

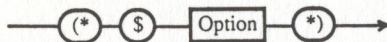
6.1 LES OPTIONS DE COMPILATION**6.1.1 DEFINITION**

Les options de compilation sont des indications placées dans le programme source et destinées à influencer le comportement du compilateur.

Par exemple, si nous demandons (*\$R-*), cela supprimera dans le code généré toutes les instructions de vérification d'intervalle.

6.1.2 SYNTAXE

Ces options sont tapées dans le source, dans un commentaire:



ATTENTION:

- il ne faut aucun espace entre le commentaire ouvrant (que vous utilisiez "(*" ou l'accolade ouvrante "(") et le \$
- la présence d'espaces entre la fin de l'option et la fin du commentaire peut modifier le comportement de l'option. Nous le préciserons lorsque cet espace est significatif.

Certaines options doivent aussi figurer à une place précise (par exemple, en début du source), alors que d'autres peuvent figurer n'importe où.

Si à un endroit vous souhaitez utiliser plusieurs options qui n'utilisent pas de chaîne de caractères, vous pouvez les regrouper, en les séparant par une virgule, sans ajouter aucun espace:

(*\$R+,I-*)

Vous pouvez aussi les laisser séparées:

(*\$R+ *)
(*\$R- *)

Si l'option comporte un nom de fichier source, il n'est pas nécessaire de taper le suffixe .PAS: le compilateur le rajoutera pour vous.

6.1.3 EFFET

Les options agissent sur le compilateur au moment où elles sont lues.

Leur effet est donc indépendant de la structure du programme. Par exemple:

```

1 PROGRAM Options;
2   VAR Nombre: INTEGER;
3       Fichier: FILE OF CHAR;
4
5   PROCEDURE Ouvre;
6   BEGIN
7     ASSIGN(Fichier, 'PROG.PAS');
8     RESET(Fichier);
9     (*$I-*)
10    WRITE(Fichier, Nombre);
11  END;
12
13 BEGIN
14   Ouvre;
15   (*$I+*)
16   CLOSE(Fichier);
17 END.
```

Fig. 6.1 Une option est valide de ligne à ligne

Aucune erreur d'entrée / sortie ne sera provoquée pour les instructions situées entre les lignes 9 à 15. Le RESET peut provoquer une erreur d'exécution, alors que WRITE n'en provoquera pas (bien que l'appel à Ouvre soit compris dans les lignes 8 à 15).

6.1.4 LA LISTE DES OPTIONS

Les options disponibles pour la version 3 sur PC sont les suivantes:

- B: met en oeuvre le tampon de saisie
- C: met en oeuvre CTRL C et CTRL S
- D: modifie l'utilisation de tampons d'entrée pour les périphériques
- F: modifie le nombre de fichiers qui peuvent être ouverts à un même instant
- G: modifie la taille des tampons d'entrée
- I: permet la gestion des erreurs d'entrée / sortie
- I nom: met en oeuvre les fichiers inclus
- K: permet la vérification de pile
- P: modifie la taille des tampons de sortie
- R: permet la vérification d'intervalle
- U: met en oeuvre la gestion de CTRL C
- V: permet la vérification de paramètres chaînes

Ces options peuvent être regroupées de la façon suivante:

- I nom: gestion des fichiers inclus
- R, I, K, V: gestion des erreurs
- B, C, D, U: tampon d'entrée et contrôle du programme
- G, P, F: paramétrage des fichiers

6.2 LES FICHIERS INCLUS: L'OPTION I

6.2.1 DEFINITION

L'option Include permet de chaîner plusieurs fichiers texte au moment de la compilation

Exemple:

```
(* $I PROG2.PAS *)
{ $I c:calculpoutrecoupe }
```

6.2.2 MOTIVATION

L'option est destinée à permettre le morcellement du source en plusieurs fichiers.

Ceci se produit:

- lorsque le texte de votre programme ne peut pas tenir en mémoire en utilisant l'éditeur (64K moins la taille des variables du compilateur (\$0802) moins la taille des messages d'erreur)
- lorsque vous utilisez un programme composé de plusieurs "modules fonctionnels".

Voici un programme monolithique;--

```
0 (* 001 optin1 *)
1 (* 20 aou 87 *)
2
3 PROGRAM monolithe;
4
5 PROCEDURE deux;
6 BEGIN
7 END; (* deux *)
8
9 PROCEDURE trois;
10 BEGIN
11 END; (* trois *)
12
```

```
13 PROCEDURE quatre;
14 BEGIN
15 END; (* quatre *)
16
17 BEGIN (* principal *)
18 END.
```

Fig. 6.2 Le programme monolithe.

et voici sa version en utilisant un fichier inclus:

```
0 (* 001 optin2 *)
1 (* 20 aou 87 *)
2
3 PROGRAM modulaire;
4
5 PROCEDURE deux;
6 BEGIN
7 END; (* deux *)
8
9 (* $i partie1 *)
10
11 BEGIN (* principal *)
12 END.
```

Fig. 6.3 Le programme modulaire.

et le fichier inclus:

```
0 (* 001 partie1 *)
1 (* 20 aou 87 *)
2
3 PROCEDURE trois;
4 BEGIN
5 END; (* trois *)
6
7 PROCEDURE quatre;
8 BEGIN
9 END; (* quatre *)
```

Fig. 6.4 Le fichier inclus

6.2.3 FONCTIONNEMENT

Lorsque le compilateur rencontre l'option Include:

- il cherche ce fichier sur le disque, et provoque une erreur s'il ne le trouve pas
- il termine de compiler la ligne contenant l'option jusqu'au RETURN (eh oui !)
- les symboles suivants sont ceux du fichier inclus
- lorsque la compilation de ce fichier est terminée, les symboles sont à nouveau lus depuis le tampon en mémoire qui contient le fichier principal

Pour simplifier la gestion du fichier inclus, le compilateur termine d'analyser la ligne comportant l'option d'inclusion avant de compiler le texte de ce fichier. Si nous avons:

```
(* $i inclus.pas *) TYPE string80 = STRING[80];
```

alors le type *string80* pourra être utilisé dans *inclus.pas*. En général il vaut donc mieux placer les options d'inclusion sur une ligne ne comportant que cette option.

Le compilateur utilise donc:

- un tampon de 64K pour le fichier principal (Work)
- un tampon de 128 octets pour lire le texte d'un fichier inclus situé sur disque

Le menu principal permet de désigner deux fichiers:

- Work qui est le fichier utilisé pour éditer
- Main qui est le fichier que le compilateur compilera en premier

Et:

- si vous ne travaillez que sur un seul fichier, il n'est pas nécessaire de préciser le nom de Main. Le compilateur utilisera Work pour la compilation.
- si vous travaillez (en le modifiant par l'éditeur) sur un des fichiers inclus, ce fichier devra être appelé par l'option Work. Pour que le compilateur commence la compilation par le fichier principal, il faudra en préciser le nom par l'option Main.

L'utilisation de deux noms évite simplement lors de multiples modifications:

- de sauvegarder le fichier inclus
- de charger par Work le fichier principal avant la compilation
- de rappeler dans Work le fichier inclus pour d'autres modifications

Il suffit le matin d'indiquer le nom du fichier inclus dans Work et du fichier principal dans Main. TURBO se charge du reste (sauvegardes, chargements...)

Un seul petit défaut: si vous compilez d'abord en mémoire, puis souhaitez compiler en mode .COM, ne sauvez pas le fichier inclus avant la compilation disque (TURBO recommence alors toute la compilation au lieu de recopier le code. En effet le fichier inclus peut être plus gros que le fichier principal, et le chargement dans le tampon mémoire peut avoir écrasé le code). Cette sauvegarde est d'ailleurs inutile ici (si la compilation a réussi, elle a déjà été effectuée).

6.2.4 UTILISATION

Notez que:

- le programme peut être découpé n'importe comment en fichiers inclus. Il n'est pas nécessaire de respecter les limites de la syntaxe (procédures, déclarations / instructions...)

- un découpage fonctionnel est cependant recommandé. Il vaut mieux regrouper dans un même fichier inclus des procédures ayant une même fonction. Par exemple les procédures de gestion de fenêtre, de séquentiel indexé, d'émission série, d'affichage graphique, de calcul matriciel...

Ce découpage améliore la lisibilité et facilite la construction de bibliothèques fonctionnelles.

La possibilité de placer des définitions (LABEL, CONST, TYPE, VAR) entre les procédures (et non pas seulement en tête du programme ou juste après la déclaration de la procédure) permet de regrouper les définitions et déclarations des variables de plusieurs procédures dans le fichier inclus.

Ainsi le programme suivant:

```
0 (* 001 optin4 *)
1 (* 20 aou 87 *)
2
3 PROGRAM monolithe2;
4 TYPE t_tri = INTEGER;
5 VAR globale: INTEGER;
6     tri1: t_tri;
7
8 PROCEDURE deux;
9 BEGIN
10  END; (* deux *)
11
12 PROCEDURE tri;
13 BEGIN
14  END; (* tri *)
15
16 PROCEDURE liste_tri;
17 BEGIN
18  END; (* liste_tri *)
19
20 BEGIN (* principal *)
21  END.
```

Fig. 6.5 Le programme *monolithe2*.

peut être découpé en:

```
0 (* 001 optin5 *)
1 (* 20 aou 87 *)
2
3 PROGRAM modulaire
4 VAR globale: INTEGER;
5
6 PROCEDURE deux;
7 BEGIN
8  END; (* deux *)
9
10 (* $i tri *)
11
12 BEGIN (* principal *)
13  END.
```

Fig. 6.6 Le programme *modulaire*.

et:

```

0 (* 001 optin6 *)
1 (* 20 aou 87 *)
2
3 TYPE t_tri = INTEGER;
4 VAR tri1: t_tri;
5
6 PROCEDURE tri;
7 BEGIN
8 END; (* tri *)
9
10 PROCEDURE liste_tri;
11 BEGIN
12 END; (* liste_tri *)

```

Fig. 6.7 Le fichier inclus.

- vous pouvez utiliser plusieurs Include pour un même programme
- il n'est pas permis d'emboîter les Include. Seul le programme principal peut contenir des options Include
- si le compilateur ne trouve pas le fichier indiqué, la compilation est interrompue (erreur 90)
- si vous manquez d'espace mémoire au moment de la compilation ou de l'exécution en mémoire, vous pouvez essayer de réduire le programme principal à une liste d'options Include, en plaçant tout le texte PASCAL dans des fichiers inclus:

```

0 (* 001 gros *)
1 (* 20 aou 87 *)
2
3 (*$i ideclare *)
4 (*$i itri *)
5
6 (*$i C:\calculipour *)
7 (*$i ippal *)

```

Fig. 6.8 Le fichier principal avec uniquement des include

6.3 LA GESTION DES ERREURS: R, I, K, V

6.3.1 LA GESTION DES ERREURS

Le langage PASCAL avait au départ une vocation uniquement pédagogique. De ce fait WIRTH avait préconisé que toute erreur devait provoquer l'arrêt du programme.

Cette attitude ne peut être maintenue pour des programmes industriels. Il faut alors distinguer plusieurs catégories d'erreur:

- les erreurs de traitement qui ne mettent en jeu que les variables directement impliquées. Par exemple:

```
valeur := 1 / 0;
```

placera dans *valeur* une valeur erronée, mais ne modifiera pas d'autre variable du programme.

- les erreurs de traitement qui peuvent détruire d'autres zones de la mémoire. Ainsi:

```
VAR tableau: ARRAY[0..5] OF INTEGER;
```

```
tableau[-10000] := 1234;
```

peut détruire d'autres variables, le code du programme, ou même le DOS.

- les erreurs d'entrée / sortie. Par exemple l'utilisateur place dans le lecteur une disquette ne contenant pas le fichier qui doit être ouvert.

Les options R, I, K et V ont été créées pour permettre:

- de demander au compilateur de provoquer ou non des erreurs d'exécution
- de corriger par des instructions du programme les erreurs détectées

Ainsi, lors de la mise au point, il est utile que le compilateur ou le code généré provoquent des erreurs d'exécution pour nous signaler des erreurs telles que:

```
VAR jour: 1..31;
```

```
jour := 45;
```

Mais lorsque le programme est déverminé, aucune situation ne doit provoquer d'erreur. C'est au programmeur de tester ce qui se passe, et corriger les erreurs. Par exemple:

```

REPEAT
  WRITE('INDICE <0..5 ? ');
  READLN(indice);
UNTIL (indice >= 0) AND (indice <= 5);
tableau[indice] := 1234;

```

La différence entre les erreurs de traitement et les erreurs d'entrée / sortie est que les premières sont prévisibles, les secondes ne le sont pas. Nous pouvons vérifier si un indice est compris dans un intervalle donné, nous ne pouvons pas vérifier quelle disquette l'utilisateur va placer dans le lecteur.

Par conséquent:

- les options évitant l'arrêt en cas d'erreur de traitement sont en principe inutiles. Au contraire l'arrêt du programme est utile pour la détection des erreurs, lors de la mise au point.
- les options évitant l'arrêt en cas d'erreur d'entrée / sortie sont indispensables pour pouvoir tester et corriger l'erreur.

Voyons à présent ces options en détail.

6.3.2 LES OPTIONS R+ R-

6.3.2.1 DEFINITION DE L'OPTION R

L'option Range Check (vérification d'intervalle):

- permet d'effectuer ou non la vérification des bornes des intervalles
- est par défaut en mode R-
- est une option locale

Exemple:

(*R-*)

6.3.2.2 MOTIVATION

Lorsque nous affectons une valeur à une variable, le programme stocke les octets à affecter dans la zone allouée à la variable.

Il est donc impératif de s'assurer que le programme ne modifie pas d'autres octets que ceux utilisés pour cette variable.

Les risques de débordements proviennent des intervalles et des tableaux.

- les intervalles peuvent utiliser moins de 16 bits, et il ne faut pas que le code modifie les bits autres que ceux de l'intervalle.

```
VAR jour: 1..31;
    suite: INTEGER;
...
jour := valeur;
```

Si *valeur* contient par exemple \$1122, alors la variable *suite* sera modifiée par l'affectation ci-dessus.

- dans le cas des tableaux, l'adresse de la zone à modifier provient d'un calcul utilisant l'indice. Si l'indice est en dehors de l'intervalle déclaré, le code peut modifier d'autres données. Ainsi:

```
VAR tableau: ARRAY[5..15] OF INTEGER;
```

```
...
tableau[55] := 8;
```

ira écraser des données (des variables ou même du code) placés au-delà de la place allouée à *tableau*.

6.3.2.3 FONCTIONNEMENT

En mode R+ le compilateur place dans le code du programme des instructions qui vérifient que les valeurs affectées à un intervalle sont dans les limites déclarées.

Voici un programme utilisant l'option R+:

```
0 (* 001 opt r *)
1 (* 20 aou 87 *)
2
3 PROGRAM option_r;
4   VAR intervalle: $10..$1F;
5       entier: INTEGER;
6       tableau: ARRAY[$10..$1F] OF INTEGER;
7 BEGIN
8   intervalle := entier;
9   tableau[entier] := $AA;
10  (* $r+ *)
11  intervalle := entier;
12  tableau[entier] := $AA;
13 END.
```

Fig. 6.9 Le programme *option_r*.

Voici le code généré:

2D4D E849DD	call	TU_PARAGRAPHS
2D50 0600	hex	0600 ; options
2D52 780B	hex	780B ; cs
2D54 3C15	hex	3C15 ; ds
2D56 DC02	hex	DC02 ; code
2D58 2900	hex	2900 ; globales
2D5A 0004	hex	0004 ; min piles
2D5C 00A0	hex	00A0 ; max piles
2D5E 1000	hex	1000 ; fichiers
2D60 0000	hex	0000 ; input
2D62 0000	hex	0000 ; output


```

2D64 8BEC      mov     bp, sp
2D66 E814DF    call    TU_TAILLE_CODE
2D69 4700      hex     4700 ; taille
2D6B 6D2D      adr     SAUT_PRINCIPAL

2D6D E90000    SAUT_PRINCIPAL jmp     PRINCIPAL

2D70 A16302    PRINCIPAL   mov     ax, [$0263]
2D73 A26202    mov     [0262], al

2D76 A16302    mov     ax, [$0263]
2D79 D1E0      shl     ax, 1
2D7B 97        xchg    di
2D7C B8AA00    mov     ax, $00AA
2D7F 89854502  mov     [di+ $0245], ax

2D83 A16302    mov     ax, [$0263]
2D86 B91000    mov     cx, $0010
2D89 BA1F00    mov     dx, $001F
2D8C E8E5E1    call    TU_RANGE_CHECK

2D8F A26202    mov     [0262], al
2D92 A16302    mov     ax, [$0263]
2D95 2D1000    sub     ax, $0010
2D98 B91000    mov     cx, $0010
2D9B E8CCE1    call    TU_INDEX_CHECK
2D9E D1E0      shl     ax, 1
2DA0 97        xchg    di
2DA1 B8AA00    mov     ax, $00AA
2DA4 89856502  mov     [di+ $0265], ax

2DA8 E90000    jmp     FIN

2DAB 33C0      xor     ax, ax
2DAD E890DE    call    TU_HALT
2DB0 0000      hex     0000

```

Fig. 6.10 Code de option_r.

Notez que:

- en mode R- (mode par défaut):
 - l'affectation à une variable d'un octet (dans l'intervalle 0..255):
 - ne peut pas détruire les autres variables (MOV [adresse], AL)
 - le code ne vérifie pas que la valeur est réellement comprise dans l'intervalle déclaré (dans notre exemple, l'octet faible de *entier* peut très bien être en dehors de l'intervalle \$10..\$1F)
 - lorsque l'indice du tableau est en dehors de la plage déclarée, l'affectation ira écraser d'autres données situées dans DS (sans aucun message d'erreur à cet instant)
- en mode R+:
 - le code vérifie que les valeurs affectées à un intervalle sont bien comprises entre les bornes déclarées
 - le code vérifie que la valeur d'un indice est bien comprise entre les bornes déclarées

Voici un autre programme:

```

0  (* 001 opt2 *)
1  (* 20 aou 87 *)
2
3  (*$r+*)
4  PROGRAM option_r_2;
5  VAR intervalle: $10..$1F;
6  tableau: ARRAY($10..$1F) OF $40..$4F;
7  indice: INTEGER;
8  BEGIN
9  intervalle:= $BB;
10 READLN(intervalle);
11
12 (* refusé à la compilation : tableau[$CC]:= ... *)
13 tableau[indice]:= $AA;
14 END.

```

Fig. 6.11 Le programme option_r_2.

dont le code est le suivant:

```

2D4D E849DD    call    TU_PARAGRAPHS
2D50 0600      hex     0600 ; options
2D52 780B      hex     780B ; cs
2D54 3C15      hex     3C15 ; ds
2D56 DC02      hex     DC02 ; code
2D58 2800      hex     2800 ; globales
2D5A 0004      hex     0004 ; min piles
2D5C 00A0      hex     00A0 ; max piles
2D5E 1000      hex     1000 ; fichiers
2D60 0000      hex     0000 ; input
2D62 0000      hex     0000 ; output

2D64 8BEC      mov     bp, sp
2D66 E814DF    call    TU_TAILLE_CODE
2D69 4800      hex     4800 ; taille
2D6B 6D2D      adr     SAUT_PRINCIPAL

2D6D E90000    SAUT_PRINCIPAL jmp     PRINCIPAL

2D70 B8BB00    PRINCIPAL   mov     ax, $00BB
2D73 B91000    mov     cx, $0010
2D76 BA1F00    mov     dx, $001F
2D79 E8F8E1    call    TU_RANGE_CHECK
2D7C A26202    mov     [0262], al

2D7F E8CFF7    call    TU_READLN
2D82 BF6202    mov     di, $0262
2D85 1E        push    ds
2D86 E8B4F9    call    TU_READ_INTERV
2D89 E847E2    call    TU_IO_CHECK

```



```

2D8C A17302      mov     ax, [$0273]
2D8F 2D1000      sub     ax, $0010
2D92 B91000      mov     cx, $0010
2D95 E8D2E1      call    TU_INDEX_CHECK
2D98 97          xchg    di
2D99 B8AA00      mov     ax, $00AA
2D9C B94000      mov     cx, $0040
2D9F BA4F00      mov     dx, $004F
2DA2 E8CFE1      call    TU_RANGE_CHECK
2DA5 88856302     mov     [di+$0263], al

2DA9 E90000      jmp     FIN

2DAC 33C0        xor     ax, ax
2DAE E88FDE      call    TU_HALT
2DB1 0000        hex     0000

```

Fig. 6.12 Le code de *option_r_2*.

Notez que:

- le compilateur ne vérifie pas si les valeurs constantes sont dans l'intervalle. C'est au moment de l'exécution que la vérification sera effectuée (le compilateur attend en effet une expression qui peut contenir des variables. Il se place donc dans le cas général)
- le compilateur détecte qu'un indice constant débordé la plage (erreur 45). Il calcule en effet l'adresse lorsque l'indice est constant, et peut donc tester sa validité. Ceci est vrai en mode R- également.
- READ et READLN n'effectuent aucune vérification d'intervalle, sauf MAXINT (la frappe de 10000 est acceptée, la frappe de 40000 est refusée: erreur 10).
- Les calculs arithmétiques sur des entiers ne testent pas le débordement. 200*200 fournira une valeur de -25536 sans sourciller. C'est au programmeur de tester le débordement (en utilisant de réels pour vérifier si la valeur est inférieure ou égale à MAXINT).

Dans le cas de chaînes:

- le programme peut accéder à n'importe quel caractère (de 0 à *taille_max*), même en mode R+.

```

(*$r+*)
VAR prenom: STRING[9];
    indice: INTEGER;

```

```

prenom:= 'JOE';
preno[7]:= 'X';
prenom[0]:= CHR(77);
prenom[0]:= CHR(indice);

```

- l'affectation à une chaîne d'une chaîne plus grande que la taille maximale provoque une troncature (pour éviter d'écraser les zones situées après la chaîne), sans aucun message d'erreur

```

(*$r+*)
VAR prenom: STRING[3];

prenom:= 'ALEXANDRE';

```

6.3.2.4 UTILISATION

L'utilisation de R+ est impérative tant que le programme n'est pas totalement au point. Nous utilisons systématiquement cette option, que nous plaçons au début de chaque programme.

R- peut être utilisé:

- pour accéder aux cellules d'un tableau dont la taille est inconnue au moment de la compilation (tableaux dynamiques)
- pour éviter l'erreur d'exécution dans les tests composés utilisant AND:

```

(*$R-*)
WHILE (Indice< Maximum) AND (Tableau[Indice]<> 'Z') DO
(*$R+*)
BEGIN
...
Indice:= Indice+ 1;
END;

```

- pour accélérer la vitesse d'exécution ou diminuer la taille d'un programme. Ceci ne devrait être utilisé que dans les parties déjà parfaitement au point.

6.3.3 LES OPTIONS I+ I-

6.3.3.1 DEFINITION DE L'OPTION I

L'option I/O check (vérification des entrées / sorties):

- permet de supprimer l'interruption du programme lorsque survient une erreur d'entrée / sortie d'un fichier.
- par défaut utilise le mode I+ (vérification d'erreurs)
- l'option I peut être utilisée localement

Exemple:

```

(*$I+*)

```


6.3.3.2 MOTIVATION

A l'inverse des erreurs de traitement, les erreurs d'entrée/sortie sont hors du contrôle du programmeur. Il est possible de vérifier que le diviseur est différent de zéro. Il n'est pas possible de s'assurer que l'ascenseur de l'immeuble se met en route alors que le programme écrit sur le disque.

