichiers

8. Passage de Z vers C (Page 49)

Déclarations
Affectation
Expressions
La boucle Tantque
La boucle Pour
L'alternative Si
Lecture
Ecriture
Action composée

Fonction
Fonctions standards
Programme

9. Implémentation des machines Z en C (Page 52)

Vecteurs
Structures
Listes linéaires chaînées
Listes bidirectionnelles
Piles
Files d'attente
Arbres de recherche binaire
Arbres de recherche m-aire
Fichiers

10. Index des mots-clés (Page 66)

1. 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.

2. 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 }
FUNCTION 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

```
/** Conversion 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 Libérer_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

```

/** Conversion 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 Libérer_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;
}

```

3. Langage Z (Base)

Généralités sur le langage Z

> Un Z-algorithme est un ensemble de modules parallèles dont le premier est principal et les autres sont soit des actions composées soit des fonctions.

> 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 même 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**.

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

> On définit ainsi des machines abstraites sur

- les structures ou objets composés
- les vecteurs
- les listes mono directionnelles
- les listes bidirectionnelles
- les piles
- les files d'attente
- les arbres de recherche binaire
- les arbres de recherche m-aire
- les fichiers

> 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**) et des entiers (**ALEANOMBRE**).

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

> 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.

Scalars

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

Définition des scalaires

[**SOIT/SOIENT**] Sep Type

 : 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**] Sep
POINTEUR VERS [sep] [**DE** Typec **DE** Typec **DE**] [**DE** Types]

 : 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 où 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 */
FUNCTION 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

```

4. 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]

```

 : 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]

```

 : 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 ]

```

 : 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] Sep LISTEBI [DE Typec DE Typec DE][DE Types]

 : 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] Sep FILE [DE Typec DE Typec DE][DE Types]

 : 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] Sep PILE [DE Typec DE Typec DE][DE Types]

 : 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] Sep ARB [DE Typec DE Typec DE][DE Types]

 : 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] Sep **ARM** (degre) [**DE** Typec **DE** Typec **DE**] [**DE** Types]

 : 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] sep **FICHIER DE** type

BUFFER

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

 : 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)** ;

5. Machines abstraites

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

[SOIT/SOIENT] Sep <Machine abstraite>

 : 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.

EMPLER (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.

6. Passage de Z vers PASCAL

Déclaration des variables

Une déclaration de variables PASCAL se fait par
VAR : Type;
où 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;
  Chaîne : STRING;
BEGIN
  Chaîne := "";
  FOR K:=1 TO N DO
    CASE Random(2) OF
      0 : Chaîne := Chaîne + CHR(97+Random(26)) ;
      1 : Chaîne := Chaîne + CHR(65+Random(26)) ;
    END;
  Aleachaine := Chaîne;
END;
```

LONGCHAINE (C)

```
FUNCTION Longchaîne(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.
```

7. 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
    Typelem_LI = INTEGER;
    Pointeur_LI = ^Maillon_LI; { type du champ 'Adresse' }
    Maillon_LI = RECORD
        Val : Typelem_LI;
        Suiv : Pointeur_LI
    END;

PROCEDURE Allouer_LI ( VAR P : Pointeur_LI );
BEGIN NEW(P) END;

PROCEDURE Libérer_LI ( P : Pointeur_LI );

```

```

BEGIN DISPOSE(P) END;

PROCEDURE Aff_val_LI(P : Pointeur_LI; Val : Typelem_LI);
BEGIN P^.Val := Val END;

FUNCTION Valeur_LI (P : Pointeur_LI) : Typelem_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
    Typelem_RI = INTEGER;
    Pointeur_RI = ^Maillon_RI; { type du champ 'Adresse' }
    Maillon_RI = RECORD
        Val : Typelem_RI;
        Suiv : Pointeur_RI;
        Prec : Pointeur_RI
    END;

PROCEDURE Allouer_RI ( VAR P : Pointeur_RI );
BEGIN NEW(P) END;