Nous avons vu que les entrées/sorties utilisaient toutes la librairie, qui elle-même fait appel au DOS.

En cas d'erreur, le DOS indique qu'il y a eu un problème, et la routine de la librairie provoque une erreur d'entrée / sortie.

Pour éviter que le programme ne soit interrompu, l'option I- ne provoque pas d'erreur d'exécution. Le programme peut continuer. Par contre c'est au programmeur de gérer l'erreur. Cette gestion est effectuée en lisant un code de résultat d'entrée / sortie à l'aide de la fonction IORESULT.

6.3.3.3 FONCTIONNEMENT

Voici un programme utilisant l'option I-:

```
0 (* 001 xopti *)
1 (* 16 oct 86 *)
2
3 (*$r+*)
4 PROGRAM option i;
5   VAR fichier: FILE OF INTEGER;
6   BEGIN
7     RESET(fichier);
8     (*$i-*)
9     RESET(fichier);
10  END.
```

Fig. 6.13 Le programme *option_i*.

Et voici son code:

```
2D4D E849DD      call    TU_PARAGRAPHS
2D50 0600      hex     0600 ; options
2D52 780B      hex     780B ; cs
2D54 3C15      hex     3C15 ; ds
2D56 DA02      hex     DA02 ; code
2D58 2B00      hex     2B00 ; globales
```

```
2D5A 0004      hex     0004 ; min piles
2D5C 00A0      hex     00A0 ; max piles
2D5E 1000      hex     1000 ; fichiers
2D60 0000      hex     0000 ; input
2D62 0000      hex     0000 ; output

2D64 8BEC      mov     bp, sp
2D66 E814DF     call    TU_TAILLE_CODE
2D69 2600      hex     2600 ; taille
2D6B 6D2D      adr     SAUT_PRINCIPAL

2D6D E90000     SAUT_PRINCIPAL jmp    PRINCIPAL

2D70 BF6202     PRINCIPAL   mov     di, $0262
2D73 1E        push    ds
2D74 B90200     mov     cx, $0002
2D77 E819FC     call    TU_RESET
2D7A E856E2     call    TU_IO_CHECK

2D7D BF6202     mov     di, $0262
2D80 1E        push    ds
2D81 B90200     mov     cx, $0002
2D84 E80CFC     call    TU_RESET

2D87 E90000     jmp     FIN

2D8A 33C0      FIN         xor     ax, ax
2D8C E8B1DE     call    TU_HALT
2D8F 0000      hex     0000
```

Fig. 6.14 Le code de *option_i*.

Notez que:

- en mode I+ (mode par défaut), le code appelle simplement la routine de la librairie TU_IO_CHECK. C'est cette routine qui provoque l'erreur d'entrée / sortie. En mode I-, cet appel n'est pas effectué.
- les instructions concernées par la vérification TU_IO_CHECK sont toutes les opérations qui accèdent au périphérique: RESET, REWRITE, READ, WRITE, SEEK, LONGSEEK, CLOSE, FLUSH, CHAIN, BLOCKREAD, BLOCKWRITE, ERASE, EXECUTE, RENAME, TRUNCATE, CHDIR, MKDIR, RMDIR.

Ces vérifications ont lieu pour tous les fichiers:

- fichier FILE OF,
- fichiers de blocs,
- fichiers TEXT,
- fichiers standard (CON, TRM, KBD, USR, LST, AUX),
- fichiers d'entrée / sortie par défaut (INPUT et OUTPUT)

Cette vérification même pour l'écran, permet de traiter les erreurs lorsque les fichiers ont été redirigés vers des périphériques qui peuvent provoquer des erreurs.

- Par contre les procédures suivantes ne sont pas concernées par la vérification d'entrée / sortie:
 - APPEND, GETDIR, OVRPATH, FILEPOS, LONGFILEPOS, FILESIZE, LONGFILESIZE, EOF, EOLN
 - les erreurs numériques:
- READLN(fichier, entier);
- provoquera une erreur d'entrée / sortie si le fichier n'est pas ouvert, mais une erreur numérique si la valeur est 4?(56. Les erreurs numériques NE PEUVENT PAS être supprimées par I-.
- pour ASSIGN, la vérification qu'un fichier FILE OF n'est pas redirigé vers un fichier standard (CON, LST...) place bien le code d'erreur \$22 dans la variable système, mais ne génère pas TU_IO_CHECK. Une erreur d'entrée / sortie \$22 est provoquée si vous travaillez en mode I+, mais mal repérée dans le texte du programme (positionnement à l'entrée / sortie précédente).
 - les routines d'entrée / sortie mettent à jour IORESULT systématiquement, que vous soyez en mode I+ ou I-.

La récupération du code d'erreur se fait de la façon suivante:

```
0 (* 001 xopti2 *)
1 (* 16 oct 86 *)
2
3 (*$r+*)
4 PROGRAM lis_ioresult;
5   VAR resultat: INTEGER;
6   BEGIN
7     resultat:= IORESULT;
8   END.
```

Fig. 6.15 Le programme lis_ioresult.

dont voici le code:

```
2D4D E849DD call TU_PARAGRAPHS
2D50 0600 hex 0600 ; options
2D52 780B hex 780B ; cs
2D54 3C15 hex 3C15 ; ds
2D56 D802 hex D802 ; code
2D58 2700 hex 2700 ; globales
2D5A 0004 hex 0004 ; min piles
2D5C 00A0 hex 00A0 ; max piles
2D5E 1000 hex 1000 ; fichiers
2D60 0000 hex 0000 ; input
2D62 0000 hex 0000 ; output

2D64 8BEC mov bp, sp
2D66 E814DF call TU_TAILLE_CODE
2D69 1500 hex 1500 ; taille
2D6B 6D2D adr SAUT_PRINCIPAL
```

```
2D6D E90000 SAUT_PRINCIPAL jmp PRINCIPAL
2D70 E859E2 PRINCIPAL call TU_IORESULT
2D73 A36202 mov [50262], ax
2D76 E90000 jmp FIN
2D79 33C0 FIN xor ax, ax
2D7B E8C2DE call TU_HALT
```

Fig. 6.16 Le code de lis_ioresult.

Notez que :

- le code invoque TU_IORESULT qui se contente de récupérer la valeur du résultat de l'entrée / sortie dans AL (0 dans AH) en mettant à zéro la variable qui contient ce code

6.3.3.4 UTILISATION

Voici par exemple comment ouvrir un fichier déjà créé en évitant que le programme ne s'arrête si le fichier n'est pas trouvé:

```
0 (* 001 xopti3 *)
1 (* 16 oct 86 *)
2
3 (*$r+*)
4 PROGRAM option i ouvre;
5   VAR fichier: FILE OF INTEGER;
6
7   PROCEDURE ouvre;
8     VAR l_resultat: INTEGER;
9     l_nom_fichier: STRING[64];
10    BEGIN
11      REPEAT
12        WRITE('nom <stock.dta> ? ');
13        READLN(l_nom_fichier);
14        ASSIGN(fichier, l_nom_fichier);
15        (*$i-*)
16        RESET(fichier);
17        (*$i+*)
18        l_resultat:= IORESULT;
19        IF l_resultat <> 0
20          THEN BEGIN
21            WRITELN('n''a pas pu ouvrir', l_nom_fichier);
22            CLOSE(fichier);
23            END;
24        UNTIL l_resultat= 0;
25      END; (* ouvre *)
26
```



```

27 BEGIN
28   ouvre;
29 END.

```

Fig. 6.17 Le programme *option_i_ouvre*.

Notez que:

- nous stockons le code du résultat de RESET dans une variable. En effet, si nous utilisons directement IORESULT dans la condition du IF, nous sortirions de la boucle même si le fichier n'était pas trouvé (IORESULT récupère le code ET met à zéro la zone qui contient ce code).
- lorsque le fichier n'est pas trouvé, il est prudent de refermer le fichier que nous avons essayé d'ouvrir. En TURBO ce n'est pas nécessaire.
- nous pourrions affiner le message en fonction du code d'erreur (fichier non trouvé, nom incorrect...)

Voici un autre exemple permettant de s'assurer si un fichier est déjà créé (pour éviter par exemple de le détruire par REWRITE):

```

0  (* 001 xopti4 *)
1  (* 16 oct 86 *)
2
3  (* $r+*)
4  PROGRAM option_i_existe;
5  TYPE string64 = STRING(64);
6  VAR fichier: FILE OF INTEGER;
7      nom_fichier: string64;
8
9  FUNCTION deja_ce_fichier(p_nom_fichier: string64): BOOLEAN;
10 BEGIN
11   ASSIGN(fichier, p_nom_fichier);
12   (* $i+*)
13   RESET(fichier);
14   (* $i+*)
15   IF IORESULT = 0
16   THEN deja_ce_fichier := TRUE
17   ELSE BEGIN
18     deja_ce_fichier := FALSE;
19     CLOSE(fichier);
20   END;
21   END; (* ouvre *)
22

```

```

23 BEGIN (* principal *)
24   WRITE('nom <stock.dta> ? ');
25   READLN(nom_fichier);
26   IF deja_ce_fichier(nom_fichier)
27   THEN BEGIN
28     WRITELN('ce fichier existe déjà');
29     EXIT;
30   END
31   ELSE BEGIN
32     REWRITE(fichier);
33     (* ... *)
34   END;
35 END.

```

Fig. 6.18 Le programme *option_i_existe*.

6.3.4 LES OPTIONS K+ K-

6.3.4.1 DEFINITION DE L'OPTION K

L'option K (vérification de pile):

- permet de supprimer la vérification de place sur la pile avant l'appel d'une procédure ou d'une fonction
- par défaut utilise le mode K+ (vérification de place)
- l'option K peut être utilisée localement

Exemple:

(* \$K-*)

6.3.4.2 MOTIVATION

Lorsque nous appelons une procédure, le programme place sur la pile:

- la sauvegarde de l'environnement de la procédure appelante (CP et son BP)
- la table d'accès aux piles locales intermédiaires (au moins le BP actuel)
- les variables locales s'il y en a
- les paramètres s'il y en a

Il convient donc de s'assurer qu'il y a encore assez de place en mémoire pour stocker ces données.

Par défaut le compilateur vérifie systématiquement que la pile permet d'accueillir un appel. L'option K- permet de supprimer cette vérification.

6.3.4.3 FONCTIONNEMENT

Voici un programme utilisant l'option K-:

```

0 (* 001 xoptk *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 PROGRAM option_k;
5
6   PROCEDURE deux;
7   BEGIN
8     END; (* deux *)
9
10  BEGIN (* deux *)
11    deux;
12    (*$k-*)
13    deux;
14  END.
```

Fig. 6.19 Le programme *option_k*.

Et voici son code:

```

2D4D E849DD      call    TU_PARAGRAPHS
2D50 0600      hex     0600; options
2D52 780B      hex     780B; cs
2D54 3C15      hex     3C15; ds
2D56 DA02      hex     DA02; code
2D58 2700      hex     2700; globales
2D5A 0004      hex     0004; min piles
2D5C 00A0      hex     00A0; max piles
2D5E 1000      hex     1000; fichiers
2D60 0000      hex     0000; input
2D62 0000      hex     0000; output

2D64 8BEC      mov     bp, sp
2D66 E814DF     call    TU_TAILLE_CODE
2D69 2900      hex     2900; taille
2D6B 6D2D      adr     SAUT_PRINCIPAL

2D6D E90000     SAUT_PRINCIPAL jmp    PRINCIPAL

2D70 55          DEUX      push   bp
2D71 8BEC      mov     bp, sp
2D73 55          push   bp
2D74 E90000     jmp     DEBUT_DEUX

2D77 E90000     DEBUT_DEUX jmp    FIN_DEUX
```

```

2D7A 8BE5      FIN_DEUX  mov     sp, bp
2D7C 5D        pop     bp
2D7D C3        ret

2D7E B90600     PRINCIPAL mov     cx, $0006
2D81 E8FEE1     call    TU_STACK_CHECK
2D84 E8E9FF     call    DEUX

2D87 E8E6FF     call    DEUX

2D8A E90000     jmp     FIN

2D8D 33C0      FIN       xor     ax, ax
2D8F E8AEDE     call    TU_HALT
```

Fig. 6.20 Le code de *option_k*.

Notez que:

- en mode K+ (mode par défaut), le code appelle simplement la routine de la librairie TU_STACK_CHECK, en lui indiquant dans CX le nombre d'octets dont la procédure aura besoin.

Cette option ne concerne pas l'utilisation de la mémoire par les pointeurs. Seuls les appels de procédures sont concernés.

6.3.4.4 UTILISATION

Les seuls programmes ayant à utiliser cette option sont ceux qui effectuent des appels à des procédures situées à un endroit différent que celui prévu normalement par la mécanique TURBO. Il s'agit là essentiellement de procédures placées dans des programmes rendus résidents et appelées par un autre programme. Ces techniques sont utilisées surtout pour gérer les interruptions ou dépasser la limite des 64K.

6.3.5 L'OPTION V

6.3.5.1 DEFINITION

L'option VERIFY:

- permet d'utiliser des paramètres variables STRING dont la taille est différente du paramètre actuel
- a par défaut la valeur V+
- est utilisable localement

Exemple:

(*\$V-*)

6.3.5.2 MOTIVATION

Considérons le programme suivant:

```

0 (* 001 xoptv *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 PROGRAM option_v;
5   TYPE string80= STRING[80];
6   VAR chaine3: STRING[3];
7
8   PROCEDURE lis_chaine(VAR pv_chaine: string80);
9   BEGIN
10    READLN(pv_chaine);
11    END; (* deux *)
12
13 BEGIN (* deux *)
14   lis_chaine(chaine3);
15 END.
```

Fig. 6.21 Le programme *option_v*.

Si l'utilisateur tape une chaîne de 30 caractères, les octets 4 à 30 iront écraser ce qui se trouve en mémoire après *chaine3*.

D'autre part le compilateur vérifie normalement que les types des paramètres formels et actuels sont identiques.

Pour permettre l'écriture de procédures génériques, le compilateur accepte des chaînes de tailles distinctes, à condition d'utiliser l'option V-.

6.3.5.3 FONCTIONNEMENT

L'option V- indique simplement au compilateur qu'il ne doit pas vérifier la concordance des tailles des chaînes.

Voici un exemple d'utilisation:

```

0 (* 001 xoptv *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 PROGRAM option_v;
5   TYPE string80= STRING[80];
6   VAR chaine3: STRING[3];
7   chaine127: STRING[127];
8
9   PROCEDURE lis_chaine(VAR pv_chaine: string80);
10  BEGIN
11    READLN(pv_chaine);
12    END; (* deux *)
13
14 BEGIN (* deux *)
15   (* refusé par le compilateur: lis_chaine(chaine127);
16                                     lis_chaine(chaine3); *)
17   (*$v.*)
18   lis_chaine(chaine3);
19 END.
```

Fig. 6.22 Le programme *option_v*.

Notez que:

- les option V+ ou V- ne produisent pas des codes différents. V- permet simplement de compiler des programmes rejetés par V+
- en TURBO il n'y a pas de risque qu'une lecture d'un paramètre variable chaîne écrase le paramètre actuel, puisque la lecture tronque la valeur lue si elle dépasse la taille prévue. V- permet simplement de rendre le programmeur conscient que certaines vérifications ne seront pas effectuées par le compilateur.

6.4 LE CLAVIER: B, D, C, U

6.4.1 LES OPTIONS B+ B-

6.4.1.1 DEFINITION DE L'OPTION B

L'option B (Buffer):

- permet de sélectionner la lecture clavier tamponnée ou standard
- est par défaut en mode B+
- est une option globale, qui doit obligatoirement figurer en tête de programme

Exemple:

(*\$B-*)

6.4.1.2 MOTIVATION

Pour essayer de faciliter la lecture du clavier, TURBO PASCAL utilise par défaut une lecture tamponnée du clavier. Dans cette mécanique:

- une instruction READ récupère tout ce qui est tapé jusqu'à la frappe de RETURN
- répartit les valeurs tapées aux variables contenues dans le READ. Si la ligne tapée ne contient pas assez de valeurs, TURBO place:
 - la valeur 26 (CTRL Z) dans les caractères
 - initialise les chaînes à la chaîne vide
 - ne modifie pas les valeurs numériques.

Considérons le programme suivant:

```

0 (* 001 xoptb3 *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 PROGRAM option b 3;
5   VAR nombre: INTEGER;
6       caractere1, caractere2, caractere3: CHAR;
7       chaine: STRING[3];
8   BEGIN
9       READ(caractere1, caractere2, caractere3);
10
11       READ(nombre, caractere1, chaine);
12   END.
```

Fig. 6.23 Le programme *option_b*.

Alors:

- Si vous tapez:

AB RETURN

alors *caractere1* contiendra A, *caractere2* contiendra B et *caractere3* contiendra CTRL Z.

Si vous tapez ensuite:

128 ESPACE OH RETURN

alors *nombre* contiendra 128, *caractere1* contiendra l'espace et *chaine* contiendra OH.

- si par contre vous tapez 128+OH une erreur sera provoquée car + n'est pas un chiffre compris entre 0 et 9.

Le PASCAL standard n'a jamais demandé que READ soit terminé par un RETURN avec modification des paramètres non contenus dans les caractères tapés. TURBO se rapproche un peu du fonctionnement du PASCAL standard lorsque vous vous placez en mode B-.

6.4.1.3 FONCTIONNEMENT

Voici un programme compilé en mode B+ (mode par défaut):

```

0 (* 001 xoptb *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 PROGRAM option b plus;
5   VAR entier: INTEGER;
6   BEGIN
7       READLN(entier);
8   END.
```

Fig. 6.24 Le programme *option_b_plus*.

et voici le code généré:

2D4D E849DD	call	TU_PARAGRAPHS
2D50 0600	hex	0600 ; options
2D52 780B	hex	780B ; cs
2D54 3C15	hex	3C15 ; ds
2D56 D902	hex	D902 ; code
2D58 2700	hex	2700 ; globales
2D5A 0004	hex	0004 ; min piles
2D5C 00A0	hex	00A0 ; max piles
2D5E 1000	hex	1000 ; fichiers
2D60 0000	hex	0000 ; input
2D62 0000	hex	0000 ; output
2D64 8BEC	mov	bp, sp
2D66 E814DF	call	TU_TAILLE_CODE
2D69 1C00	hex	1C00 ; taille
2D6B 6D2D	adr	SAUT_PRINCIPAL
2D6D E90000	SAUT_PRINCIPAL	jmp
2D70 E8DEF7	PRINCIPAL	call
2D73 BF6202		mov
2D76 1E		push
2D77 E8C6F9		call
2D7A E856E2		call


```

2D7D E90000          jmp     FIN
2D80 33C0            FIN      xor     ax, ax
2D82 E8BBDE          call    TU_HALT

```

Fig. 6.25 Le code de *option_b_plus*.

Voici le même programme compilé en B-:

```

0 (* 001 xopib *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 (*$b-*)
5 PROGRAM option b moins;
6   VAR entier: ENTIER;
7   BEGIN
8     READLN(entier);
9   END.

```

Fig. 6.26 Le programme *option_b_moins*.

dont voici le code:

```

2D4D E849DD          call    TU_PARAGRAPHS
2D50 0600            hex      0600 ; options
2D52 780B            hex      780B ; cs
2D54 3C15            hex      3C15 ; ds
2D56 D902            hex      D902 ; code
2D58 2700            hex      2700 ; globales
2D5A 0004            hex      0004 ; min piles
2D5C 00A0            hex      00A0 ; max piles
2D5E 1000            hex      1000 ; fichiers
2D60 0000            hex      0000 ; input
2D62 0000            hex      0000 ; output

2D64 8BEC            mov     bp, sp
2D66 E814DF          call    TU_TAILLE_CODE
2D69 1F00            hex      1F00 ; taille
2D6B 6D2D            adr      SAUT_PRINCIPAL

2D6D E90000          SAUT_PRINCIPAL jmp     PRINCIPAL
2D70 E880F7          PRINCIPAL call    TU_READ
2D73 BF6202          mov     di, $0262
2D76 1E              push    ds
2D77 E8C6F9          call    TU_READ_ENTIER
2D7A E854FA          call    TU_READLN_VAR
2D7D E853E2          call    TU_IO_CHECK
2D80 E90000          jmp     $2D83

```

```

2D83 33C0            LR_2D83 xor     ax, ax
2D85 E8B8DE          call    TU_HALT

```

Fig. 6.27 Le code de *option_b_moins*.

6.4.1.4 UTILISATION

Ni B+ ni B- ne sont conformes au PASCAL standard. Lequel n'avait d'ailleurs pas été prévu pour fonctionner avec des terminaux interactifs. Aucune solution n'est donc parfaite.

Sachant que notre solution n'est pas meilleure que les autres, voici comment nous procédons:

- nous utilisons KBD qui n'est jamais tamponné, en affichant ce qui est tapé par WRITE
- nous n'utilisons jamais plus d'une variable dans un READ ou un READLN (il faut de toutes façons afficher à l'écran ce que l'utilisateur doit taper avant chaque lecture)
- pour les petits programmes d'essai, nous utilisons READ pour les caractères (frappe d'une touche sans avoir besoin de RETURN) et READLN pour les valeurs numériques et les chaînes
- pour les programmes qui doivent fonctionner sans erreur, nous utilisons:
 - READ pour les caractères (frappe d'une touche sans nécessité de taper RETURN).
 - READLN pour les chaînes
 - des procédures de saisie contrôlées pour les saisies numériques (lecture caractère par caractère avec contrôle à chaque frappe et conversion numérique, gestion du curseur...). De telles procédures pour les entiers et les réels ont été publiées dans TOPIQUES PASCAL en PASCAL UCSD, ou dans PASCALISIME.

Attention, avec KBD, EOLN ne fonctionne pas correctement (retard d'un caractère). Il faut tester CHR(13) pour détecter la frappe de RETURN.

6.4.2 LES OPTIONS D+ ET D-

6.4.2.1 DEFINITION

L'option D (Device):

- permet de modifier la gestion des fichiers TEXT
- par défaut fonctionne en mode D+
- doit être placée en début de programme

Exemple:

```
(*$D-*)
```


6.4.2.2 MOTIVATION

Les fichiers TEXT peuvent être des fichiers disque ou des fichiers de caractère. Les fichiers disque utilisent un tampon, alors que les fichiers de caractères travaillent octet par octet. Lorsqu'une ouverture d'un fichier TEXT est rencontrée, le programme teste si ce fichier est associé à un fichier disque ou un périphérique. Dans le second cas, le mécanisme du tampon n'est pas utilisé.

L'option D- permet de ne pas effectuer le test du type de périphérique au moment du RESET, REWRITE ou APPEND. Dans ce cas toutes les entrées / sorties seront tamponnées.

6.4.2.3 FONCTIONNEMENT

Voici un programme utilisant l'option D-:

```
0 (* 001 xoptd *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 (*$d-*)
5 PROGRAM option_b;
6 BEGIN
7 END.
```

Fig. 6.28 Le programme option_d.

et voici son code:

```
2D4D E849DD      call    TU_PARAGRAPHS
2D50 0400      hex     0400 ; options
2D52 780B      hex     780B ; cs
2D54 3C15      hex     3C15 ; ds
2D56 D802      hex     D802 ; code
2D58 2700      hex     2700 ; globales
2D5A 0004      hex     0004 ; min piles
2D5C 00A0      hex     00A0 ; max piles
2D5E 1000      hex     1000 ; fichiers
2D60 0000      hex     0000 ; input
2D62 0000      hex     0000 ; output

2D64 8BEC      mov     bp, sp
2D66 E814DF      call    TU_TAILLE_CODE
2D69 0F00      hex     0F00 ; taille
2D6B 6D2D      adr     SAUT_PRINCIPAL
```

```
2D6D E90000      SAUT_PRINCIPAL jmp    PRINCIPAL
2D70 E90000      PRINCIPAL      jmp    FIN
2D73 33C0      FIN              xor     ax, ax
2D75 E8C8DE      call    TU_HALT
```

Fig. 6.29 Le code utilisant l'option D-.

Notez que:

- la seule différence par rapport à un programme sans D- est la valeur du paramètre en \$2D50. C'est donc la routine TU_PARAGRAPHS qui récupère la valeur du bit 3 de ce paramètre, et modifie le comportement de READ et WRITE en conséquence.

Un exemple utilisant D est présenté au paragraphe 6.5.2.

6.4.3 LES OPTIONS C+ C-

6.4.3.1 DEFINITION

L'option C (Contrôle):

- permet de supprimer le test de CTRL C et CTRL S
- a la valeur C+ par défaut
- est une option globale qui doit être placée au début du programme

Exemple:

```
(*$C-*)
```

6.4.3.2 MOTIVATION

La frappe de CTRL C et CTRL S permet à l'utilisateur de contrôler l'exécution du programme:

- lorsque le programme exécute un READ ou un READLN et que l'utilisateur tape CTRL C, l'exécution est interrompue
- lorsque le programme exécute un WRITE ou un WRITELN vers l'écran, CTRL S permet de bloquer temporairement l'exécution. La frappe de n'importe quelle touche permet alors de relancer le programme.

CTRL C permet donc d'arrêter un programme. CTRL S permet d'effectuer une pause pour pouvoir examiner des résultats affichés à l'écran.

Pour éviter qu'un utilisateur n'interrompe par mégarde un programme qui doit se terminer normalement (données en mémoire qui doivent être mises à jour sur disque par exemple) ou qu'il ne croie que le programme s'est bloqué, il est possible d'annuler l'interprétation de ces touches en utilisant l'option C-.

6.4.3.3 FONCTIONNEMENT

Voici un programme compilé en C-

```
0 (* 001 xoptc *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 (*$c-*)
5 PROGRAM option c;
6   VAR entier: ENTIER;
7   BEGIN
8     READLN(entier);
9   END.
```

Fig. 6.30 Le programme *option_c*.

et voici son code:

```
2D4D E849DD      call    TU_PARAGRAPHS
2D50 0200      hex     0200 ; options
2D52 780B      hex     780B ; cs
2D54 3C15      hex     3C15 ; ds
2D56 D902      hex     D902 ; code
2D58 2700      hex     2700 ; globales
2D5A 0004      hex     0004 ; min piles
2D5C 00A0      hex     00A0 ; max piles
2D5E 1000      hex     1000 ; fichiers
2D60 0000      hex     0000 ; input
2D62 0000      hex     0000 ; output

2D64 8BEC      mov     bp, sp
2D66 E814DF      call    TU_TAILLE_CODE
2D69 1C00      hex     1C00 ; taille
2D6B 6D2D      adr     SAUT_PRINCIPAL

2D6D E90000      SAUT_PRINCIPAL jmp    PRINCIPAL
```

```
2D70 E8DEF7      PRINCIPAL call    TU_READLN
2D73 BF6202      mov     di, $0262
2D76 1E          push    ds
2D77 E8C6F9      call    TU_READ_ENTIER
2D7A E856E2      call    TU_IO_CHECK

2D7D E90000      jmp     FIN

2D80 33C0      FIN      xor     ax, ax
2D82 E8BBDE      call    TU_HALT
```

Fig. 6.31 Le code de *option_c*.

Notez que:

- la seule différence par rapport au même programme est la valeur du paramètre en \$2D50. C'est donc la routine TU_PARAGRAPHS qui récupère la valeur du bit 1 de ce paramètre, et modifie le comportement de READ et WRITE

6.4.3.4 UTILISATION

L'utilisation de l'option C-:

- accélère l'affichage écran
- permet l'utilisation de KEYPRESSED

6.4.4 LES OPTIONS U+ ET U-

6.4.4.1 DEFINITION

L'option U (User interrupt):

- permet d'interrompre le programme par la frappe de CTRL C
- a par défaut la valeur U-
- doit être placée au début du programme.

Exemple:

```
(*$U+*)
```

6.4.4.2 MOTIVATION

Normalement CTRL C ne permet d'interrompre un programme que lorsque le programme exécute un READ ou un READLN et que l'utilisateur tape CTRL C.

Une boucle telle que:

```
REPEAT
UNTIL TRUE;
```

ne peut être interrompue qu'en utilisant CTRL ALT DEL ou en débranchant la machine.

L'option U+ permet d'interrompre le programme à n'importe quel moment.

6.4.4.3 FONCTIONNEMENT

Voici un programme comportant l'option U+:

```
0 (* 001 xoptu *)
1 (* 20 aou 87 *)
2
3 (* $r+*)
4 (* $u+*)
5 PROGRAM option_u;
6   VAR entier: INTÉGER;
7   BEGIN
8     REPEAT
9       entier:= 5;
10    UNTIL TRUE;
11  END.
```

Fig. 6.32 Le programme *option_u*.

et voici son code:

```
2D4D E849DD      call    TU_PARAGRAPHS
2D50 0E00        hex     0E00 ; options
2D52 780B        hex     780B ; cs
2D54 3C15        hex     3C15 ; ds
2D56 D902        hex     D902 ; code
2D58 2700        hex     2700 ; globales
2D5A 0004        hex     0004 ; min piles
2D5C 00A0        hex     00A0 ; max piles
2D5E 1000        hex     1000 ; fichiers
2D60 0000        hex     0000 ; input
2D62 0000        hex     0000 ; output

2D64 8BEC        mov     bp, sp
2D66 E814DF      call    TU_TAILLE_CODE
2D69 2300        hex     2300 ; taille
2D6B 6D2D        adr     SAUT_PRINCIPAL
```

```
2D6D E90000      SAUT_PRINCIPAL  jmp    PRINCIPAL
2D70
2D70 CC          PRINCIPAL
                DEBUT_REPEAT  int    3
2D71 CC          int    3
2D72 B80500      mov     ax, $0005
2D75 A36202      mov     [0262], ax
2D78 CC          int    3
2D79 B80100      mov     ax, $0001
2D7C 0BC0        or      ax, ax
2D7E 7503        jnz     FIN_REPEAT
2D80 E9EEFF      jmp     DEBUT_REPEAT
2D83 CC          FIN_REPEAT  int    3
2D84 E90000      jmp     FIN
2D87 33C0        FIN        xor     ax, ax
2D89 E8B4DE      call    TU_HALT
```

Fig. 6.33 Le code de *option_u*.

Notez que:

- en mode U+, le paramètre en \$2D50 force TU_PARAGRAPHS à implanter le test de CTRL C au vecteur d'interruption3
- par la suite, le code testera systématiquement la frappe ce CTRL C avant chaque instruction. Ce test peut même être effectué plusieurs fois de suite lorsque plusieurs instructions commencent au même endroit

6.4.4.4 UTILISATION

Le mode U+ peut être utilisé au cours du développement, lorsque le programme se bloque dans des boucles infinies. Comme il ralentit l'exécution, il n'est en général pas maintenu dans la version finale du programme.

6.5 LES FICHIERS: F, G, P**6.5.1 L'OPTION F****6.5.1.1 DEFINITION DE L'OPTION F**

L'option F (Files):

- permet d'ouvrir un plus grand nombre de fichiers simultanément
- est une option globale, qui doit obligatoirement figurer en tête de programme

Exemple:

```
(*F17*)
```

6.5.1.2 MOTIVATION

Chaque fichier ouvert nécessite une zone de données contenant les informations nécessaires à la gestion de ce fichier (nom, taille, date de dernière modification...). Pour économiser la place mémoire, le DOS et TURBO ont limité le nombre de fichiers qui peuvent être ouverts à un instant donné.

Si votre programme a besoin de traiter plus de fichiers que le nombre prévu par défaut, il faut le préciser en utilisant l'option F.

6.5.1.3 FONCTIONNEMENT

L'option F modifie simplement la valeur du paramètre situé en \$2D5E. La routine TU_PARAGRAPHS augmente alors la taille de la table des handle utilisés (\$0242). Le compilateur positionnera alors la première variable globale non pas à \$0262 mais après le handle du dernier fichier demandé.

Pour que cette option fonctionne, il faut aussi demander au DOS d'accommoder plus de fichiers. Cela doit être effectué au moyen du fichier CONFIG.SYS.

Mentionnons aussi que rares sont les logiciels nécessitant plus de 4 ou 5 fichiers disques ouverts à un même instant. Avant de recourir à l'option F, consultez peut-être un bon ouvrage sur la gestion de fichiers...

6.5.2 LES OPTIONS G ET P**6.5.2.1 DEFINITION**

Les options G (Get) et P (Put):

- permettent de rediriger les fichiers d'entrée et de sortie
- doivent être placés en début du programme

Exemple:

```
(*G$12, P1024 *)
```

6.5.2.2 MOTIVATION

Par défaut, le fichier utilisé par READ est le clavier, et le fichier utilisé par WRITE est l'écran.

Les options G et P permettent d'utiliser comme fichier d'entrée par défaut ou de sortie par défaut d'autres périphériques.

6.5.2.3 UTILISATION

Voici un programme qui lit un fichier et met les lettres minuscules en majuscules:

```
0 (* 001 xoptgp *)
1 (* 20 aou 87 *)
2
3 (*$r+*)
4 (*$g$12,p$12,d-*)
5 PROGRAM option_gp;
6   VAR caractere: CHAR;
7   BEGIN
8     WHILE NOT EOF DO
9       BEGIN
10        READ(caractere);
11        WRITE(UPCASE(caractere));
12      END;
13    END.
```

Fig. 6.34 Le programme *option_gp*.

Pour utiliser ce programme, il faudra le compiler sur disque, puis taper, depuis la ligne de commande du DOS:

optiongp <source> destination

6.6 CONCLUSION

Dans ce chapitre nous avons examiné:

- l'option Include qui permet d'utiliser des textes de n'importe quelle taille
- les options R, I, K, V qui permettent la gestion des erreurs
- les options U, C, B et D qui modifient le comportement des fichiers d'entrée / sortie
- les options F G et P qui permettent de modifier le nombre et la nature des fichiers utilisés par défaut.

Exemple:

Par défaut, le fichier utilisé par READ est le clavier, et le fichier utilisé par WRITE est le disque.

Chaque fichier ouvert reçoit une adresse de données. Les données sont envoyées au fichier d'entrée par défaut ou au fichier d'entrée par défaut.

Si votre programme a besoin de traiter plus de fichiers, le nombre de fichiers à traiter doit être spécifié dans le programme.

6.6.1 FONCTIONNEMENT

L'option F modifie l'implémentation de la valeur du paramètre FILE. Les données sont envoyées au fichier d'entrée par défaut ou au fichier d'entrée par défaut.

Pour que cette option fonctionne, il faut spécifier le nombre de fichiers à traiter dans le programme.

Le nombre de fichiers à traiter est spécifié dans le programme. Les données sont envoyées au fichier d'entrée par défaut ou au fichier d'entrée par défaut.

CHAPITRE 7

LA LIBRAIRIE

Ce chapitre va présenter:

- le contenu de la librairie
- les variables globales utilisées

A la fin de ce chapitre:

- vous connaîtrez les points d'entrée des routines utilisées dans le code compilé
- vous saurez comment sont passés les paramètres aux routines de la librairie
- vous pourrez afficher le contenu des variables globales utilisées par tout code compilé

7.1 LA LIBRAIRIE ET LES GLOBALES

Nous avons vu au chapitre précédent que le code compilé comportait:

- des traitements effectués directement en utilisant les registres du 8088
- des appels à des routines de la librairie

La librairie comporte des groupes de routines chargées de gérer:

- les entrées / sorties
- les pointeurs
- les traitements divers (OVERLAYS, STR et VAL, erreurs, nombres aléatoires)
- les chaînes
- les ensembles
- les réels
- les fichiers (TEXT, READ / WRITE, FILE OF, FILE, directories)
- le chaînage des programmes

Nous allons présenter dans ce chapitre toutes les routines que le compilateur peut invoquer. Pour chacune de ces routines nous indiquerons quels paramètres elle reçoit et comment elle retourne une valeur.

Tout programme TURBO PASCAL comporte aussi des variables globales. Ces données sont utilisées:

- pour gérer les périphériques
- comme variables temporaires des routines de la librairie

Nous présenterons par conséquent ces données, en fournissant un programme qui permet d'en afficher le contenu.

7.2 LES DONNEES GLOBALES**7.2.1 LA LISTE DES DONNEES GLOBALES**

Voici la liste des variables que le programme peut utiliser:

- v_attr_norm**
 - adresse: \$0000
 - valeur de l'attribut de vidéo normale
- v_attr_low**
 - adresse: \$0001
 - valeur de l'attribut de vidéo faible
- v_reverse**
 - adresse: \$0003
 - valeur de l'attribut de vidéo inverse

- v_fenetre_x_bas**
 - adresse: \$0004
 - marge gauche de la fenêtre texte

- v_fenetre_y_bas**
 - adresse: \$0005
 - haut de l'écran

- v_mode_ecran**
 - adresse: \$0006
 - type d'affichage

- v_couleur**
 - adresse: \$0007
 - couleur actuelle

- v_attribut**
 - adresse: \$0008
 - attribut d'affichage courant

- v_mode_graphique**
 - adresse: \$0009
 - mode graphique courant

- v_fen_graphxdeb**
 - adresse: \$000A
 - marge gauche de la fenêtre graphique

- v_fen_graphxfin**
 - adresse: \$000C
 - marge droite de la fenêtre graphique

- v_fen_graphydeb**
 - adresse: \$000E
 - haut de la fenêtre graphique

- v_fen_graphyfin**
 - adresse: \$0010
 - bas de la fenêtre graphique

- v_delai_1**
 - adresse: \$0012
 - variable utilisée dans le calcul du délai

- v_draw_1**
 - adresse: \$0014
 - variable temporaire utilisée pour le tracé de droite graphique

- v_draw_4**
 - adresse: \$0016
 - variable temporaire utilisée pour le tracé de droite graphique

- v_draw_3**
 - adresse: \$0018
 - variable temporaire utilisée pour le tracé de droite graphique

v_draw_6

- adresse: \$001A
- variable temporaire utilisée pour le tracé de droite graphique

v_draw_2

- adresse: \$001C
- variable temporaire utilisée pour le tracé de droite graphique

v_draw_5

- adresse: \$001E
- variable temporaire utilisée pour le tracé de droite graphique

v_palette

- adresse: \$0020
- palette courante

v_delai_2

- adresse: \$0021
- variable utilisée dans le calcul du délai

v_ofs_deb_free

- adresse: \$0024
- décalage de la liste des zones du tas libérées par **DISPOSE**

v_seg_deb_free

- adresse: \$0026
- segment de la liste des zones du tas libérées par **DISPOSE**

v_ofs_pt_deb_fr

- adresse: \$0028
- décalage du pointeur de la liste des zones du tas libérées par **DISPOSE**

v_seg_pt_deb_fr

- adresse: \$002A
- segment du pointeur de la liste des zones du tas libérées par **DISPOSE**

v_ofs_pt1

- adresse: \$002C
- décalage d'un pointeur utilisé pour les calculs d'allocation

v_seg_pt1

- adresse: \$002E
- segment d'un pointeur utilisé pour les calculs d'allocation

v_ofs_pointeur

- adresse: \$0030
- décalage d'un pointeur utilisé pour les calculs d'allocation

v_seg_pointeur

- adresse: \$0032
- segment d'un pointeur utilisé pour les calculs d'allocation

v_ofs_pt3

- adresse: \$0034
- décalage d'un pointeur utilisé pour les calculs d'allocation

v_seg_pt3

- adresse: \$0036
- segment d'un pointeur utilisé pour les calculs d'allocation

v_tampon_read

- adresse: \$0038
- tampon de lecture clavier

v_tampon_wrk

- adresse: \$00B8
- tampon général

v_adr_test_con

- adresse: \$0138
- décalage dans CS de la routine de test du clavier

v_adr_lis_con

- adresse: \$013A
- décalage dans CS de la routine de lecture du clavier

v_adr_ecris_con

- adresse: \$013C
- décalage dans CS de la routine d'affichage écran

v_adr_ecris_lst

- adresse: \$013E
- décalage dans CS de la routine d'écriture imprimante

v_adr_ecris_aux

- adresse: \$0140
- décalage dans CS de la routine d'écriture série

v_adr_lis_aux

- adresse: \$0142
- décalage dans CS de la routine de lecture série

v_adr_ecris_usr

- adresse: \$0144
- décalage dans CS de la routine d'écriture série 2

v_adr_lis_usr

- adresse: \$0146
- décalage dans CS de la routine de lecture série 2

v_handle_read

- adresse: \$015C
- handle de lecture

v_read_write

- adresse: \$015E
- temporaire pour distinguer la lecture ou l'écriture

v_debut_glob_pg
- adresse: \$0160
- décalage dans DS du début des globales utilisateur

v_sv_input
- adresse: \$0162
- sauvegarde de la taille du tampon précisé avec l'option G

v_handle_write
- adresse: \$0168
- handle d'écriture

v_colonne_max
- adresse: \$016A
- marge droite de l'écran texte

v_ligne_max
- adresse: \$016B
- dernière ligne de l'écran texte

v_debut_glob_2
- adresse: \$016C
- décalage dans DS du début des globales utilisateur

v_sv_output
- adresse: \$016E
- sauvegarde de la taille du tampon précisé avec l'option P

v_test_device
- adresse: \$0174
- variable contenant l'option D

v_turbo_cs
- adresse: \$0176
- segment CS de TURBO.COM lors de l'exécution en mémoire

v_turbo_ds
- adresse: \$0178
- segment DS de TURBO.COM lors de l'exécution en mémoire

pt_table_handle
- adresse: \$017A
- décalage du début de la table des handle

v_nb_fichiers
- adresse: \$017C
- nombre maximum de fichiers ouverts à un instant (option F)

v_adr_proc_err
- adresse: \$017E
- décalage dans CS de la routine d'erreur

v_num_err_exec
- adresse: \$0180
- code de l'erreur

v_ind_read_max
- adresse: \$0181
- indice maximal du tampon de lecture

v_ind_tamp_read
- adresse: \$0182
- indice courant du tampon de lecture

v_taille_tmp_rd
- adresse: \$0184
- indice du dernier caractère lu

v_find_error_pc
- adresse: \$0188
- PC d'une erreur

v_reg_186
- adresse: \$0186
- variable utilisée pour effectuer les retours lorsque la pile contient des paramètres qu'il faut d'abord dépiler

v_ofs_tas
- adresse: \$018A
- décalage du sommet du tas (HEAPPTR)

v_seg_tas
- adresse: \$018C
- segment du sommet du tas (HEAPPTR)

v_sv_sp_2
- adresse: \$018E
- sauvegarde de SP pour pouvoir restaurer la pile lors du retour dans TURBO.COM

v_sv_ofs_int_0
- adresse: \$0190
- sauvegarde du décalage du vecteur \$00 au moment du lancement du programme

v_sv_seg_int_0
- adresse: \$0192
- sauvegarde du segment du vecteur \$00 au moment du lancement du programme

v_car_keypressd
- adresse: \$0194
- scan code du caractère tapé au moment de KEYPRESSED

v_clavier
- adresse: \$0196
- étiquette du clavier d'entrée (option C)

v_tampon_ovl
- adresse: \$0198
- tampon contenant le nom d'un fichier OVERLAY

v_taille_string
- adresse: \$01E8
- temporaire utilisée par les routines VAL et STR

- v_val_ent_reel**
 - adresse: \$01E9
 - temporaire utilisée par les routines VAL et STR
- v_ofs_string**
 - adresse: \$01EA
 - temporaire utilisée par les routines VAL et STR
- v_seg_string**
 - adresse: \$01EC
 - temporaire utilisée par les routines VAL et STR
- v_ofs_ent_reel2**
 - adresse: \$01EE
 - temporaire utilisée par les routines VAL et STR
- v_seg_ent_reel2**
 - adresse: \$01F0
 - temporaire utilisée par les routines VAL et STR
- v_ofs_resu_val**
 - adresse: \$01F2
 - temporaire utilisée par les routines VAL et STR
- v_seg_resu_val**
 - adresse: \$01F4
 - temporaire utilisée par les routines VAL et STR
- v_tamp_reel1**
 - adresse: \$01F6
 - temporaire utilisée par les routines VAL et STR
- v_ofs_random**
 - adresse: \$01FE
 - décalage du nombre aléatoire
- v_erreur_i_o**
 - adresse: \$01FC
 - code d'erreur d'entrée / sortie
- v_seg_random**
 - adresse: \$0200
 - segment du nombre aléatoire
- v_taille_ch_des**
 - adresse: \$0202
 - temporaire utilisée par les routines de gestion de chaînes
- v_ind_ins_del**
 - adresse: \$0204
 - temporaire utilisée par les routines de gestion de chaînes
- v_qt_delete**
 - adresse: \$0206
 - temporaire utilisée par les routines de gestion de chaînes

- v_ofs_chai_dest**
 - adresse: \$0208
 - temporaire utilisée par les routines de gestion de chaînes
- v_seg_chai_dest**
 - adresse: \$020A
 - temporaire utilisée par les routines de gestion de chaînes
- v_ofs_chai_depart**
 - adresse: \$020C
 - temporaire utilisée par les routines de gestion de chaînes
- v_seg_chai_depart**
 - adresse: \$020E
 - temporaire utilisée par les routines de gestion de chaînes
- v_ofs_fichier**
 - adresse: \$0234
 - décalage du fichier utilisé pour READ ou WRITE
- v_seg_fichier**
 - adresse: \$0236
 - segment du fichier utilisé pour READ ou WRITE
- v_reset_write_a**
 - adresse: \$023A
 - variable pour distinguer RESET, REWRITE ou APPEND
- v_err_rd_wr_fof**
 - adresse: \$023B
 - code d'erreur
- v_tamp_lis_reel**
 - adresse: \$023C
 - tampon utilisé pour la conversion numérique
- v_table_handle**
 - adresse: \$0242
 - table des handles des fichiers