PROCEDURE Libérer_RI ( P : Pointeur_RI );
BEGIN DISPOSE(P) END;

PROCEDURE Aff_val_RI (P : Pointeur_RI; Val : Typelem_RI);
BEGIN P^.Val := Val END;

FUNCTION Valeur_RI (P : Pointeur_RI) : Typelem_RI;
BEGIN Valeur_RI := P^.Val END;

FUNCTION Suivant_RI( P : Pointeur_RI) : Pointeur_RI;
BEGIN Suivant_RI := P^.Suiv END;

FUNCTION Precédent_RI( P : Pointeur_RI) : Pointeur_RI;
BEGIN Precédent_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 Liberernoed_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 Creerpil_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_Noed = ^_Noed;
    _Noed = RECORD
        Var_fich : Thandle;
        Nom_fich : string;
        Sauv_pos : Longint;
        Suiv : _Ptr_Noed
    END;

VAR
    _Pile_ouverts : _Ptr_Noed = NIL;

FUNCTION _Ouvert (Fp : String) : _Ptr_Noed;
VAR
    P : _Ptr_Noed;
    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_Noed ;
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_Noed ;
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-t-il 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 Ecrire_dir_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 Lire_dir_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;
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.

```


8. Passage de Z vers C

Déclaration des variables

Une déclaration de variables en C se fait par
Type ;
où 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 "**CHAÎNE**" on associe le type construit Chaîne que l'on définit par
typedef char Chaîne[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

```
typedef 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
}

9. 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 Libérer_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 Libernoeud_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 */
_PPtr_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-t-il 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 Ecrire_dir_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 Lire_dir_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;
}

```

10. Index des mots-clés Z

A

ACTION
ACTIONS
AFF_ADR
AFF_ADRD
AFF_ADRG
AFF_DEGRE
AFF_ELEMENT
AFF_ENTETE
AFF_FD
AFF_FG
AFF_FILS
AFF_INFO
AFF_INFOR
AFF_PERE
AFF_STRUCT
AFF_VAL
ALEACHAINE
ALEANOMBRE
ALLOC_BLOC
ALLOC_STRUCT
ALLOC_TAB
ALLOUER
APPEL
ARB
ARM

B

BOOLEEN
BOOLEENS
BUFFER

C

CAR
CARACT
CARACTERE
CARACTERES
CHAINE
CHAINES
CREERFILE
CREERNOEUD
CREERPILE
CREER_ARB
CREER_ARM
CREER_FILE
CREER_LISTE
CREER_LISTEBI
CREER_PILE

D

DE
DEBUT
DEFILER
DEGRE
DEPILER

DES
DYNAMIQUE
DYNAMIQUES

E

ECRIRE
ECRIREDIR
ECRIRESEQ
ELEMENT
EMPLER
ENFILER
ENTETE
ENTIER
ENTIERS
ET
EXP

F

FAUX
FD
FERMER
FG
FICHIER
FILE
FILES
FILEVIDE
FILS
FIN
FINFICH
FINPOUR
FINSI
FINTANTQUE
FONCTION
FONCTIONS
FPOUR
FSI
FTQ

I

INFO
INFOR
INIT_STRUCT
INIT_TAB
INIT_VECT

L

LIBERER
LIBER_STRUCT
LIBER_TAB
LIBERERNOEUD
LIRE
LIREDIR
LIRESEQ
LISTE
LISTEBI
LISTES
LONGCHAINE

M

MAX

MIN
MOD

N

NIL
NON
O
OU
OUVRIR

P

PERE
PILE
PILES
PILEVIDE
POINTEUR
POINTEURS
POUR
PRECEDENT

R

RAJOUTER

S

SI
SINON
SOIENT
SOIT
STRUCT STRUCTURE
STRUCTURES
SUIVANT

T

TABLEAU
TABLEAUX
TANTQUE
TQ

U

UN
UNE

V

VALEUR
VECTEUR
VECTEURS
VERS
VRAI