7.2.2 LE PROGRAMME D'AFFICHAGE DES GLOBALES

Voici un programme permettant l'affichage des variables globales:

```

0 (* 003 globals *)
1 (* 22 au 87 *)
2
3 (*$r+*)
4 PROGRAM affiche_globales;
5   TYPE string80= STRING[80];
6   VAR choix: CHAR;
7       pointeur1, pointeur2: ^INTEGER;
8       fichier: FILE OF INTEGER;
9       chaine, chaine1, chaine2, chaine3: string80;
10      reel: REAL;
11      resultat_val: INTEGER;
```



```

12      sortie: TEXT;
13      nom_sortie: STRING[24];
14
15  PROCEDURE affiche_octet_hexadecimal(p_octet: INTEGER);
16  CONST kt_hex: ARRAY[0..15] OF CHAR = '0123456789ABCDEF';
17  BEGIN
18    WRITE(sortie, kt_hex[p_octet DIV 16], kt_hex[p_octet MOD 16]);
19    END; (* affiche_octet_hexadecimal *)
20
21  PROCEDURE affiche_entier_hexadecimal(p_entier: INTEGER);
22  BEGIN
23    affiche_octet_hexadecimal((p_entier AND $FF00) SHR 8);
24    affiche_octet_hexadecimal(p_entier AND $00FF);
25    END; (* affiche_entier_hexadecimal *)
26
27  PROCEDURE affiche_memoire(l_segment, l_alignement_debut,
28    l_alignement_fin: INTEGER);
29  VAR l_colonne: INTEGER;
30    l_alignement, l_quantite: INTEGER;
31    l_octet: BYTE;
32  BEGIN
33    l_alignement := l_alignement_debut AND $0007;
34    l_alignement_debut := l_alignement_debut AND $FFF8;
35    l_quantite := l_alignement_fin + 1 - l_alignement_debut;
36
37    REPEAT
38      WRITE(sortie, '$'); affiche_entier_hexadecimal(l_segment);
39      WRITE(sortie, ':$');
40      affiche_entier_hexadecimal(l_alignement_debut); WRITE(sortie, ' ');
41
42      IF l_alignement > 0
43        THEN BEGIN
44          WRITE(sortie, '': 4 * l_alignement);
45          l_colonne := l_alignement;
46          END
47        ELSE l_colonne := 0;
48      REPEAT
49        l_octet := mem[l_segment: l_alignement_debut + l_colonne];
50        WRITE(sortie, '$');
51        affiche_octet_hexadecimal(l_octet); WRITE(sortie, ' ');
52        l_colonne := l_colonne + 1;
53      UNTIL (l_colonne > 7) OR (l_quantite - l_colonne <= 0);
54
55      WRITE(sortie, '': 4 * (8 - l_colonne), ' ');
56      IF l_alignement > 0
57        THEN BEGIN
58          WRITE(sortie, '': l_alignement);
59          l_colonne := l_alignement;
60          l_alignement := 0;
61          END
62      ELSE l_colonne := 0;

```

```

63  REPEAT
64    l_octet := mem[l_segment: l_alignement_debut + l_colonne];
65    IF l_octet IN [32..127]
66      THEN WRITE(sortie, CHR(l_octet))
67      ELSE WRITE(sortie, '.');
68    l_colonne := l_colonne + 1;
69  UNTIL (l_colonne > 7) OR (l_quantite - l_colonne <= 0);
70  WRITELN(sortie);
71
72  l_quantite := l_quantite - 8;
73  l_alignement_debut := l_alignement_debut + 8;
74  UNTIL l_quantite <= 0;
75  END; (* affiche_memoire *)
76
77  PROCEDURE affiche_donnees_turbo;
78
79  PROCEDURE affiche(p_titre: string80; p_decalage, p_taille: INTEGER);
80  VAR l_indice: INTEGER;
81  BEGIN
82    WRITE(sortie, '$'); affiche_entier_hexadecimal(dseg);
83    WRITE(sortie, ':$');
84    affiche_entier_hexadecimal(p_decalage); WRITE(sortie, ' ');
85
86    WRITE(sortie, p_titre);
87    affiche_octet_hexadecimal(p_taille);
88
89    WRITE(sortie, '$');
90    IF p_taille > 20
91      THEN p_taille := 20;
92    FOR l_indice := 0 TO p_taille - 1 DO
93      affiche_octet_hexadecimal(mem[dseg: p_decalage + l_indice]);
94
95    WRITELN(sortie);
96    END; (* affiche *)
97
98  BEGIN (* affiche_donnees_turbo *)
99    WRITELN('globales turbo');
100
101    affiche('v_attr_norm', $0000, 1);
102    affiche('v_attr_low', $0001, 1);
103    affiche('k_attr_reverse', $0003, 1);
104    affiche('v_fenetre_x_bas', $0004, 1);
105    affiche('v_fenetre_y_bas', $0005, 1);
106    affiche('v_mode_ecran', $0006, 1);
107    affiche('v_couleur', $0007, 1);
108    affiche('v_attribut', $0008, 1);
109    affiche('v_mode_graphiqu', $0009, 1);
110    affiche('v_fen_graphxdeb', $000A, 2);
111    affiche('v_fen_graphxfin', $000C, 2);
112    affiche('v_fen_graphydeb', $000E, 2);
113    affiche('v_fen_graphyfin', $0010, 2);
114
115    affiche('v_delai_1', $0012, 2);
116

```



```

117 affiche('v_draw_1', $0014, 2);
118 affiche('v_draw_4', $0016, 2);
119 affiche('v_draw_3', $0018, 2);
120 affiche('v_draw_6', $001A, 2);
121 affiche('v_draw_2', $001C, 2);
122 affiche('v_draw_5', $001E, 2);
123 affiche('v_palette', $0020, 1);
124
125 affiche('v_dela_2', $0021, 2);
126
127 affiche('v_ofs_deb_free', $0024, 2);
128 affiche('v_seg_deb_free', $0026, 2);
129 affiche('v_ofs_pt_deb_fr', $0028, 2);
130 affiche('v_seg_pt_deb_fr', $002A, 2);
131 affiche('v_ofs_pt1', $002C, 2);
132 affiche('v_seg_pt1', $002E, 2);
133 affiche('v_ofs_pointeur', $0030, 2);
134 affiche('v_seg_pointeur', $0032, 2);
135 affiche('v_ofs_pt3', $0034, 2);
136 affiche('v_seg_pt3', $0036, 2);
137
138 affiche('v_tamp_read', $0038, $40);
139 affiche('v_tampon_wrk', $00B8, $40);
140
141 affiche('v_adr_test_con', $0138, 2);
142 affiche('v_adr_lis_con', $013A, 2);
143 affiche('v_adr_ecris_con', $013C, 2);
144 affiche('v_adr_ecris_lst', $013E, 2);
145 affiche('v_adr_ecris_aux', $0140, 2);
146 affiche('v_adr_lis_aux', $0142, 2);
147 affiche('v_adr_ecris_usr', $0144, 2);
148 affiche('v_adr_lis_clav', $0146, 2);
149
150
151 affiche('v_handle_read', $015C, 2);
152 affiche('v_read_write', $015E, 2);
153 affiche('v_debut_glob_pg', $0160, 2);
154 affiche('v_sv_input', $0162, 2);
155 affiche('v_handle_write', $0168, 2);
156 affiche('v_colonne_max', $016A, 1);
157 affiche('v_ligne_max', $016B, 1);
158 affiche('v_debut_glob_2', $016C, 2);
159 affiche('v_sv_output', $016E, 2);
160
161 affiche('v_test_device', $0174, 2);
162 affiche('v_turbo_cs', $0176, 2);
163 affiche('v_turbo_ds', $0178, 2);
164
165 affiche('pt_table_handle', $017A, 2);
166 affiche('v_nb_fichiers', $017C, 2);
167 affiche('v_adr_proc_err', $017E, 2);
168 affiche('v_num_err_exec', $0180, 1);
169 affiche('v_read', $0181, 1);
170 affiche('v_ind_tamp_read', $0182, 2);
171 affiche('v_taille_tmp_rd', $0184, 2);
172 affiche('v_find_error_pc', $0188, 2);
173 affiche('v_reg_186', $0186, 2);
174

```

```

175 affiche('v_ofs_tas', $018A, 2);
176 affiche('v_seg_tas', $018C, 2);
177 affiche('v_sv_sp_2', $018E, 2);
178 affiche('v_sv_ofs_int_0', $0190, 2);
179 affiche('v_sv_seg_int_0', $0192, 2);
180 affiche('v_car_keypressd', $0194, 2);
181 affiche('v_clavier', $0196, 2);
182
183 affiche('v_tampon_ovl', $0198, $40);
184
185 affiche('v_taille_string', $01E8, 1);
186 affiche('v_val_ent_reel', $01E9, 1);
187 affiche('v_ofs_string', $01EA, 2);
188 affiche('v_seg_string', $01EC, 2);
189 affiche('v_ofs_ent_reel2', $01EE, 2);
190 affiche('v_seg_ent_reel2', $01F0, 2);
191 affiche('v_ofs_resu_val', $01F2, 2);
192 affiche('v_seg_resu_val', $01F4, 2);
193 affiche('v_tamp_reel1', $01F6, 6);
194 affiche('v_ofs_random', $01FE, 2);
195 affiche('v_erreur_i_o', $01FC, 2);
196 affiche('v_seg_random', $0200, 2);
197
198 affiche('v_taille_ch_des', $0202, 2);
199 affiche('v_ind_ins_del', $0204, 2);
200 affiche('v_qt_delete', $0206, 2);
201 affiche('v_ofs_chai_dest', $0208, 2);
202 affiche('v_seg_chai_dest', $020A, 2);
203 affiche('v_ofs_chai_dept', $020C, 2);
204 affiche('v_seg_chai_dept', $020E, 2);
205
206 affiche('pt_handle', $0234, 2);
207 affiche('v_sv_ds_es_fich', $0236, 2);
208 affiche('v_reset_write_a', $023A, 1);
209 affiche('v_err_rd_wr_jof', $023B, 1);
210 affiche('v_tamp_lis_reel', $023C, 6);
211
212 affiche('k_table_handle', $0242, $10);
213 END; (* affiche_donnees_turbo *)
214
215 PROCEDURE status;
216 VAR l_adresse: INTEGER;
217
218 PROCEDURE affiche_pointeur_hexadecimal(p_segment, p_alignement:
INTEGER);
BEGIN
219 affiche_entier_hexadecimal(p_segment);
220 WRITE(sortie, ':');
221 affiche_entier_hexadecimal(p_alignement);
222 END; (* affiche_pointeur_hexadecimal *)
223
224 BEGIN (* status *)
225 WRITE(sortie, 'cs: $');
226 affiche_entier_hexadecimal(cseg);
227

```



```

228 WRITE(sortie, ':$0100,pg:$');
229 l_adresse:= memw[cseg:$0101];
230 affiche_entier_hexadecimal(l_adresse+ $0103);
231 WRITE(sortie, ', ds:$');
232 affiche_entier_hexadecimal(dseg);
233 WRITE(sortie, ', heapptr:$');
234 affiche_pointeur_hexadecimal(memw[SEG(heapptr):OFS(heapptr)+ 2],
235                               memw[SEG(heapptr):OFS(heapptr)]);
236 WRITE(sortie, ', ss:$');
237 affiche_entier_hexadecimal(ssseg);
238 WRITELN(sortie);
239 END; (* status *)
240
241 overlay PROCEDURE procedure_overlay;
242 BEGIN
243 END; (* PROCEDURE overlay *)
244
245 PROCEDURE initialise;
246 BEGIN
247   nom_sortie:= 'CON.';
248   ASSIGN(sortie, nom_sortie);
249   REWRITE(sortie);
250
251   window(2, 2, 78, 23);
252   graphwindow($11, $22, $33, $44);
253
254   NEW(pointeur1);
255   NEW(pointeur2); NEW(pointeur1); NEW(pointeur1);
256   DISPOSE(pointeur2);
257
258   ASSIGN(fichier, 'STOCK.DTA');
259   REWRITE(fichier);
260
261   WRITE('reel <16.25> ? '); READLN(reel);
262
263   procedure_overlay;
264
265   VAL('1.75', reel, resultat_val);
266   STR(48.50, chaine);
267
268   chaine1:= 'ABC'; chaine2:= 'ABCD';
269   INSERT(chaine1, chaine2, $11);
270   DELETE(chaine3, $22, $33);
271 END; (* initialise *)
272
273 BEGIN (* principal *)
274   initialise;
275   REPEAT
276     status;
277     WRITE('m(emoire, g(lobales, q(uite ? ');
278     READ(KBD, choix); WRITELN(sortie, choix);
279     CASE choix OF
280       'g': affiche_donnees_turbo;
281       'm': affiche_memoire(dseg, 0, $262);

```

```

282 's': BEGIN
283   WRITE('nom sortie <a:essai.txt> ? ');
284   READLN(nom_sortie);
285   CLOSE(sortie);
286   ASSIGN(sortie, nom_sortie);
287   REWRITE(sortie);
288   END;
289 END;
290 UNTIL choix= 'q';
291 CLOSE(sortie);
292 END. (* principal *)

```

Fig. 7.1 Le programme *affiche_globales*.

7.2.3 L'AFFICHAGE DES GLOBALES

Voici le contenu brut de la zone des globales:

```

cs: $1392:$0100, pg: $2D4D, ds: $17F3, heapptr: $1844:0000, ss: $6000
m
$17F3:$0000 $0F $07 $07 $70 $01 $01 $07 $00 :...p...
$17F3:$0008 $0F $FF $11 $00 $33 $00 $22 $00 :...3..
$17F3:$0010 $44 $00 $43 $00 $00 $00 $00 $00 :D.C....
$17F3:$0018 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0020 $00 $43 $00 $00 $08 $00 $42 $18 :.C...B.
$17F3:$0028 $63 $02 $F3 $17 $08 $00 $00 $00 :c.....
$17F3:$0030 $24 $00 $F3 $17 $08 $00 $00 $00 :$......
$17F3:$0038 $62 $3A $72 $65 $73 $75 $2E $70 :b:resu.p
$17F3:$0040 $61 $73 $1A $00 $00 $00 $00 $00 :as.....
$17F3:$0048 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0050 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0058 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0060 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0068 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0070 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0078 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0080 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0088 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0090 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$0098 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00A0 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00A8 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00B0 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00B8 $62 $3A $72 $65 $73 $75 $2E $70 :b:resu.p
$17F3:$00C0 $61 $73 $00 $30 $45 $2B $30 $31 :as.OE+01
$17F3:$00C8 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00D0 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00D8 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00E0 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00E8 $00 $00 $00 $00 $00 $00 $00 $00 :.....
$17F3:$00F0 $00 $00 $00 $00 $00 $00 $00 $00 :.....

```



```

$17F3:$00F8 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0100 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0108 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0110 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0118 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0120 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0128 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0130 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0138 $0D $08 $26 $08 $59 $08 $F6 $08 ...&Y...
$17F3:$0140 $FE $08 $06 $09 $59 $08 $26 $08 ...Y.&...
$17F3:$0148 $FF $FF $C1 $00 $FF $FF $82 $6D .....m
$17F3:$0150 $FF $FF $43 $00 $FF $FF $C4 $00 ...C....
$17F3:$0158 $FF $FF $C5 $00 $FF $FF $E1 $1A .....
$17F3:$0160 $62 $02 $00 $00 $00 $00 $00 $00 ...b....
$17F3:$0168 $FF $FF $C1 $00 $62 $02 $00 $00 .....b...
$17F3:$0170 $00 $00 $00 $00 $07 $00 $7B $06 .....{...
$17F3:$0178 $3F $10 $42 $02 $10 $00 $81 $10 ...?B....
$17F3:$0180 $00 $7E $43 $00 $43 $00 $FE $2E ...-C.C./...
$17F3:$0188 $00 $00 $00 $00 $44 $18 $FE $FF .....D...
$17F3:$0190 $43 $31 $DB $00 $00 $00 $01 $00 ...Cl....
$17F3:$0198 $00 $6C $6F $62 $61 $6C $73 $2E ...lobals.
$17F3:$01A0 $30 $30 $30 $00 $00 $00 $00 $00 ...000....
$17F3:$01A8 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$01B0 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$01B8 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$01C0 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$01C8 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$01D0 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$01D8 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$01E0 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$01E8 $50 $01 $B7 $02 $F3 $17 $FB $03 ...P....
$17F3:$01F0 $F3 $17 $01 $04 $F3 $17 $86 $00 .....
$17F3:$01F8 $00 $00 $00 $42 $00 $00 $00 $00 ...B....
$17F3:$0200 $00 $00 $50 $00 $22 $00 $33 $00 ...P...3.
$17F3:$0208 $AA $03 $F3 $17 $F4 $FF $00 $60 .....
$17F3:$0210 $82 $33 $33 $33 $33 $9B $84 $00 ...3333...
$17F3:$0218 $00 $00 $00 $A0 $00 $00 $00 $00 .....
$17F3:$0220 $00 $00 $00 $80 $00 $0A $FF $FE .....
$17F3:$0228 $34 $38 $35 $00 $00 $00 $00 $00 ...485....
$17F3:$0230 $00 $00 $00 $00 $03 $04 $F3 $17 .....
$17F3:$0238 $00 $00 $00 $01 $00 $85 $00 $00 .....
$17F3:$0240 $00 $02 $05 $00 $06 $00 $00 $00 .....
$17F3:$0248 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0250 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0258 $00 $00 $00 $00 $00 $00 $00 $00 .....
$17F3:$0260 $00 $00 $6D .....m

```

Fig. 7.2 La zones des globales

Et voici le contenu interprété:

cs: \$1392:\$0100, pg: \$2D4D, ds: \$17F3, heapptr: \$1844:0000, ss: \$6000

```

$17F3:$0000 v_attr_norm 01 $0F
$17F3:$0001 v_attr_low 01 $07
$17F3:$0003 k_attr_reverse 01 $70
$17F3:$0004 v_fenetre_x_bas 01 $01
$17F3:$0005 v_fenetre_y_bas 01 $01
$17F3:$0006 v_mode_ecran 01 $07
$17F3:$0007 v_couleur 01 $00
$17F3:$0008 v_attribut 01 $0F
$17F3:$0009 v_mode_graphiqu 01 $FF
$17F3:$000A v_fen_graphxdeb 02 $1100
$17F3:$000C v_fen_graphxfin 02 $3300
$17F3:$000E v_fen_graphydeb 02 $2200
$17F3:$0010 v_fen_graphyfin 02 $4400

$17F3:$0012 v_delai_1 02 $4300

$17F3:$0014 v_draw_1 02 $0000
$17F3:$0016 v_draw_4 02 $0000
$17F3:$0018 v_draw_3 02 $0000
$17F3:$001A v_draw_6 02 $0000
$17F3:$001C v_draw_2 02 $0000
$17F3:$001E v_draw_5 02 $0000
$17F3:$0020 v_palette 01 $00

$17F3:$0021 v_delai_2 02 $4300

$17F3:$0024 v_ofs_deb_free 02 $0800
$17F3:$0026 v_seg_deb_free 02 $4218
$17F3:$0028 v_ofs_pt_deb_fr 02 $6302
$17F3:$002A v_seg_pt_deb_fr 02 $F317
$17F3:$002C v_ofs_pt1 02 $0800
$17F3:$002E v_seg_pt1 02 $0000
$17F3:$0030 v_ofs_pointeur 02 $2400
$17F3:$0032 v_seg_pointeur 02 $F317
$17F3:$0034 v_ofs_pt3 02 $0800
$17F3:$0036 v_seg_pt3 02 $0000

$17F3:$0038 v_tamp_read 40 $623A726573752E7061731A0000...
$17F3:$00B8 v_tampon_wrk 40 $623A726573752E706173003000...

$17F3:$0138 v_adr_test_con 02 $0D08
$17F3:$013A v_adr_lis_con 02 $2608
$17F3:$013C v_adr_ecris_con 02 $5908
$17F3:$013E v_adr_ecris_lst 02 $F608
$17F3:$0140 v_adr_ecris_aux 02 $FE08
$17F3:$0142 v_adr_lis_aux 02 $0609
$17F3:$0144 v_adr_ecris_usr 02 $5908
$17F3:$0146 v_adr_lis_clav 02 $2608

```


TU_CLRSCR

- adresse: \$01EF
- utilise l'interruption \$10, pour purger l'écran, dimensionner la fenêtre, initialiser le mode

TU_DELLINE

- adresse: \$0223
- utilise l'interruption \$10 du BIOS pour dérouler l'écran en effaçant la ligne

TU_INSLINE

- adresse: \$024D
- utilise l'interruption \$10 du BIOS pour dérouler l'écran en créant une nouvelle ligne

TU_LOWVIDEO

- adresse: \$0258
- place dans l'attribut courant (*v_attribut*) la valeur de l'attribut de vidéo faible (*v_attr_low*)

TU_HIGHVIDEO

- adresse: \$0261
- place dans l'attribut courant (*v_attribut*) la valeur de l'attribut fort (*v_attr_norm*)

TU_CLREOL

- adresse: \$0271
- utilise l'interruption \$10 du BIOS

TU_CRTINIT

- adresse: \$028E
- utilise l'interruption \$10 du BIOS pour initialiser l'écran et la fenêtre

TU_TEXT_MODE

- adresse: \$0295

TU_CRTEXIT

- adresse: \$0308
- ne fait rien

TU_UPCASE

- adresse: \$0330
- entrée: caractère dans AL, \$00 dans AH
- sortie: caractère en majuscule dans AL
- saute à la routine MAJUSCULE (\$0A63) qui passe les lettres en majuscule

TU_WHEREX

- adresse: \$0333
- sortie: valeur de x dans AL, \$00 dans AH
- invoque l'interruption \$10 du BIOS, recalcule la position relative à la fenêtre, ajoute 1

TU_WHEREY

- adresse: \$0341
- sortie: valeur de y dans AL, \$00 dans AH
- invoque l'interruption \$10 du BIOS, recalcule la position relative à la fenêtre, ajoute 1

TU_WINDOW

- adresse: \$034F
- entrée: sur la pile: x1, y1, x2. AX = y2
- met à jour les paramètres de la fenêtre (*v_fenetre_x_bas*, *v_fenetre_y_bas*, *v_colonne_max*, *v_ligne_max*), et affiche le curseur si la fenêtre contient le coin gauche supérieur de l'écran

TU_TEXTCOLOR

- adresse: \$036C
- entrée: AX contient le code de la couleur
- met à jour l'attribut courant (*v_attribut*)

TU_TEXTBACKGRO

- adresse: \$0380
- entrée: AX contient le code de la couleur
- combine avec la couleur (*v_couleur*) et met à jour l'attribut courant (*v_attribut*)

TU_GRAPHCOLORMODE

- adresse: \$03BF
- initialise le mode graphique (*v_mode_graphique*), utilise l'interruption \$10 du BIOS pour définir la palette, initialise la fenêtre

TU_GRAPHMODE

- adresse: \$03C9
- initialise le mode graphique (*v_mode_graphique*), utilise l'interruption \$10 du BIOS pour définir la palette, initialise la fenêtre

TU_HIRES

- adresse: \$03D0
- initialise le mode graphique (*v_mode_graphique*), utilise l'interruption \$10 du BIOS pour définir la palette, initialise la fenêtre, initialise la couleur

TU_GRAPHBACKGROUND

- adresse: \$03E0
- entrée: AX = la couleur
- met à jour la palette (*v_palette*) et utilise l'interruption \$10 du BIOS

TU_PALETTE

- adresse: \$03FB
- entrée: AX = la palette
- met à jour la palette (*v_palette*) et utilise l'interruption \$10 du BIOS

TU_HIRESCOLOR

- adresse: \$0426
- entrée: AX= la couleur
- utilise l'interruption \$10 du BIOS

TU_GRAPHWINDOW

- adresse: \$042F
- entrée: Sur la pile: x1, y1, x2. AX= y2.
- met à jour les paramètres de la fenêtre graphique

TU_PLOT

- adresse: \$0463
- entrée: Sur la pile: x et y. AX= la couleur.
- vérifie que le point est dans la fenêtre et utilise l'interruption \$10 du BIOS

TU_DRAW

- adresse: \$048A
- entrée: Sur la pile: x1, y1, x2, y2. AX= la couleur.
- utilise TU_PLOT pour tracer le trait

TU_SOUND

- adresse: \$054D
- entrée: AX= la fréquence en Hertz
- émet vers les ports de sortie \$42, \$43 et \$61, et lit le port \$61

TU_NOSOUND

- adresse: \$0574
- écrit et lit le port \$61

7.3.3 LA GESTION DES POINTEURS

Voici les routines chargées de la gestion des pointeurs:

TU_GETMEM

- adresse: \$057B
- entrée: Sur la pile: le segment et le décalage du pointeur. AX= le nombre d'octets à allouer.

TU_NEW

- adresse: \$05A8
- entrée: Sur la pile: le segment du pointeur. DI= le décalage du pointeur, CX= la taille de la cellule pointée.

TU_FREEMEM

- adresse: \$0672
- entrée: Sur la pile: le segment et le décalage du pointeur. AX= le nombre d'octets à libérer.

TU_DISPOSE

- adresse: \$0677
- entrée: Sur la pile: le segment du pointeur. DI= le décalage du pointeur, CX= la taille de la cellule pointée.

TU_MEMAVAIL

- adresse: \$0795
- sortie: AX= le nombre de paragraphes libres

TU_MAXAVAIL

- adresse: \$07D2
- sortie: AX= le nombre de paragraphes du plus gros espace libre d'un seul tenant

TU_MARK

- adresse: \$07D8
- entrée: Sur la pile: le segment du pointeur qui stockera la position actuelle du sommet du tas. DI= le décalage de ce pointeur.

TU_RELEASE

- adresse: \$07EA
- entrée: Sur la pile: le segment du pointeur qui contient la position actuelle du sommet du tas. DI= le décalage de ce pointeur.

7.3.4 ROUTINES DIVERSES

Voici des routines ayant des fonctions très différentes:

TU_PARAGRAPHS

- adresse: \$0A99
- chargé d'initialiser le contexte avant l'exécution d'un programme PASCAL. Se charge:

- de vérifier que la version DOS est correcte
- de vérifier que la mémoire disponible est suffisante, en utilisant les paramètres situés de \$2D56 à \$2D5D (taille du code, des globales et du tas)
- d'initialiser le tas et les listes de cellules libres
- de demander au DOS d'allouer la mémoire
- de sauvegarder l'environnement de TURBO.COM en cas de compilation mémoire
- d'allouer la place pour les fichiers si vous avez demandé plus de \$10 fichiers
- de mettre en place les routines de gestion de l'interruption 0 et 3
- d'ouvrir les fichiers d'entrée et de sortie si vous avez utilisé les options G et P
- d'initialiser l'écran

TU_HALT

- adresse: \$0C40
- invoqué à la fin de chaque programme. Se charge:
 - de fermer les fichiers
 - de retourner dans TURBO.COM ou de terminer le programme

TU_TAILLE_CODE

- adresse: \$0C7D
- se charge de créer dans le code des emplacements pour accueillir le code des procédures OVERLAY

TU_APPEL_OVERLAY

- adresse: \$0CBF
- chargé de charger une procédure OVERLAY

TU_OVRPATH

- adresse: \$0D41
- entrée: sur la pile, le nom du chemin d'accès à la procédure OVERLAY
- dépile le nom et le recopie dans le tampon des noms de fichiers OVERLAY

TU_KEYPRESSED

- adresse: \$0D55
- sortie: AL= le résultat du test
- appelle la routine de lecture du périphérique courant (v_adr_test_con)

TU_GOTOXY

- adresse: \$0D5B
- entrée: Sur la pile: x. AX= y.
- décrémente x et y et appelle l'interruption \$10 du BIOS

TU_PARAMSTR

- adresse: \$0D69
- entrée: Sur la pile, le segment et le décalage de la chaîne où sera placé le paramètre. AX= le numéro du paramètre à récupérer.
- sortie: le paramètre sous forme de chaîne sur la pile (affecté à la chaîne par TU_POP_STRING)

TU_PARAMCOUNT

- adresse: \$0D86
- sortie: AL= le nombre de paramètres de la ligne de commande

TU_STR_ENTIER

- adresse: \$0DC3
- entrée: Sur la pile: la valeur entière, le format, le segment de la chaîne. DI= le décalage de la chaîne. CL= la taille maximale de la chaîne.

TU_STR_REEL

- adresse: \$0DDC
- entrée: Sur la pile: les 6 octets de la valeur réelle, le nombre de décimales, le nombre de caractères totaux, le segment de la chaîne. DI= le décalage de la chaîne, CL= la taille maximale de la chaîne.

TU_VAL_ENTIER

- adresse: \$0E38
- entrée: Sur la pile: la chaîne, le segment et le décalage et de l'entier, le segment du résultat. DI= le décalage du résultat.

TU_VAL_REEL

- adresse: \$0E3C
- entrée: Sur la pile: le segment et le décalage du réel, la chaîne, le segment du résultat. DI= le décalage du résultat.

TU_RANDOMIZE

- adresse: \$0EC5
- utilise l'interruption \$21 du DOS, AH= \$2C (heure) pour initialiser la valeur aléatoire (v_seg_random et v_ofs_random)

TU_COPIE_STRUCTURE

- adresse: \$0ED3
- entrée: Sur la pile: segment de la variable de destinataire, décalage de la variable destinataire, segment de la variable départ. DI= décalage de la variable départ, CX= taille de la table.
- utilisé pour les affectations globales de tableaux et d'enregistrements

TU_POUSSE_STRUCTURE

- adresse: \$0EE2
- entrée: Sur la pile: le segment de la structure. DI= décalage de la structure.

TU_FILLCHAR

- adresse: \$0EF5
- entrée: Sur la pile: segment de départ et décalage de départ, quantité d'octets à modifier (2 octets). AX= octet à placer dans la zone.

TU_MOVE

- adresse: \$0EFE
- entrée: Sur la pile: segment départ, décalage départ, segment destination décalage destination. AX= nombre d'octets à déplacer.

TU_INTR_DEB

- adresse: \$0F18
- entrée: Sur la pile: segment de l'enregistrement contenant les registres. DI= décalage de l'enregistrement contenant les registres.
- cette routine place dans les registres du 8088 les valeurs que vous avez placées dans l'enregistrement PASCAL

TU_INTR_FIN

- adresse: \$0F40
- cette routine place dans l'enregistrement PASCAL les valeurs qui se trouvent dans les registres après l'interruption

TU_INDEX_CHECK

- adresse: \$0F6A
- entrée: AX= rang de la cellule (indice moins indice minimal), CX= nombre de cellules (indice maximal plus un moins indice minimal)
- invoqué au moment de chaque calcul d'indice de tableau comportant une variable, lorsque vous travaillez en mode R+

TU_RANGE_CHECK

- adresse: \$0F74
- entrée: AX= valeur à affecter, CX= borne minimale de l'intervalle, DX= borne maximale de l'intervalle
- invoqué avant chaque affectation d'une valeur à une variable de type intervalle, lorsque vous travaillez en mode R+

TU_STACK_CHECK

- adresse: \$0F82
- entrée: CX= nombre d'octets (environnement+ locales+ paramètres) utilisés par la procédure
- invoqué avant chaque appel de procédure utilisateur pour vérifier que la pile peut accommoder l'appel. Peut être supprimé par l'option K-

TU_IORESULT

- adresse: \$0FCC
- sortie: AL= code d'erreur de l'entrée / sortie précédente, et mise à zéro de la variable système contenant ce code

TU_IO_CHECK

- adresse: \$0FD3
- vérification que le code d'erreur est nul. Dans le cas contraire, une erreur d'exécution est provoquée. Peut ne pas être généré lorsque vous travaillez en mode I-

TU_ABS_ENTIER

- adresse: \$1084
- entrée: AX= la valeur de l'entier
- sortie: AX= la valeur absolue de l'entier

TU_RANDOM_ENTIER

- adresse: \$108B
- entrée: AX= valeur maximale entière de la valeur aléatoire
- sortie: AX= une valeur aléatoire entre 0 et la valeur maximale demandée

7.3.5 LES CHAINES

Voici les routines chargées de la gestion des chaînes:

TU_POUSSE_STRING

- adresse: \$1182
- entrée: Sur la pile: segment de la chaîne. DI= décalage de la chaîne.
- pousse la chaîne sur la pile (l'octet de longueur au sommet)

TU_POUSSE_K_ST

- adresse: \$119B
- entrée: La chaîne constante se trouve dans le code juste après l'appel (son segment est toujours CS, et son décalage est sur la pile à cause du CALL. Sa taille est le premier octet après l'appel).

TU_POP_STRING

- adresse: \$11B1
- entrée: Sur la pile: le segment et le décalage de la chaîne destination, la chaîne à dépiler (octet de longueur au sommet). CL= taille maximale de la chaîne destination.
- dépile la chaîne dans la variable destination, en tronquant la valeur si elle dépasse la taille de la variable destination

TU_POUSSE_TAB_CAR

- adresse: \$11D9
- entrée: Sur la pile: segment du tableau de caractères. DI= décalage du tableau de caractères, CL= taille du tableau.
- pousse le tableau sous forme de chaîne pour permettre l'affichage

TU_PARAM_F_STRING

- adresse: \$11F3
- entrée: CL= taille de la chaîne (sans l'octet de longueur)
- utilisé lorsqu'une chaîne est un paramètre valeur d'une fonction

TU_EGAL_STRING

- adresse: \$1230
- entrée: Sur la pile, les deux chaînes à comparer
- sortie: AL= le résultat de la comparaison

TU_DIFF_STRING

- adresse: \$123C
- entrée: Sur la pile, les deux chaînes à comparer
- sortie: AL= le résultat de la comparaison

TU_SUP_EGAL_ST

- adresse: \$1248
- entrée: Sur la pile, les deux chaînes à comparer
- sortie: AL= le résultat de la comparaison

TU_INF_EQUAL ST

- adresse: \$1254
- entrée: Sur la pile, les deux chaînes à comparer
- sortie: AL= le résultat de la comparaison

TU_SUP_STRING

- adresse: \$1260
- entrée: Sur la pile, les deux chaînes à comparer
- sortie: AL= le résultat de la comparaison

TU_INF_STRING

- adresse: \$126C
- entrée: Sur la pile, les deux chaînes à comparer
- sortie: AL= le résultat de la comparaison

TU_CONCAT

- adresse: \$12B3
- entrée: Sur la pile, les deux chaînes à concaténer
- sortie: sur la pile: la chaîne résultant de la concaténation

TU_COPY

- adresse: \$12F8
- entrée: Sur la pile: le segment et le décalage de la chaîne destination, l'indice de départ de la copie, la chaîne de départ. AX= nombre de caractères à copier.

TU_LENGTH

- adresse: \$1341
- entrée: sur la pile, la chaîne dont la longueur doit être mesurée
- sortie: AL= la longueur de la chaîne

TU_POS

- adresse: \$134E
- entrée: sur la pile, la chaîne de départ, la chaîne recherchée
- sortie: AL= l'indice de la position de la chaîne recherchée dans la chaîne de départ

TU_INSERT

- adresse: \$1392
- entrée: Sur la pile: la chaîne de départ, le segment et le décalage de la chaîne à insérer. CL= taille maximale de la chaîne destinataire, AX= indice où sera effectué l'insertion.

TU_DELETE

- adresse: \$13EA
- entrée: Sur la pile: segment de la chaîne, décalage de la chaîne et indice de départ de l'effacement, AX= quantité d'octets à effacer.

TU_MOV_CAR STR

- adresse: \$1433
- entrée: sur la pile, une chaîne (de taille 1)
- sortie: AL= le caractère de la chaîne
- utilisé pour affecter une chaîne de taille 1 à un caractère

TU_MOV_STR CAR

- adresse: \$1446
- entrée: sur la pile, une chaîne
- sortie: AL= le caractère
- utilisé dans les expressions du type *caractère*= *caractère*1+ *chaîne*;

TU_RESU_FCT ST

- adresse: \$145B
- entrée: DX= taille utilisée par les paramètres, CL= taille du résultat de la fonction
- utilisé comme conclusion des fonctions dont le résultat est de type chaîne

7.3.6 LES ENSEMBLES

Voici les routines chargées de la gestion des ensembles:

TU_POUSSE SET

- adresse: \$148F
- entrée: Sur la pile: le segment de l'ensemble, DI= décalage de l'ensemble, CH= octet de départ (soit *borne inférieure* DIV 8), CL= nombre d'octets utilisés par l'ensemble (soit *borne supérieure* DIV 8- *borne inférieure* DIV 8+ 1).
- utilisé pour les opérations arithmétiques sur les ensembles

TU_INITIE SET

- adresse: \$14BF
- pousse 32 (\$20) octets égaux à zéro sur la pile (correspondant à l'ensemble vide de taille maximal)

TU_AJ_EL SET

- adresse: \$14D1
- entrée: Sur la pile: l'ensemble (32 octets). AX= l'élément à ajouter.

TU_AJ_INTERV SET

- adresse: \$14D8
- entrée: Sur la pile: l'ensemble (32 octets), la borne inférieure de l'intervalle. AX= borne supérieure de l'intervalle à ajouter.

TU_AFFECTE SET

- adresse: \$14F7
- entrée: Sur la pile: le segment et le décalage de l'ensemble, les 32 octets à affecter. CH= octet de départ (soit *borne inférieure* DIV 8), CL= nombre d'octets utilisés par l'ensemble (soit *borne supérieure* DIV 8- *borne inférieure* DIV 8+ 1).

TU_ALIGNE SET

- adresse: \$1515
- entrée: Sur la pile: le segment et le décalage de l'ensemble, les 32 octets de l'ensemble. CH= octet de départ (soit *borne inférieure* DIV 8), CL= nombre d'octets utilisés par l'ensemble (soit *borne supérieure* DIV 8- *borne inférieure* DIV 8+ 1).
- utilisé avant l'appel d'une procédure utilisant un paramètre valeur de type ensemble

TU_EQUAL SET

- adresse: \$153B
- entrée: sur la pile deux ensembles (de 32 octets)
- sortie: AL= le résultat de la comparaison

TU_DIFF SET

- adresse: \$1540
- entrée: sur la pile deux ensembles (de 32 octets)
- sortie: AL= le résultat de la comparaison

TU_SUP_EQUAL SE

- adresse: \$1553
- entrée: sur la pile deux ensembles (de 32 octets)
- sortie: AL= le résultat de la comparaison

TU_INF_EQUAL SE

- adresse: \$1557
- entrée: sur la pile deux ensembles (de 32 octets)
- sortie: AL= le résultat de la comparaison

TU_ADD SET

- adresse: \$1578
- entrée: sur la pile deux ensembles (de 32 octets)
- sortie: sur la pile le résultat de l'union des deux ensembles

TU_SOUSTR SET

- adresse: \$1586
- entrée: sur la pile deux ensembles (de 32 octets)
- sortie: sur la pile le résultat de la différence des deux ensembles

TU_INTERSECT S

- adresse: \$1596
- entrée: sur la pile deux ensembles (de 32 octets)
- sortie: sur la pile le résultat de l'intersection des deux ensembles

TU_IN SET

- adresse: \$15A4
- entrée: sur la pile: la valeur à tester (2 octets), l'ensemble (32 octets)
- sortie: AL= résultat du test d'appartenance

et deux routines de pointeurs perdues au milieu des routines d'ensembles:

TU_EQUAL PTEUR

- adresse: \$15EF
- entrée: AX= décalage de *pointeur2*, DX= segment de *pointeur2*, BX= décalage de *pointeur1*, CX= segment de *pointeur1*
- sortie: AL= résultat du test

TU_DIFF_PTEUR

- adresse: \$15FE
- entrée: AX= décalage de *pointeur2*, DX= segment de *pointeur2*, BX= décalage de *pointeur1*, CX= segment de *pointeur1*
- sortie: AL= résultat du test

7.3.7 LES REELS

Voici les routines chargées de la gestion des réels:

TU_POUSSE REEL

- adresse: \$1936
- entrée: Sur la pile: le segment du réel. DI= décalage du réel.
- sortie: les 6 octets du réel sur la pile (l'octet de l'exposant au sommet)

TU_POUSSE K REELLE

- adresse: \$1945
- entrée: DI= décalage du réel. Le segment est toujours CS
- sortie: les 6 octets de la constante du code sont sur la pile (l'octet de l'exposant au sommet)

TU_POP REEL

- adresse: \$1956
- entrée: sur la pile, le segment du réel destinataire, le décalage du réel destinataire, les 6 octets du réel (l'octet de l'exposant au sommet).
- sortie: le réel destinataire contient les 6 octets auparavant sur la pile

TU_ADD REEL

- adresse: \$1969
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: sur la pile, les 6 octets du résultat de l'addition

TU_SUB_REEL

- adresse: \$1984
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: sur la pile, les 6 octets du résultat de la soustraction

TU_MULTIPLIE_REEL

- adresse: \$1993
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: sur la pile, les 6 octets du résultat de la multiplication

TU_DIVISE_REEL

- adresse: \$19A2
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: sur la pile, les 6 octets du résultat de la division

TU_OPPOSE_REEL

- adresse: \$19BA
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, les 6 octets de l'opposé (-réel)

TU_ABS_REEL

- adresse: \$19C9
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, les 6 octets du réel positif (ABS(réel))

TU_EQUAL_REEL

- adresse: \$19D1
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: AL= résultat de la comparaison

TU_DIFF_REEL

- adresse: \$19EB
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: AL= résultat de la comparaison

TU_SUP_EQUAL_REEL

- adresse: \$1A05
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: AL= résultat de la comparaison

TU_INF_EQUAL_REEL

- adresse: \$1A1F
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: AL= résultat de la comparaison

TU_SUP_REEL

- adresse: \$1A39
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: AL= résultat de la comparaison

TU_INF_REEL

- adresse: \$1A53
- entrée: sur la pile les 2 réels (12 octets en tout)
- sortie: AL= résultat de la comparaison

TU_SQR

- adresse: \$1A6D
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction SQR (6 octets)

TU_INT

- adresse: \$1A7D
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction INT (6 octets)

TU_FRAC

- adresse: \$1A8A
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction FRAC (6 octets)

TU_RANDOM_REEL

- adresse: \$1A97
- sortie: sur la pile, le résultat de la fonction RANDOM (6 octets)

TU_ROUND

- adresse: \$1AB9
- entrée: sur la pile le réel (6 octets en tout)
- sortie: AX= la valeur entière correspondant au réel tronqué

TU_TRUNC

- adresse: \$1ABD
- entrée: sur la pile le réel (6 octets en tout)
- sortie: AX= la valeur entière correspondant au réel arrondi

TU_FLOTTE_ENTIER

- adresse: \$1AF2
- entrée: sur la pile, les 2 octets d'un entier
- sortie: sur la pile, les 6 octets correspondant à un réel ayant la même valeur que l'entier

TU_FLOTTE_PRECEDENT

- adresse: \$1AFB
- sur la pile: les 2 octets d'un entier, les 6 octets d'un réel
- sortie: sur la pile: les 6 octets d'un réel ayant la même valeur que l'entier, les 6 octets du premier réel (remonté de 4 octets). Utilisé dans les opérations du type *réel:= entier+ réel2*;

TU_SQRT

- adresse: \$1B10
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction **SQRT** (6 octets)

TU_COS

- adresse: \$1B6E
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction **COS** (6 octets)

TU_SIN

- adresse: \$1B83
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction **SIN** (6 octets)

TU_LN

- adresse: \$1C16
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction **LN** (6 octets)

TU_EXP

- adresse: \$1CC6
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction **EXP** (6 octets)

TU_ARCTAN

- adresse: \$1D6F
- entrée: sur la pile le réel (6 octets en tout)
- sortie: sur la pile, le résultat de la fonction **ARCTAN** (6 octets)

7.3.8 LES FICHIERS

Voici les routines chargées de la gestion des fichiers:

TU_ASSIGN_TEXT

- adresse: \$229D
- entrée: sur la pile, le segment de la variable fichier, le décalage de la variable fichier, le nom du fichier (octet de longueur au sommet).

TU_RESET_TEXT

- adresse: \$22F2
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier, CX= taille du tampon (\$80 par défaut).

TU_REWRITE_TEXT

- adresse: \$22F6
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier, CX= taille du tampon (\$80 par défaut).

TU_APPEND_TEXT

- adresse: \$22FA
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier, CX= taille du tampon (\$80 par défaut).

TU_TRUNCATE_TEXT

- adresse: \$23D7
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_FLUSH_TEXT

- adresse: \$2409
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_CLOSE_TEXT

- adresse: \$241D
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_READ_INPUT

- adresse: \$24F3
- utilisé lorsque l'option G est en oeuvre (Get)

TU_DEB_READ_TEXT

- adresse: \$2505
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- utilisé pour la lecture d'un fichier **TEXT** ou des fichiers prédéfinis (**CON**, **TRM**, **KBD**, **LST**, **USR**, **AUX**)

TU_DEB_WRITE_DEFAULT

- adresse: \$2522
- utilisé pour l'affichage du fichier de sortie par défaut

TU_DEB_WRITE_TEXT

- adresse: \$2534
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- utilisé pour l'affichage d'un fichier **TEXT** ou des fichiers prédéfinis (**CON**, **TRM**, **KBD**, **LST**, **USR**, **AUX**)

TU_READLN

- adresse: \$2551
- utilisé pour la lecture du return

TU_READ_DEFAULT

- adresse: \$2555
- utilisé pour la lecture du fichier d'entrée par défaut

TU_READ_CAR

- adresse: \$272C
- entrée: Sur la pile, le segment du caractère. DI= le décalage du caractère.

TU_READ_OCTET

- adresse: \$273D
- entrée: Sur la pile, le segment de l'octet. DI= le décalage de l'octet

TU_READ_ENTIER

- adresse: \$2740
- entrée: Sur la pile, le segment de l'entier. DI= le décalage de l'entier.

TU_READ_REEL

- adresse: \$275F
- entrée: Sur la pile, le segment du réel. DI= le décalage du réel.

TU_READ_CHAINE

- adresse: \$277D
- entrée: Sur la pile, le segment de la chaîne. DI= le décalage de la chaîne, CL= la taille maximale de la chaîne.

TU_READ_ARRAY

- adresse: \$27AA
- entrée: Sur la pile, le segment du tableau. DI= le décalage du tableau de caractères, CL= la taille du tableau.

TU_READLN TEXT

- adresse: \$27D1
- utilisé pour lire le RETURN de fichiers de type TEXT

TU_WRITE_CAR

- adresse: \$2867
- entrée: Sur la pile, le caractère (2 octets: le caractère et \$00). AX= le format d'affichage (0 par défaut).

TU_WRITE_ENTIER

- adresse: \$2883
- entrée: Sur la pile, l'entier (2 octets). AX= le format d'affichage (0 par défaut).

TU_WRITE_REEL

- adresse: \$28B8
- entrée: Sur la pile: les 6 octets du réel (l'exposant au sommet), le nombre total d'octets à utiliser pour l'affichage (par défaut 18, soit \$12). AX= le nombre de décimales à afficher (par défaut \$FFFF).

TU_WRITE_BOOLEAN

- adresse: \$28D0
- entrée: Sur la pile, le booléen (2 octets: le booléen et \$00). AX= le format d'affichage (0 par défaut).

TU_WRITE_CHAINE

- adresse: \$28F0
- entrée: AX= le format d'affichage (0 par défaut). Sur la pile, la chaîne (l'octet de taille au sommet)

TU_WRITE_K_STRING

- adresse: \$2927
- entrée: la chaîne littérale, qui se trouve juste après cet appel

TU_WRITELN

- adresse: \$293F
- utilisé pour exécuter WRITELN

TU_SEEKEOLN

- adresse: \$2949
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: AL= résultat du test

TU_EOLN

- adresse: \$294E
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: AL= résultat du test

TU_SEEKEOF

- adresse: \$2953
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: AL= résultat du test

TU_EOF TEXT

- adresse: \$2958
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: AL= résultat du test

TU_ASSIGN FILE

- adresse: \$298E
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier, le nom du fichier (octet de longueur au sommet).

TU_RESET

- adresse: \$2993
- entrée: Sur la pile, le segment de la variable fichier. DI= décalage de la variable fichier, CX= taille de la fiche.

TU_REWRITE

- adresse: \$2997
- entrée: Sur la pile, le segment de la variable fichier. DI= décalage de la variable fichier, CX= taille de la fiche.

TU_TRUNCATE

- adresse: \$29C2
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_FLUSH

- adresse: \$29D5
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_CLOSE

- adresse: \$29D8
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_DEB RW FICHIER

- adresse: \$29EA
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_READ FICHIER

- adresse: \$2A07
- entrée: Sur la pile: le segment de la variable à lire. DI= décalage de la variable à lire.

TU_WRITE FICHI

- adresse: \$2A0F
- entrée: Sur la pile: le segment de la variable à écrire. DI= décalage de la variable à écrire.

TU_SEEK

- adresse: \$2A61
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier. AX= le numéro de la fiche.

TU_LONGSEEK

- adresse: \$2A95
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier, les 6 octets du réel désignant le numéro de la fiche.

TU_EOF

- adresse: \$2A9A
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: AL= résultat du test

TU_FILEPOS

- adresse: \$2AB1
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: AX= numéro de la fiche courante

TU_LONGFILEPOS

- adresse: \$2AC8
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: sur la pile: les 6 octets représentant le numéro de la fiche courante

TU_FILESIZE

- adresse: \$2AD1
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: AX= nombre de fiches contenues dans le fichier

TU_LONGFILESIZ

- adresse: \$2B0D
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.
- sortie: sur la pile: les 6 octets représentant le nombre de fiches du fichier.

TU_RESET FILE

- adresse: \$2B46
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier. AX= la taille du tampon (par défaut \$80).

TU_REWRITE FILE

- adresse: \$2B4D
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier. AX= la taille du tampon (par défaut \$80).

TU_BLOCKREAD

- adresse: \$2B54
- entrée: Sur la pile: le segment de la variable fichier, le décalage de la variable fichier, le segment du tampon, le décalage du tampon. AX= le nombre de blocs à lire.

TU_BLOCKWRITE

- adresse: \$2B5C
- entrée: Sur la pile: le segment de la variable fichier, le décalage de la variable fichier, le segment du tampon, le décalage du tampon. AX= le nombre de blocs à écrire.

TU_BLOCKREAD_RESU

- adresse: \$2B84
- entrée: Sur la pile: le segment de la variable fichier, le décalage de la variable fichier, le segment du tampon, le décalage du tampon, le nombre de blocs à lire, le segment de la variable qui contiendra le nombre de blocs lus. DI= le décalage de la variable qui contiendra le nombre de blocs lus.

TU_BLOCKWRITE_RESU

- adresse: \$2B8C
- entrée: Sur la pile: le segment de la variable fichier, le décalage de la variable fichier, le segment du tampon, le décalage du tampon, le nombre de blocs à écrire, le segment de la variable qui contiendra le nombre de blocs écrits. DI= le décalage de la variable qui contiendra le nombre de blocs écrits.

TU_ERASE

- adresse: \$2C08
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_RENAME

- adresse: \$2C25
- entrée: sur la pile: le segment de la variable fichier, le décalage de la variable fichier, le nouveau nom (octet de longueur au sommet).

TU_CHDIR

- adresse: \$2C56
- entrée: sur la pile: le nom du répertoire (octet de longueur au sommet)

TU_MKDIR

- adresse: \$2C91
- entrée: sur la pile: le nom du répertoire (octet de longueur au sommet)

TU_RMDIR

- adresse: \$2C95
- entrée: sur la pile: le nom du répertoire (octet de longueur au sommet)

TU_GETDIR

- adresse: \$2CA6
- entrée: Sur la pile: le numéro de la table, le segment de la chaîne. DI= décalage de la chaîne, CL= taille maximale de la chaîne.

TU_EXECUTE

- adresse: \$2CED
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

TU_CHAIN

- adresse: \$2CF2
- entrée: Sur la pile: le segment de la variable fichier. DI= décalage de la variable fichier.

7.4 LES TABLES DE LA LIBRAIRIE

Finalement le code de la librairie contient un certain nombre de tables:

- de \$0103 à \$012C le copyright BORLAND avec le numéro de série
- de \$016A à \$017A les paramètres de l'écran utilisé
- de \$09E9 à \$0A25 les adresses des routines d'entrée / sortie (ces adresses sont recopiées dans les globales à l'initialisation)
- en \$1BEC, \$1CA2, \$1D3F, \$1E51, \$1E81 les constantes de calculs de fonctions transcendentes (log, sinus...)
- en \$D1D4 la table des conversions décimales
- en \$2489 la table des noms des périphériques pré déclarés (CON, TRM...)
- en \$28E5 et \$28EA les chaînes TRUE et FALSE

7.5 CONCLUSION

Au cours de ce chapitre nous avons présenté:

- les variables globales d'un programme TURBO PASCAL
- les routines de la librairie directement appelées par le code compilé
- les tables de données situées dans la librairie

CHAPITRE 8 MODIFICATIONS DE TURBO.COM

Ce chapitre va présenter:

- des modification du module de commande de TURBO.COM
- modification des commandes de l'éditeur
- modification des numéros d'erreur de compilation

A l'issue de ce chapitre:

- vous saurez comment éviter de taper Y pour charger les messages d'erreur
- vous pourrez reconfigurer l'éditeur plus facilement
- vous aurez des messages d'erreurs de compilation tous distincts

8.1 AVERTISSEMENT

Nous allons vous indiquer quelles modification nous avons apportées à notre version de TURBO PASCAL. Ces modifications sont valables pour la version 3.02A. Elles détruiront toute autre version de TURBO PASCAL.

Si vous souhaitez apporter certaines de ces modifications, il est impératif de réaliser une sauvegarde de votre version actuelle, et de n'effectuer ces modifications que sur une copie.

Nous ne sommes pas responsables des modifications que vous apporteriez à votre version de TURBO PASCAL.

8.2 MODIFICATION DU MODULE DE COMMANDE

Lors du chargement de TURBO PASCAL, vous devez indiquer si vous souhaitez charger les messages d'erreur ou non. Si le fichier n'est pas là, il n'est de toutes façon pas chargé. La question vous permet donc simplement de disposer pour certains très gros textes d'un peu plus de place mémoire.

Nous préférons modifier le code de TURBO.COM pour toujours charger les messages, quitte à les supprimer de la disquette (ou à les renommer), si d'aventure nous ne souhaitons pas les charger.

Nous avons par la même occasion supprimé la temporisation qui se produit lorsque vous tapez un nom de fichier ne correspondant à aucun fichier disque (temporisation destinée à vous laisser voir le message NEW FILE).

Voici le programme de modification:

```

0 (* 008 msgyes *)
1 (* 22 jul 87 *)
2
3 (*$r+*)
4 PROGRAM modifie_demande_messages;
5   VAR choix, choix1: CHAR;
6
7   PROCEDURE modifie_commande;
8     TYPE string80= STRING[80];
9     VAR l_turbo_com: FILE;
10      l_modification: string80;
11
12   PROCEDURE modifie(p_adresse: INTEGER; p_modification: string80;
13     p_quantite: INTEGER);
14     VAR l_tampon: ARRAY[0..$FF] OF BYTE;
```

```

15 BEGIN (* modifie *)
16   SEEK(l_turbo_com, p_adresse SHR 8);
17   BLOCKREAD(l_turbo_com, l_tampon, 1);
18
19   MOVE(p_modification[1], l_tampon[p_adresse AND $00FF], p_quantite);
20
21   SEEK(l_turbo_com, p_adresse SHR 8);
22   BLOCKWRITE(l_turbo_com, l_tampon, 1);
23 END; (* modifie *)
24
25 BEGIN (* modifie_commande *)
26   WRITE('o(k modifier turbo.com ? ');
27   READ(KBD, choix1); WRITELN(choix1);
28   IF choix1 <> 'o'
29     THEN EXIT;
30
31   ASSIGN(l_turbo_com, 'turboo.com');
32   RESET(l_turbo_com, $100);
33
34   (* supprime la lecture clavier et positionne l'étiquette *)
35   (* remplit la chaîne des modifications *)
36   (* MOV [v_messages], $01 *)
37   l_modification[1]:= CHR($C6);
38   l_modification[2]:= CHR($06);
39   l_modification[3]:= CHR($4A);
40   l_modification[4]:= CHR($03);
41   l_modification[5]:= CHR($01);
42   (* et complète avec des NOP *)
43   FILLCHAR(l_modification[6], 3, $90);
44   modifie($2F53- $0100, l_modification, 8);
45
46   (* supprime la temporisation après NEW FILE *)
47   FILLCHAR(l_modification[1], 6, $90);
48   modifie($349D- $0100, l_modification, 6);
49
50   CLOSE(l_turbo_com);
51 END; (* modifie *)
52
53 BEGIN (* principal *)
54   IF memw[cseg:$0101]+ $0103 <> $2D4D
55     THEN BEGIN
56       WRITELN('cet utilitaire ne fonctionne');
57       WRITELN('qu'avec la version 3.02B de TURBO');
58       EXIT;
59     END;
60
61   REPEAT
62     WRITE('modifier la demande message, q(uite ? ');
63     READ(KBD, choix); WRITELN(choix);
64     CASE choix OF
65       'm': modifie_commande;
66     END;
67   UNTIL choix = 'q';
68 END. (* principal *)
```

Fig. 8.1 Le programme msgyes.

8.3 LES COMMANDES DE L'ÉDITEUR

Le programme TINST livré sur la disquette TURBO PASCAL permet de modifier les commandes de l'éditeur. Si vous trouvez que CTRL Q évoque plus Quitter que KTRL K D, il vous suffit de modifier les commandes de l'éditeur.

En fait TURBO.COM contient deux tables:

- une table des commandes originales, similaires à celle de WORD STAR
- une table des commandes que vous souhaitez utiliser.

Si vous utilisez TINST, la seconde table est modifiée, et ce sont ses codes qui seront utilisés. Si vous n'avez pas utilisé TINST, ou si vous ne modifiez pas certaines commandes, ce seront les commandes originales qui seront utilisées.

Toutefois TINST vous demande de modifier les 45 commandes de l'éditeur, avant de vérifier qu'elles sont toutes distinctes. Si ce n'est pas le cas, vous serez invité à modifier l'ensemble de 45 touches. Et ceci jusqu'à ce que toutes les touches soient distinctes.

Pour rendre ce travail moins pénible, nous utilisons un programme qui permet:

- d'afficher les codes des commandes originales
- d'afficher les codes des commandes actuelles
- de modifier les commandes actuelles et de les placer dans TURBO.COM.

Certes ce programme ne vous permet pas de tester les nouvelles commandes en temps réel. C'est à vous de placer les nouveaux codes dans le texte du programme et de le compiler. Le programme ne vérifie pas que toutes les commandes sont distinctes. Comme il affiche à l'écran les combinaisons choisies, une inspection visuelle suffit en général.

Voici ce programme:

```
0 (* 004 install *)
1 (* 23 aou 87 *)
2
3 (*$r+*)
4 PROGRAM installe_editeur_turbo;
5   CONST k_table_modifiable= $57CE;
6         k_table_wordstar= $5896;
7   TYPE string2= STRING[2];
8         string4= STRING[4];
9         string64= STRING[64];
10  VAR choix, choix1: CHAR;
11      nom_turbo_com: string64;
12      titre_commande: ARRAY[0..45] OF STRING[80];
13
```

```
14 PROCEDURE stop;
15   VAR l_caractere: CHAR;
16   BEGIN
17     READ(KBD, l_caractere);
18     END; (* stop *)
19
20 (* fonctions de conversion *)
21
22 FUNCTION f_octet_hexadecimal(p_octet: BYTE): string2;
23   CONST kt_hexadecimal: ARRAY[0..$0F] OF CHAR= '0123456789ABCDEF';
24   BEGIN
25     f_octet_hexadecimal:= CONCAT(kt_hexadecimal[p_octet DIV 16],
26                                   kt_hexadecimal[p_octet MOD 16]);
27   END; (* f_octet_hexadecimal *)
28
29 FUNCTION f_entier_hexadecimal(entier: INTEGER): string4;
30   BEGIN
31     f_entier_hexadecimal:= CONCAT(f_octet_hexadecimal(HI(entier)),
32                                   f_octet_hexadecimal(LO(entier)));
33   END; (* f_entier_hexadecimal *)
34
35 (* visualisation et modification de la table *)
36
37 PROCEDURE initialise_titre_commande;
38   BEGIN
39     FILLCHAR(titre_commande, SIZEOF(titre_commande), 0);
40
41     titre_commande[0]:= 'return';
42     titre_commande[1]:= '<-';
43     titre_commande[2]:= '<- alt';
44     titre_commande[3]:= '->';
45     titre_commande[4]:= 'mot <';
46     titre_commande[5]:= 'mot >';
47     titre_commande[6]:= 'ligne ^';
48     titre_commande[7]:= 'ligne dessous';
49     titre_commande[8]:= 'scroll up';
50     titre_commande[9]:= 'scroll dwn';
51     titre_commande[10]:= 'page up';
52     titre_commande[11]:= 'page down';
53     titre_commande[12]:= 'ligne <';
54     titre_commande[13]:= 'ligne >';
55     titre_commande[14]:= 'page ^';
56     titre_commande[15]:= 'page bas';
57     titre_commande[16]:= 'debut fichier';
58     titre_commande[17]:= 'fin fichier';
59     titre_commande[18]:= 'debut bloc';
60     titre_commande[19]:= 'fin bloc';
61     titre_commande[20]:= 'dernier curseur';
62     titre_commande[21]:= 'insere remplace';
63     titre_commande[22]:= 'insere ligne';
64     titre_commande[23]:= 'efface ligne';
65     titre_commande[24]:= 'efface fin ligne';
66     titre_commande[25]:= 'efface mot <';
67     titre_commande[26]:= 'efface ss curs';
68     titre_commande[27]:= 'efface >';
69     titre_commande[28]:= 'efface > alt';
```



```

70 titre_commande[29]:= 'mk debut bloc';
71 titre_commande[30]:= 'mk fin bloc';
72 titre_commande[31]:= 'marque mot';
73 titre_commande[32]:= 'cache bloc';
74 titre_commande[33]:= 'copie bloc';
75 titre_commande[34]:= 'deplace bloc';
76 titre_commande[35]:= 'efface bloc';
77 titre_commande[36]:= 'lit bloc';
78 titre_commande[37]:= 'ecris bloc';
79 titre_commande[38]:= 'fin';
80 titre_commande[39]:= 'tab';
81 titre_commande[40]:= 'auto tab';
82 titre_commande[41]:= 'resuscite ligne';
83 titre_commande[42]:= 'trouve';
84 titre_commande[43]:= 'remplace';
85 titre_commande[44]:= 'repete trouve';
86 titre_commande[45]:= 'prefixe controle';
87 END; (* initialise_titre_commande *)
88
89 PROCEDURE charge_et_affiche(p_adresse_table: INTEGER);
90 (* charge la table des codes wordstar et l'affiche *)
91 VAR l_octet: BYTE;
92 l_indice: BYTE;
93 l_taille_commande: INTEGER;
94 l_numero_commande: INTEGER;
95 l_turbo_com: FILE;
96 l_tampon: ARRAY[0..$1FF] OF BYTE;
97 l_indice_tampon: INTEGER;
98 BEGIN
99 CLRSCR;
100
101 (* charge la table *)
102 ASSIGN(l_turbo_com, nom_turbo_com);
103 RESET(l_turbo_com, $100);
104 SEEK(l_turbo_com, (p_adresse_table- $0100) DIV $100);
105 BLOCKREAD(l_turbo_com, l_tampon, 2);
106 CLOSE(l_turbo_com);
107
108 l_indice_tampon:= p_adresse_table AND $00FF;
109 l_numero_commande:= 0;
110
111 REPEAT
112 IF l_numero_commande< 23
113 THEN BEGIN
114 GOTOXY(1, l_numero_commande+ 1);
115 END
116 ELSE GOTOXY(41, l_numero_commande+ 1-23);
117 WRITE(l_numero_commande:2, ' ');
118 WRITE(titre_commande[l_numero_commande]);
119
120 l_taille_commande:= l_tampon[l_indice_tampon];
121 IF l_taille_commande<> 0
122 THEN BEGIN
123 FOR l_indice:= LENGTH(titre_commande[l_numero_commande])+ 1
124 TO 20 DO
125 WRITE(' ');
126 FOR l_indice:= 1 TO l_taille_commande DO
127 WRITE(f_octet_hexadecimal(l_tampon[l_indice_tampon+ l_indice]));

```

```

128 FOR l_indice:= l_taille_commande+ 1 TO 20 DO
129 WRITE(' ');
130 WRITE(' ');
131
132 FOR l_indice:= 1 TO l_taille_commande DO
133 BEGIN
134 l_octet:= l_tampon[l_indice_tampon+ l_indice];
135 IF l_octet IN [0..31]
136 THEN WRITE('^', CHR(l_octet+ $40), ' ')
137 ELSE
138 IF l_octet IN [32..127]
139 THEN WRITE(CHR(l_octet), ' ')
140 ELSE WRITE('. ');
141 END;
142 l_indice_tampon:= l_indice_tampon+ l_taille_commande+ 1;
143 END;
144 l_numero_commande:= l_numero_commande+ 1;
145 UNTIL l_numero_commande> 45;
146 END; (* charge_et_affiche *)
147
148 PROCEDURE remplace_installe;
149 (* modifie la table *)
150 VAR commande: ARRAY[0..45] OF RECORD
151 taille_commande, octet_1, octet_2: BYTE;
152 END;
153 l_tampon_max: INTEGER;
154 l_bloc: INTEGER;
155 indice: INTEGER;
156
157 PROCEDURE remplis_table_temporaire;
158
159 PROCEDURE escape(p_numero_commande, l_indice: BYTE);
160 BEGIN
161 WITH commande[p_numero_commande] DO
162 BEGIN
163 taille_commande:= 2;
164 octet_1:= $1B;
165 octet_2:= l_indice;
166 END;
167 END; (* escape *)
168
169 PROCEDURE controle(p_numero_commande: BYTE; caractere: CHAR);
170 BEGIN
171 WITH commande[p_numero_commande] DO
172 BEGIN
173 taille_commande:= 1;
174 octet_1:= ORD(caractere)- $40;
175 END;
176 END; (* controle *)
177
178 PROCEDURE inutilise(p_numero_commande: BYTE);
179 BEGIN
180 WITH commande[p_numero_commande] DO
181 BEGIN
182 taille_commande:= 1;
183 octet_1:= $FF;
184 END;
185 END; (* inutilise *)

```



```

186 BEGIN (* remplis_table_temporaire *)
187 (* valeurs que vous pouvez modifier *)
188 controle(0, 'M');
189 escape(1, $4B);
190 inutilise(2);
191 escape(3, $4D);
192 escape(4, $43);
193 escape(5, $44);
194 escape(6, $48);
195 escape(7, $50);
196 inutilise(8);
197 inutilise(9);
198 escape(10, $3D);
199 escape(11, $3E);
200 escape(12, $41);
201 escape(13, $42);
202 escape(14, $3F);
203 escape(15, $40);
204 escape(16, $3B);
205 escape(17, $3C);
206 inutilise(18);
207 inutilise(19);
208 controle(20, 'G');
209 escape(21, $52);
210 inutilise(22);
211 inutilise(23);
212 inutilise(24);
213 inutilise(25);
214 escape(26, $53);
215 controle(27, 'H');
216 inutilise(28);
217 controle(29, 'B');
218 controle(30, 'E');
219 inutilise(31);
220 inutilise(32);
221 controle(33, 'K');
222 controle(34, 'V');
223 controle(35, 'D');
224 controle(36, 'R');
225 controle(37, 'W');
226 controle(38, 'C');
227 controle(39, 'I');
228 inutilise(40);
229 inutilise(41);
230 controle(42, 'F');
231 controle(43, 'X');
232 controle(44, 'L');
233 controle(45, 'P');
234 END; (* remplis_table_temporaire *)
235
236
237 PROCEDURE affiche_table_temporaire;
238 VAR l_numero_commande: INTEGER;
239     l_indice: INTEGER;
240 BEGIN
241     CLRSCR;
242     l_numero_commande := 0;
243

```

```

244 REPEAT
245     WITH commande[l_numero_commande] DO
246     BEGIN
247         IF l_numero_commande < 23
248         THEN BEGIN
249             GOTOXY(1, l_numero_commande + 1);
250         END
251         ELSE GOTOXY(41, l_numero_commande + 1 - 23);
252         WRITE(l_numero_commande:2, ' ');
253         WRITE(titre_commande[l_numero_commande]);
254
255         IF taille_commande <> 0
256         THEN BEGIN
257             FOR l_indice :=
258                 LENGTH(titre_commande[l_numero_commande]) + 1 TO 20 DO
259                 WRITE(' ');
260
261             WRITE(taille_commande:2, ' ');
262             WRITE(f_octet_hexadecimal(octet_1));
263             IF taille_commande = 2
264             THEN WRITE(f_octet_hexadecimal(octet_2), ' ')
265             ELSE WRITE(' ');
266
267             IF octet_1 IN [0..31]
268             THEN WRITE('^', CHR(octet_1 + $40), ' ')
269             ELSE
270                 IF octet_1 IN [32..127]
271                 THEN WRITE(CHR(octet_1), ' ')
272                 ELSE WRITE(' ');
273
274             IF taille_commande = 2
275             THEN
276                 IF octet_2 IN [0..31]
277                 THEN WRITE('^', CHR(octet_2 + $40), ' ')
278                 ELSE
279                     IF octet_2 IN [32..127]
280                     THEN WRITE(CHR(octet_2), ' ')
281                     ELSE WRITE(' ');
282
283             WRITELN;
284         END;
285     END;
286     l_numero_commande := l_numero_commande + 1;
287     UNTIL l_numero_commande > 45;
288 END; (* affiche_table_temporaire *)
289
290 PROCEDURE modifie_turbo_com;
291 VAR l_turbo_com: FILE;
292     l_tampon: ARRAY[0..$1FF] OF BYTE;
293     l_indice_tampon: INTEGER;
294     l_numero_commande: INTEGER;
295 BEGIN
296     GOTOXY(1, 24);
297     WRITE('o(k modifier turbo.com ?) ');
298     READ(KBD, choix1); WRITE(choix1);
299     IF choix1 <> 'o'
300     THEN EXIT;
301

```



```
302 (* charge les blocs contenant la table *)
303 ASSIGN(l_turbo_com, nom_turbo_com);
304 RESET(l_turbo_com, $100);
305 SEEK(l_turbo_com, (k_table_modifiable- $0100) DIV $100);
306 BLOCKREAD(l_turbo_com, l_tampon, 2);
307
308 (* place les commandes, avec la syntaxe de TURBO.COM *)
309 l_indice_tampon:= k_table_modifiable AND $00FF;
310 l_numero_commande:= 0;
311
312 REPEAT
313   WITH commande[l_numero_commande] DO
314     BEGIN
315       IF taille_commande <> 0
316         THEN BEGIN
317           l_tampon[l_indice_tampon]:= taille_commande;
318           l_tampon[l_indice_tampon+1]:= octet 1;
319           l_indice_tampon:= l_indice_tampon+2;
320           IF taille_commande= 2
321             THEN BEGIN
322               l_tampon[l_indice_tampon]:= octet 2;
323               l_indice_tampon:= l_indice_tampon+1;
324             END;
325         END; (* WITH *)
326         l_numero_commande:= l_numero_commande+1;
327       UNTIL l_numero_commande > 45;
328
329       l_tampon[l_indice_tampon]:= 0;
330       l_tampon[l_indice_tampon+1]:= $FF;
331
332       (* met sur disque *)
333       SEEK(l_turbo_com, (k_table_modifiable- $0100) DIV $100);
334       BLOCKWRITE(l_turbo_com, l_tampon, 2);
335       CLOSE(l_turbo_com);
336       END; (* modifie_turbo_com *)
337
338 BEGIN (* remplace_installe *)
339   remplis_table_temporaire;
340   affiche_table_temporaire;
341   modifie_turbo_com;
342   END; (* remplace_installe *)
343
344 PROCEDURE initialise;
345 BEGIN
346   nom_turbo_com:= 'turbo.com';
347   initialise_titre_commande;
348   CLRSCR;
349   END; (* initialise *)
350
351 BEGIN
352   initialise;
353
354
```

```
355 IF memw[cseg:$0101]+ $0103 <> $2D4D
356 THEN BEGIN
357   WRITELN('cet utilitaire ne fonctionne');
358   WRITELN('qu''avec la version 3.02B de TURBO');
359   EXIT;
360 END;
361
362 REPEAT
363   GOTOXY(1, 24);
364   WRITELN(nom_turbo_com);
365
366   WRITE('o(riginal, m(odifiée, r(emplace, q(uite ? ');
367   READ(KBD, choix); WRITE(choix);
368   CASE choix OF
369     'r' : remplace_installe;
370     'o' : charge_et_affiche(k_table_wordstar);
371     'm' : charge_et_affiche(k_table_modifiable);
372   END;
373   UNTIL choix='q';
374 END.
```

Fig. 8.2 Le programme *installe_editeur*.

8.4 MODIFICATION DES OPTIONS DE COMPILATION

Nous préférons compiler nos programmes en mode R+ par défaut. Pour éviter d'avoir à placer cette option en début de chaque programme, il suffit de modifier la constante que le compilateur utilise pour initialiser les valeurs utilisées par défaut. Cette constante se trouve à l'adresse \$5A9D, et a par défaut la valeur \$ED.

Voici un programme qui installe la valeur R+ par défaut:

```
0 (* 008 rplus *)
1 (* 22 jul 87 *)
2
3 (* $r+ *)
4 PROGRAM modifie_option_r_plus;
5   VAR choix, choixI: CHAR;
6
7   PROCEDURE modifie_commande;
8     TYPE string80= STRING(80);
9     VAR l_turbo_com: FILE;
10        l_modification: string80;
11
12   PROCEDURE modifie(p_adresse: INTEGER; p_modification: string80;
13     p_quantite: INTEGER);
14     VAR l_tampon: ARRAY[0..$FF] OF BYTE;
15     BEGIN (* modifie *)
16       SEEK(l_turbo_com, p_adresse SHR 8);
17       BLOCKREAD(l_turbo_com, l_tampon, 1);
18
```



```

19  MOVE(p_modification[1], l_tampon[p_adresse AND $00FF], p_quantite);
20
21  SEEK(l_turbo_com, p_adresse SHR 8);
22  BLOCKWRITE(l_turbo_com, l_tampon, 1);
23  END; (* modifie *)
24
25  BEGIN (* modifie commande *)
26  WRITE('o(k modifier turbo.com ? ');
27  READ(KBD, choix1); WRITELN(choix1);
28  IF choix1 <> 'o'
29  THEN EXIT;
30
31  ASSIGN(l_turbo_com, 'turboo.com');
32  RESET(l_turbo_com, $100);
33
34  (* met la valeur $EF à $5A9D *)
35  modifier($5A9D- $0100, CHR($EF), 1);
36
37  CLOSE(l_turbo_com);
38  END; (* modifie *)
39
40  BEGIN (* principal *)
41  IF memw[cseg: $0101] + $0103 <> $2D4D
42  THEN BEGIN
43    WRITELN('cet utilitaire ne fonctionne');
44    WRITELN('qu''avec la version 3.02B de TURBO');
45    EXIT;
46  END;
47
48  REPEAT
49  WRITE('m(odifier l'option R, q(uite ? ');
50  READ(KBD, choix); WRITELN(choix);
51  CASE choix OF
52    'm' : modifie_commande;
53  END;
54  UNTIL choix = 'q';
55  END. (* principal *)

```

Fig. 8.3 Le programme option_r_plus.

8.5 MODIFICATION DES NUMEROS D'ERREUR

Les erreurs du compilateur ont été regroupées en catégories pour diminuer le nombre de messages (et donc la taille du fichier des messages d'erreur).

Dans certaines situations, des messages distincts permettraient de mieux identifier la cause de l'erreur de compilation.

Aussi avons nous modifié les numéros d'erreurs qui peuvent apparaître à plusieurs endroits, de façon que chaque numéro d'erreur soit unique. Les numéros qui ne se trouvent qu'en un seul exemplaire n'ont pas été modifiés.

L'utilitaire qui suite permet donc:

- de modifier les numéros utilisés par le compilateur
- de récupérer dans un fichier la table comportant les numéros ainsi modifiés et les messages correspondants.

Cet utilitaire ne modifie pas le fichier TURBO.MSG. En effet le module de commande n'affiche que les message dont le numéro n'utilise pas plus de 2 chiffres. Comme TURBO utilise 85 numéros, nous ne pourrions pas afficher les messages des erreurs dont le code dépasserait 99. La modification des numéros impose donc l'utilisation d'une table imprimée de ces nouveaux numéros.

Le fichier contenant les nouveaux numéros et leur message peut être modifiée en utilisant l'éditeur, puis imprimée.

Voici le programme de modification:

```

0  (* 008 erreur *)
1  (* 23.aou 87 *)
2
3  (* $r+*)
4  PROGRAM modifie_codes_erreur;
5  VAR choix, choix1: CHAR;
6
7  PROCEDURE modifie_codes;
8  TYPE string80 = STRING[80];
9  VAR l_turbo_com: FILE;
10  l_tampon: ARRAY[0..$FF] OF BYTE;
11  l_bloc_courant: INTEGER;
12  l_a_modifie: BOOLEAN;
13  l_code: BYTE;
14
15  l_sortie: TEXT;
16  l_nom_sortie: STRING[64];
17
18  PROCEDURE modifie(p_vieux_code: BYTE; p_adresse: INTEGER;
19  p_message: string80);
20  VAR l_bloc: INTEGER;
21  BEGIN (* modifie *)
22    l_bloc := (p_adresse - $0100) SHR 8;
23    IF l_bloc <> l_bloc_courant
24    THEN BEGIN
25      IF l_a_modifie
26      THEN BEGIN
27        SEEK(l_turbo_com, l_bloc_courant);
28        BLOCKWRITE(l_turbo_com, l_tampon, 1);
29      END;
30      l_bloc_courant := l_bloc;
31      SEEK(l_turbo_com, l_bloc_courant);
32      BLOCKREAD(l_turbo_com, l_tampon, 1);
33      l_a_modifie := FALSE;
34    END;
35

```



```
36 (* affichage et verification *)
37 WRITELN(l_sortie, p_vieux_code: 3, l_code: 4, ' ', p_message);
38 IF l_tampon[p_adresse AND $00FF] <> p_vieux_code
39 THEN BEGIN
40   WRITELN('erreur');
41   READLN;
42 END;
43
44 (* modifie le code *)
45 l_tampon[p_adresse AND $00FF] := l_code;
46 l_a_modifie := TRUE;
47
48 (* calcule le code suivant, en sautant les codes speciaux *)
49 l_code := l_code + 1;
50 IF l_code = $80
51 THEN l_code := $81;
52 IF l_code = $C9
53 THEN l_code := $CB;
54 END; (* modifie *)
55
56 BEGIN (* modifie commande *)
57   WRITE('o(k modifier turbo.com ? ');
58   READ(KBD, choix1); WRITELN(choix1);
59   IF choix1 <> 'o'
60   THEN EXIT;
61
62   WRITE('nom du fichier de sortie <err.pas> ? ');
63   READLN(l_nom_sortie);
64   ASSIGN(l_sortie, l_nom_sortie);
65   REWRITE(l_sortie);
66
67   ASSIGN(l_turbo_com, 'turbo.com');
68   RESET(l_turbo_com, $100);
69
70   l_bloc_courant := 0;
71   l_a_modifie := FALSE;
72   l_code := 100;
73
74   modifie($5C, $61D3, 'erreur creation fichier');
75   modifie($62, $6504, 'debordement a l'execution. Table des intermediaires');
76   modifie($5A, $65F6, 'erreur chargement OVERLAY');
77   modifie($62, $6695, 'debordement a l'execution. Globales');
78   modifie($62, $669F, 'debordement a l'execution. Locales');
79   modifie($62, $66AF, 'debordement a l'execution. Parametres');
80   modifie($24, $685C, 'attend nom de TYPE');
81   modifie($62, $6893, 'debordement d'indice');
82   modifie($62, $68B5, 'debordement du nombre d'octets du tableau');
83   modifie($31, $6A97, 'taille STRING 0');
84   modifie($1E, $6B3E, 'attend comme borne d'intervalle un TYPE scalaire');
85   modifie($2B, $6C22, 'déjà rencontré ce LABEL');
86   modifie($29, $6CC0, 'compteur du FOR pas déclaré');
87   modifie($1E, $6CDA, 'le compteur du FOR doit être un TYPE simple');
88   modifie($2C, $6CF2,
89     'la borne inférieure du FOR n'a pas le TYPE du compteur');
90   modifie($2C, $6DOA,
91     'la borne supérieure du FOR n'a pas le TYPE du compteur');
92   modifie($0D, $6D13, 'attend DO après le FOR');
93   modifie($0E, $6E73, 'attend le END de l'instruction CASE');
```

```
94 modifie($29, $6E77, 'erreur en fin de CASE');
95 modifie($2E, $6E9B, 'TYPE de l'étiquette du CASE <> du TYPE du sélecteur');
96 modifie($48, $6EB9, 'GOTO hors du bloc');
97 modifie($29, $6EE5, 'nom après WITH inconnu');
98 modifie($0D, $6F38, 'DO attendu après le WITH');
99 modifie($29, $708A, 'nom du paramètre illégal');
100 modifie($29, $71B6, 'peut pas affecter à ce nom');
101 modifie($2C, $722A, 'TYPE dans l'expression incompatibles');
102 modifie($2C, $729B, 'types scalaires dans l'expression incompatibles');
103 modifie($29, $72A7, 'peut pas affecter à une fonction en dehors de son corps');
104 modifie($29, $742F, 'READ FILE ou FILE OF');
105 modifie($29, $743B, 'READ d'une variable');
106 modifie($41, $7531, 'pas WRITE d'un fichier de bloc');
107 modifie($42, $7585, 'pas WRITE de ce TYPE de variables');
108 modifie($23, $760D, 'attend FILE OF');
109 modifie($18, $77D0, 'résultat de VAL doit être réel ou entier');
110 modifie($2F, $7B10, 'test de SET illégal');
111 modifie($2C, $7B1A, 'SETS comparés de TYPE différents');
112 modifie($2F, $7B69, 'pas addition de pointeurs');
113 modifie($2F, $7BD8, 'pour les STRING, seul + est permis');
114 modifie($2F, $7BE9, 'OR ou XOR que sur les entiers et les booléens');
115 modifie($2F, $7C04, 'pas de multiplication de pointeurs');
116 modifie($2F, $7C5A, 'l que entre réels');
117 modifie($2F, $7C6C, 'multiplication, divisions que sur des entiers');
118 modifie($2F, $7C8A, 'NOT que sur des entiers et des booléens');
119 modifie($2C, $7D8E, 'facteurs différents 1');
120 modifie($2C, $7DA5, 'facteurs différents 2');
121 modifie($2C, $7DC5, 'facteurs différents 3');
122 modifie($29, $7EAB, 'facteur illégal');
123 modifie($3E, $7F0A, 'attend facteur tableau de caractères');
124 modifie($3E, $7F16, 'attend tableau de caractères d'une dimension');
125 modifie($3E, $7F1F, 'attend tableau de caractères d'indice 0.255');
126 modifie($2C, $7FFE, 'éléments de SET littéral de types différents');
127 modifie($2C, $8019, 'borne supérieure du SET littéral de TYPE différent');
128 modifie($1E, $804D, 'ne peut surtyper que des types simples');
129 modifie($23, $81FC, 'EOLN que de fichiers TEXT');
130 modifie($3F, $8216, 'pas FILESIZE ou FILEPOS de fichiers TEXT');
131 modifie($3F, $822C, 'pas LONGFILEZIZE ou POS de fichiers TEXT');
132 modifie($29, $828B, 'le paramètre de OFS doit être un nom');
133 modifie($29, $82A3, 'le paramètre de SEG doit être un nom');
134 modifie($29, $82DF, 'le paramètre de SIZEOD doit être un nom');
135 modifie($29, $8368, 'nom du paramètre de la fonction standard inconnu');
136 modifie($15, $836C, 'le paramètre de la fonction standard doit être un fichier');
137 modifie($2C, $8557, 'indice du mauvais TYPE');
138 modifie($2D, $856E, 'indice constant hors intervalle');
139 modifie($29, $8646, 'nom de zone inconnu');
140 modifie($29, $8772, 'attend un nom');
141 modifie($29, $88D4, 'attend une constante');
142 modifie($16, $88DF, 'attend une constante entière');
143 modifie($19, $8949, 'signe + ou - que devant entier ou réel');
144 modifie($62, $8B64, 'code dépasse 64K');
145
146 CLOSE(l_turbo_com);
147 CLOSE(l_sortie);
148 END; (* modifie_codes *)
149
```



```

150 BEGIN (* principal *)
151 IF memw/cseg:$0101]+ $0103<> $2D4D
152 THEN BEGIN
153   WRITELN('cet utilitaire ne fonctionne');
154   WRITELN('qu'avec la version 3.02B de TURBO');
155   EXIT;
156 END;
157
158 REPEAT
159 WRITE('m(odifier codes erreur, q(uitte ? ');
160 READ(KBD, choix); WRITELN(choix);
161 CASE choix OF
162   'm' : modifie_codes;
163   END;
164 UNTIL choix='q';
165 END. (* principal *)

```

Fig. 8.4 Le programme *modifie_codes_erreur*.

8.6 CONCLUSION

Ce chapitre a présenté des modifications du module de commande, de la table des commandes de l'éditeur, des options de compilation par défaut et des numéros d'erreur.

LISTE DES PROGRAMMES

Ce livre contient deux types de programmes:

- des utilitaires de diagnostic et de traitement
- des programmes qui nous ont servi à analyser le fonctionnement de TURBO PASCAL.

Le second type de programmes n'a servi qu'à notre exposé, et nous n'en référons pas la liste. Les utilitaires peuvent au contraire vous servir pour la programmation en TURBO PASCAL. La table qui suit vous permettra de les retrouver plus facilement dans le livre. Ce sont eux qui sont placés sur la disquette que vous pouvez vous procurer à l'Institut Pascal.

AFFICHAGE MEMOIRE ET DISQUE

affichage décimal du contenu de la mémoire	45
affichage hexadécimal d'un entier	50
affichage hexadécimal du contenu mémoire	52
affichage binaire d'un entier	54
affichage hexadécimal d'un secteur disque	55
affichage du plan mémoire TURBO PASCAL	70
affichage du bloc de contrôle d'un fichier	306
affichage du bloc de contrôle d'un fichier TEXT	318
affichage des variables globales d'un programme	433

TRAITEMENTS

décodage d'un réel	124
décodage d'un réel BCD	132
décodage d'un réel 8087	137
ouverture de fichier	405
test de présence d'un fichier	407
conversion majuscule de fichier	423

MODIFICATION DE TURBO.COM

chargement automatique des messages	468
reconfiguration des touches de l'éditeur	470
modification de l'option R- par défaut	477
modification des numéros d'erreur du compilateur	478

TABLE DES MATIERES

1	INTRODUCTION	17
1.1	LES OBJECTIFS DE CE LIVRE	18
1.2	LA CONFIGURATION ET LES LOGICIELS UTILISES	18
1.3	NOTATIONS	19
1.4	AMELIORATIONS	19
2	LES DONNEES EN 8088	21
2.1	LES UNITES EN INFORMATIQUE	22
2.1.1	LE BIT ET LES REGROUPEMENTS DE BITS	22
2.1.2	LES BASES DE NUMERATION	23
2.1.3	LE QUARTET	24
2.1.4	L'OCTET	24
2.1.5	LE MOT	25
2.1.6	LE SECTEUR, LE BLOC, LE CLUSTER	26
2.1.7	LE K	26
2.1.8	LE PARAGRAPHE	27
2.2	LE FONCTIONNEMENT DU 8088	27
2.2.1	L'ARCHITECTURE DU PC	27
2.2.2	L'ADRESSAGE SEGMENTE DU 8088	29
2.2.3	LE CODE, LES DONNEES ET LA PILE	35
2.2.4	LES REGISTRES DU 8088	41
2.2.5	LE REGISTRE D'ETAT	42
2.2.6	LES INSTRUCTIONS DU 8088	42
2.3	PROGRAMMES D'EXAMEN	45
2.3.1	AFFICHAGE DU CONTENU MEMOIRE	45
2.3.2	LES MANIPULATIONS AU NIVEAU DU BIT EN TURBO	46
2.3.3	AFFICHAGE HEXADECIMAL	49
2.3.4	LE PROGRAMME D'AFFICHAGE STANDARD	52
2.3.5	AFFICHAGE BINAIRE	54
2.3.6	LECTURE D'UN BLOC DISQUE	55
2.4	CONCLUSION	58
2.5	BIBLIOGRAPHIE	58
3	ORGANISATION MEMOIRE DE TURBO PASCAL	59
3.1	ORGANISATION MEMOIRE DU PC	60
3.1.1	LES COUCHES DU PC	60
3.1.2	LA MEMOIRE DU PC ETEINT	60
3.1.3	ORGANISATION MEMOIRE DU DOS	62
3.2	L'ORGANISATION MEMOIRE DE TURBO PASCAL	63
3.3	COMPILATION D'UN PROGRAMME EN MEMOIRE	65
3.4	L'AFFICHAGE DU PLAN MEMOIRE	69
3.5	ORGANISATION D'UN PROGRAMME .COM	76
3.6	COMPILATION D'UN PROGRAMME AVEC INCLUDE	79
3.7	ORGANISATION D'UN PROGRAMME AVEC OVERLAY	83
3.8	ORGANISATION D'UN PROGRAMME .CHN	87
3.9	ORGANISATION D'UN PROGRAMME AVEC EXTERNAL	93
3.A	ORGANISATION D'UN PROGRAMME AVEC INLINE	94
3.B	LA CAPACITE MEMOIRE	96
3.C	LE MENU OPTION DE TURBO	96
3.D	CONCLUSION	97

4	IMPLEMENTATION DES TYPES	99
4.1	LES TYPES SIMPLES ELEMENTAIRES	100
4.1.1	LE TYPE ENTIER	100
4.1.1.1	DEFINITION DU TYPE ENTIER	100
4.1.1.2	REPRESENTATION DU TYPE ENTIER	100
4.1.1.3	IMPLANTATION DES ENTIERS	102
4.1.1.4	EXAMEN MEMOIRE	103
4.1.1.5	RESUME	104
4.1.2	LE TYPE CARACTERE	105
4.1.2.1	DEFINITION DU TYPE CARACTERE	105
4.1.2.2	REPRESENTATION DU TYPE CARACTERE	105
4.1.2.3	EXAMEN DE LA MEMOIRE	107
4.1.2.4	LES CARACTERES ET LES PERIPHERIQUES	108
4.1.2.5	RESUME	109
4.1.3	LE TYPE BOOLEEN	110
4.1.3.1	DEFINITION DU TYPE BOOLEEN	110
4.1.3.2	REPRESENTATION DU TYPE BOOLEEN	110
4.1.3.3	EXAMEN DE BOOLEENS	111
4.1.3.4	RESUME	113
4.1.4	LE TYPE ENUMERE	113
4.1.4.1	DEFINITION DU TYPE ENUMERE	113
4.1.4.2	REPRESENTATION DU TYPE ENUMERE	113
4.1.4.3	IMPLANTATION DU TYPE ENUMERE	114
4.1.4.4	EXAMEN DES TYPES ENUMERES	114
4.1.4.5	RESUME	116
4.1.5	LE TYPE INTERVALLE NUMERIQUE	116
4.1.5.1	DEFINITION DU TYPE INTERVALLE	116
4.1.5.2	REPRESENTATION DU TYPE INTERVALLE	116
4.1.5.3	IMPLANTATION EN MEMOIRE	116
4.1.5.4	EXAMEN DU TYPE INTERVALLE	117
4.1.5.5	RESUME	120
4.2	LE TYPE REEL	120
4.2.1	DEFINITION	120
4.2.2	REPRESENTATION DES REELS	121
4.2.3	IMPLANTATION MEMOIRE	122
4.2.4	EXAMEN DES REELS	123
4.2.5	DECODAGE D'UN REEL	124
4.2.6	RESUME	128
4.3	LE TYPE BCD	128
4.3.1	DEFINITION	128
4.3.2	UTILITE DES REELS BCD	129
4.3.3	REPRESENTATION DES REELS BCD	129
4.3.4	IMPLANTATION MEMOIRE	130
4.3.5	EXAMEN DES REELS BCD	130
4.3.6	DECODAGE D'UN REEL BCD	132
4.3.7	RESUME	134
4.4	LE TYPE REEL 8087	134
4.4.1	DEFINITION	134
4.4.2	UTILITE DES REELS 8087	136
4.4.3	REPRESENTATION DES REELS 8087	136
4.4.4	IMPLANTATION MEMOIRE	138
4.4.5	EXAMEN DES REELS 8087	138
4.4.6	DECODAGE D'UN REEL 8087	139
4.4.7	RESUME	142

4.5	LE TYPE STRUCTURE TABLEAU	142
4.5.1	DEFINITION	142
4.5.2	IMPLEMENTATION DES TABLEAUX	142
4.5.3	IMPLANTATION DES TABLEAUX EN MEMOIRE	144
4.5.4	EXAMEN D'UN TABLEAU	145
4.5.5	LES TABLEAUX COMPACTES	147
4.5.6	LES TABLEAUX DE CARACTERES	149
4.5.7	LES CHAINES DE CARACTERES	150
4.5.8	RESUME	154
4.6	LES ENREGISTREMENTS	154
4.6.1	DEFINITION	154
4.6.2	IMPLEMENTATION DES ENREGISTREMENTS	154
4.6.3	IMPLANTATION MEMOIRE DES ENREGISTREMENTS	155
4.6.4	LES ENREGISTREMENTS VARIANTS	157
4.6.4.1	DEFINITION	157
4.6.4.2	EXEMPLE D'ENREGISTREMENT VARIANT	158
4.6.4.3	IMPLEMENTATION DES ENREGISTREMENTS VARIANTS	161
4.6.4.4	IMPLEMENTATION DES ENREGISTREMENTS VARIANTS LIBRES	164
4.6.4.5	RISQUE DES ENREGISTREMENTS VARIANTS	166
4.6.5	RESUME	168
4.7	LES ENSEMBLES	168
4.7.1	DEFINITION	168
4.7.2	IMPLEMENTATION DES ENSEMBLES NUMERIQUES	168
4.7.3	IMPLANTATION EN MEMOIRE DES ENSEMBLES	170
4.7.4	EXAMEN MEMOIRE DES ENSEMBLES	170
4.7.5	IMPLEMENTATION D'ENSEMBLES NON NUMERIQUES	174
4.7.6	RESUME	176
4.8	LE TYPE POINTEUR	176
4.8.1	DEFINITION	176
4.8.2	LE MECANISME D'ALLOCATION	177
4.8.3	IMPLEMENTATION DES POINTEURS	185
4.8.4	IMPLANTATION MEMOIRE DU TYPE POINTEUR	186
4.8.5	VISUALISATION DE LA VALEUR D'UN POINTEUR	186
4.8.6	EXAMEN DES POINTEURS	186
4.8.7	LES POINTEURS D'ENREGISTREMENTS VARIANTS	188
4.8.8	RESUME	189
4.9	FORMAT DES FICHIERS SUR DISQUETTE	190
4.9.1	INTRODUCTION	190
4.9.2	LES FICHIERS DE CODE	190
4.9.3	LES FICHIERS DE DONNEES	191
4.9.4	LES FICHIERS DE BLOCS	193
4.9.5	LES FICHIERS DE TEXTE	193
4.A	CONCLUSION	196
4.B	BIBLIOGRAPHIE	197
5	LE CODE COMPILE	199
5.1	L'ANALYSE D'UN PROGRAMME COMPILE	200
5.1.1	L'ANALYSE DU CODE	200
5.1.2	L'ANALYSE D'UN PROGRAMME SIMPLE	200
5.1.3	LA STRATEGIE DE COMPILATION	202
5.2	LES PHASES DE COMPILATION	205
5.2.1	LES ELEMENTS D'UN PROGRAMME	205
5.2.2	LES DEFINITIONS DE LABELS	206
5.2.3	LES DEFINITIONS DE TYPES	206
5.2.4	LES DEFINITIONS DE CONSTANTES SIMPLES	207
5.2.5	LES VARIABLES	209
5.2.6	LES INSTRUCTION PASCAL	211

5.3	ENTIERS, CARACTERES, BOOLEENS, ENUMERES, REELS	211
5.3.1	LES ENTIERS	211
5.3.1.1	ARITHMETIQUE ENTIERE	211
5.3.1.2	COMPARAISON D'ENTIERS	214
5.3.1.3	AND, OR, XOR, SHR et SHL	216
5.3.1.4	PROCEDURES ET FONCTIONS STANDARD	216
5.3.1.5	AFFECTATION ET EXPRESSION D'INTERVALLES	216
5.3.1.6	RESUME	223
5.3.2	AFFECTATION ET EXPRESSIONS REELLES	223
5.3.2.1	AFFECTATION D'UN REEL	223
5.3.2.2	ARITHMETIQUE REELLE	224
5.3.2.3	COMPARAISON DE REELS	226
5.3.2.4	LES CONSTANTES ET VALEURS LITTERALES REELLES	227
5.3.2.5	LES PROCEDURES STANDARD	229
5.3.2.6	READ ET WRITE DE REELS	230
5.3.2.7	LES REELS ET LES ENTIERS	232
5.3.2.8	RESUME	235
5.3.3	LE TYPE ENUMERE	235
5.3.4	LE TYPE BOOLEEN	237
5.3.4.1	AFFECTATION DE BOOLEENS	237
5.3.4.2	ARITHMETIQUE BOOLEENNE	239
5.3.4.3	AFFICHAGE DE BOOLEENS	240
5.3.4.4	RESUME	241
5.3.5	LES CARACTERES	242
5.3.5.1	AFFECTATION DE CARACTERES	242
5.3.5.2	LECTURE ET ECRITURE DE CARACTERES	243
5.3.6	RESUME SUR LES TYPES SIMPLES	245
5.4	LES TYPES STRUCTURES	245
5.4.1	GENERALITES	245
5.4.2	LES TABLEAUX	246
5.4.2.1	TRAITEMENT DES INDICES CONSTANTS	246
5.4.2.2	AFFECTATION A UNE CELLULE	248
5.4.2.3	OPTIMISATION DU CALCUL D'INDICE	250
5.4.2.4	ACCES A UNE CELLULE DANS UNE EXPRESSION	252
5.4.2.5	TABEAU A DEUX DIMENSIONS	253
5.4.2.6	LES OPERATIONS GLOBALES SUR LES TABLEAUX	255
5.4.2.7	RESUME SUR LES TABLEAUX	257
5.4.3	LE TYPE STRING	257
5.4.3.1	PARTICULARITES DU TYPE STRING	257
5.4.3.2	AFFECTATION DE CHAINE	258
5.4.3.3	COMPARAISON DE CHAINES	259
5.4.3.4	LECTURE ET ECRITURE DE CHAINES	261
5.4.3.5	PROCEDURES ET FONCTIONS STRING STANDARD	263
5.4.3.6	RESUME	265
5.4.3.7	ASSIMILATION DES CARACTERES AUX CHAINES	265
5.4.3.8	LES TABLEAUX DE CARACTERES	267
5.4.4	LES ENREGISTREMENTS	270
5.4.4.1	AFFECTATION ET EXPRESSION D'ENREGISTREMENTS	270
5.4.4.2	STRUCTURES DE STRUCTURES	271
5.4.4.3	L'INSTRUCTION WITH	273
5.4.4.4	RESUME	276
5.4.5	LES ENSEMBLES	276
5.4.5.1	AFFECTATION D'UN ENSEMBLE	276
5.4.5.2	ARITHMETIQUE SUR LES ENSEMBLES	278
5.4.5.3	LES TESTS SUR LES ENSEMBLES	280
5.4.5.4	RESUME	282

5.4.6	LES POINTEURS	282
5.4.6.1	AFFECTATION DE POINTEURS	282
5.4.6.2	LES PROCEDURES DE TRAITEMENT DE POINTEURS	284
5.4.6.3	AFFECTATION A LA VARIABLE POINTEE	286
5.4.6.4	LES POINTEURS DE TABLEAUX:	288
5.4.6.5	POINTEURS ET ENREGISTREMENTS	289
5.4.6.6	LES POINTEURS ET WITH	292
5.4.6.7	RESUME	294
5.5	READ ET WRITE	294
5.5.1	INTRODUCTION	294
5.5.2	WRITE(OUTPUT) ET READ(INPUT)	294
5.5.2.1	WRITE VERS OUTPUT	294
5.5.2.2	WRITELN	297
5.5.2.3	FORMATAGE DE L'AFFICHAGE	298
5.5.2.4	READ DE INPUT	300
5.5.2.5	READLN	302
5.5.3	LES FICHIERS SUR DISQUE	303
5.5.3.1	TRAITEMENT D'UN FICHIER DE FICHES	303
5.5.3.2	LE BLOC DE CONTROLE D'UN FICHIER	306
5.5.3.3	LES AUTRES PROCEDURES STANDARD	307
5.5.4	LES FICHIERS DE BLOCS	310
5.5.4.1	TRAITEMENT D'UN FICHIER DE BLOCS	310
5.5.4.2	LES AUTRES PROCEDURES STANDARD	312
5.5.5	LES FICHIERS DE TYPE TEXT	312
5.5.5.1	TRAITEMENT D'UN FICHIER DE TYPE TEXT	312
5.5.5.2	LES PERIPHERIQUES COMME FICHIERS TEXT	314
5.5.5.3	LE BLOC DE CONTROLE DES FICHIERS TEXT	318
5.5.6	RESUME	323
5.6	LES INSTRUCTIONS DE CONTROLE	324
5.6.1	IF	324
5.6.2	CASE	326
5.6.3	REPEAT	329
5.6.4	WHILE	331
5.6.5	FOR	333
5.6.6	EXIT	337
5.6.7	L'INSTRUCTION GOTO	338
5.7	LES APPELS DE PROCEDURES	341
5.7.1	UN APPEL DE ROUTINE 8088	341
5.7.2	APPEL DE PROCEDURE SIMPLE	345
5.7.3	UNE PROCEDURE AVEC DES VARIABLES LOCALES	349
5.7.4	UNE CHAINE D'APPELS	353
5.7.5	APPELS EMBOITES	357
5.7.6	LES PARAMETRES VALEUR ET VARIABLE	364
5.7.7	LES APPELS DE FONCTIONS	371
5.8	LES CONSTANTES TYPEES	376
5.8.1	DEFINITION DES CONSTANTES TYPEES	376
5.8.2	LES CONSTANTES TYPEES SIMPLES	377
5.8.3	LES CONSTANTES TYPEES COMPLEXES	379
5.8.4	RESUME	381
5.9	LES VARIABLES ABSOLUES	382
5.9.1	DEFINITION DES VARIABLES ABSOLUES	382
5.9.2	ACCES AUX VARIABLES ABSOLUES	382
5.9.3	RESUME	384
5.A	SYNTHESE	384
5.B	CONCLUSION	385

6	LES OPTIONS DE COMPILATION	387
6.1	LES OPTIONS DE COMPILATION	388
6.1.1	DEFINITION	388
6.1.2	SYNTAXE	388
6.1.3	EFFET	389
6.1.4	LA LISTE DES OPTIONS	389
6.2	LES FICHIERS INCLUS: L'OPTION I	390
6.2.1	DEFINITION	390
6.2.2	MOTIVATION	390
6.2.3	FONCTIONNEMENT	391
6.2.4	UTILISATION	392
6.3	LA GESTION DES ERREURS: R, I, K, V	394
6.3.1	LA GESTION DES ERREURS	394
6.3.2	LES OPTIONS R+ R-	396
6.3.2.1	DEFINITION DE L'OPTION R	396
6.3.2.2	MOTIVATION	396
6.3.2.3	FONCTIONNEMENT	397
6.3.2.4	UTILISATION	401
6.3.3	LES OPTIONS I+ I-	401
6.3.3.1	DEFINITION DE L'OPTION I	401
6.3.3.2	MOTIVATION	402
6.3.3.3	FONCTIONNEMENT	402
6.3.3.4	UTILISATION	405
6.3.4	LES OPTIONS K+ K-	407
6.3.4.1	DEFINITION DE L'OPTION K	407
6.3.4.2	MOTIVATION	407
6.3.4.3	FONCTIONNEMENT	408
6.3.4.4	UTILISATION	409
6.3.5	L'OPTION V	409
6.3.5.1	DEFINITION	409
6.3.5.2	MOTIVATION	410
6.3.5.3	FONCTIONNEMENT	410
6.4	LE CLAVIER: B, D, C, U	411
6.4.1	LES OPTIONS B+ B-	411
6.4.1.1	DEFINITION DE L'OPTION B	411
6.4.1.2	MOTIVATION	412
6.4.1.3	FONCTIONNEMENT	413
6.4.1.4	UTILISATION	415
6.4.2	LES OPTIONS D+ ET D-	415
6.4.2.1	DEFINITION	415
6.4.2.2	MOTIVATION	416
6.4.2.3	FONCTIONNEMENT	416
6.4.2.4	UTILISATION	417
6.4.3	LES OPTIONS C+ C-	417
6.4.3.1	DEFINITION	417
6.4.3.2	MOTIVATION	417
6.4.3.3	FONCTIONNEMENT	417
6.4.3.4	UTILISATION	417
6.4.4	LES OPTIONS U+ ET U-	417
6.4.4.1	DEFINITION	417
6.4.4.2	MOTIVATION	417
6.4.4.3	FONCTIONNEMENT	417
6.4.4.4	UTILISATION	421
6.5	LES FICHIERS: F, G, P	422
6.5.1	L'OPTION F	422
6.5.1.1	DEFINITION DE L'OPTION F	422
6.5.1.2	MOTIVATION	422
6.5.1.3	FONCTIONNEMENT	422

- caractères 109
^ - caractères 109

A

ABS - compilation 216
ABS - compilation réels 229
absolues - variables 382
ABSOLUTE 180
absolute - compilation 382
adresse - paramètres 364
affichage - pointeurs 186
affichage disque 55
affichage hexadécimal 449
affichage mémoire 449
affichage plan mémoire 69
allocation - compilation 209
AND 47
AND - compilation 216
appel de procédure - compilation 341
appels emboîtés 357
ASCII 105
ASSIGN - compilation 303, 310, 312
AUX - compilation 314

B

B - option Buffer 411
BCD - réel 128
BIOS 60
bit - traitement de 47
bloc de contrôle - TEXT 318
bloc de contrôle fichier 306
BLOCKREAD 55
BLOCKREAD - compilation 310
BLOCKWRITE - compilation 310
BOOLEAN - implémentation 110
booléens - compilation 237
BYTE 117

C

C - option Control 417
caractère - implémentation 105
caractères - compilation 242
caractères - tableaux 149
caractères et STRING 265
CASE - compilation 326
CHAR - implémentation 105
chaînage de programmes 87
chaîne - compilation 261
chaîne d'appel de procédures 353
chaînes - implémentation 150
chaînes et caractères 265

CHN - compilation 87
CHR 109
CLOSE - compilation 303, 310, 312
code - structure globale 201
code ASCII 105
COM - compilation 78
COM - format 190
COMMAND.COM 63
compactage - tableaux 147
compilation - absolute 382
compilation - appel de procédure 341
compilation - ASSIGN 303, 310, 312
compilation - AUX 314
compilation - BLOCKREAD 310
compilation - BLOCKWRITE 310
compilation - booléens 237
compilation - caractères 242
compilation - CASE 326
compilation - CHN 87
compilation - CLOSE 303, 310, 312
compilation - CON 314
compilation - CONST 207
compilation - constantes 376
compilation - enregistrements 270
compilation - ensembles 276
compilation - entiers 211
compilation - énumérés 235
compilation - EOF 314
compilation - EOLN 314
compilation - EXIT 337
compilation - EXTERNAL 93
compilation - fichiers 294
compilation - FILE 310
compilation - FILE OF 303
compilation - fonction 371
compilation - FOR 333
compilation - GOTO 338
compilation - IF 324
compilation - include 79
compilation - INLINE 94
compilation - INPUT 294
compilation - intervalle 219
compilation - KBD 314
compilation - LABEL 206
compilation - littéral 208
compilation - LST 314
compilation - MEMAVAIL 284
compilation - NIL 282
compilation - options de 387
compilation - overlay 83
compilation - OUTPUT 294
compilation - paramètres 364
compilation - pointeurs 282
compilation - PTR 284
compilation - READ 294, 303, 312
compilation - READLN 302
compilation - REPEAT 239

compilation - **REWRITE** 303, 310, 312
 compilation - réels 223
 compilation - **SET** 276
 compilation - **STRING** 257
 compilation - tableaux 245
 compilation - **TEXT** 312
 compilation - **TRM** 314
 compilation - **TYPE** 206
 compilation - **USR** 314
 compilation - **VAR** 209
 compilation - **WITH** 273
 compilation - **WHILE** 331
 compilation - **WRITE** 294, 297, 303, 312
 compilation - **Writeln** 297
 compilation mémoire 65
 compilation sur disque 78
 complément à 2 100
CON - compilation 314
CONCAT - STRING 263
CONFIG.SYS 63
 configuration - touches éditeur 470
CONST - compilation 207
 constante typée - structurée 379
 constantes typées - compilation 376
 coprocesseur 8087 135
COS - compilation réels 229
CSEG 69

D

D - option Device 415
 D option 423
DELETE - STRING 263
 disque - compilation sur 78
DIV - compilation 211
DOS 60, 62
DSEG 69

E

éditeur - installation 470
 emboîtés - appels 357
 enregistrement variant 157
 enregistrements - compilation 270
 enregistrements - implémentation 154
 enregistrements et pointeurs 289
 ensemble - implémentation 168
 ensembles - compilation 276
 entier - formatage **WRITE** 298
 entier - implémentation 100
 entiers - compilation 211
 entiers et réels 232
 énuméré - compilation 235
 énuméré - implémentation 113

EOF - compilation 314
EOLN - compilation 314
 erreurs - gestion 394
 erreurs de compilation 478
EXIT - FOR 339
EXIT - compilation 337
EXTERNAL - compilation 93

F

F - option Files 422
FCB 306
FCB - TEXT 318
FIB 306
FIB - TEXT 318
 fichier - **TEXT** 193
 fichier - bloc de contrôle 306
 fichier .COM 190
 fichiers - compilation 294
 fichiers - format 190
 fichiers de bloc - format 193
FILE - compilation 310
FILE - format 193
FILE OF - format 191
FILE OF - compilation 303
FILES 63
FILLCHAR 69
 flottaisons - entiers 232
 fonction - compilation 371
FOR - compilation 333
FOR - EXIT 339
FOR - GOTO 339
 format - **TEXT** 193
 format **FILE** 193
 format **FILE OF** 191
 formatage numérique - compilation 298

G

G - option Get 423
GETMEM 180
 globale - opération tableaux 255
 globales 426
GOTO - FOR 339
GOTO - compilation 338

H

handle fichier 306
HEAPPTR 69, 178
 hexadécimal - conversion 449
HI 449

I

I - option 390, 401
 IF - compilation 324
 implementation **INTEGER** 100
 implémentation - **BOOLEAN** 110
 implémentation - **CHAR** 105
 implémentation - **REAL** 120
 implémentation - **SET** 168
 implémentation - **STRING** 150
 implémentation - enregistrements 154
 implémentation - intervalle 116
 implémentation - pointeur 176
 implémentation - réel 8087 128, 135
 implémentation - tableaux 143
 implémentation - variants 157
 implémentation - énuméré 113
 IN - compilation 280
 include - compilation 79
 indice - compilation 245
INLINE - compilation 94
INPUT - compilation 300
INPUT - compilation 294
INTEGER - compilation 211
INTEGER - implémentation 100
INTEL - format 8087 135
 intervalle - compilation 219
 intervalle - implémentation 116
 intervalle - option 396
IORESULT 401

K

K - option 407
KBD - compilation 314
KEYBFR 63

L

LABEL - compilation 206
LENGTH 151
 librairie - les routines 443
 librairie - tables 465
 libres - variants 164
 littéral - compilation 208
LO 449
 locales - variables 349
LST - compilation 314

M

MAXINT 100
MEM 45
MEMAVAIL - compilation 284
 mémoire 60
 mémoire - affichage plan 69
MEMW 74
 menu - options 90, 96
 messages - chargement 468
MOD - compilation 211

N

NEW 178
NEW - variants 188
NIL - compilation 282
NOT 47
NOT - compilation 216

O

OFS 69
 optimisation - calcul d'indice 249
 option - menu 96
 option B (Buffer) 411
 option C (Contrôle) 417
 option D 423
 option D (Device) 415
 option F (Files) 422
 option G (Get) 423
 option I (Include) 390
 option I (input output) 401
 option K (stack check) 407
 option P (Put) 423
 option R 249
 option R (range check) 396
 option R - modification 477
 option U (User) 419
 option V (Verify) 409
 options - paramètre 201
 options de compilation 387
 options du menu 90
OR 47
OR - compilation 216
ORD 109
OUTPUT - compilation 294
OVERLAY - compilation 83

P

P - option Put 423
PACKED 147
 paramètres - compilation 364
 plan mémoire 60, 65
 plan mémoire - affichage 69
 plan mémoire DOS 62
 pointeur - implémentation 176
 pointeurs - **WITH** 292
 pointeurs - affichage 186
 pointeurs - compilation 282
 pointeurs - variants 188
 pointeurs de tableaux 288
 pointeurs et enregistrements 289
 PSP 66
PTR - compilation 284

R

R - modification de l'option 477
 R - option 249, 396
READ - STRING 261
READ - caractères 243
READ - compilation 300
READ - compilation 294, 303, 312
READ - compilation réels 230
READ - entier compilation 216
READ - tableaux de caractères 268
READLN - compilation 302
REAL - implémentation 120
RECORD - compilation 270
RECORD - implémentation 154
RECORD variant 157
 réel - formatage **WRITE** 298
 réel - implémentation 120
 réel 8087 - implémentation 135
 réel BCD - implémentation 128
 réels - compilation 223
 réels et entiers 232
REPEAT - compilation 239
REWRITE - compilation 303, 310, 312
 ROM 60
 routines librairie 443

S

SEG 69
SET - compilation 276
SET - implémentation 168

SHL 48
SHL - compilation 216
SHR 48
SHR - compilation 216
SQR - compilation 211
SSEG 69
 statiques - variables 376
STR - STRING 263
STRING - compilation 257
STRING - implémentation 150
STRING et caractères 265
 structurée - constante typée 379
SUCC - compilation 211

T

tableau - implémentation 143
 tableau à 2 dimensions 253
 tableaux - affectation globale 255
 tableaux - compilation 245
 tableaux - pointeurs de 288
 tableaux de caractères 149, 267
TEXT - compilation 312
TEXT - format fichier 193
TEXT - FCB 318
TEXT - FIB 318
TEXT - blocETEITE 318
TINST 470
 traitement de bits 47
TRM - compilation 314
TYPE - compilation 206
 type **INTEGER** 100

U

U - option User 419
 union libre 164
USR - compilation 314

V

V - option Verify 409
VAL - STRING 263
 valeur - paramètres 364
VAR - compilation 209
 variable - paramètres 364
 variables - absolues 382
 variables globales 426
 variables initialisées 376
 variables locales - compilation 349

AU COEUR DE TURBO PASCAL présente les structures et le fonctionnement des programmes compilés avec TURBO PASCAL sur PC et compatible.

AU COEUR DE TURBO PASCAL décrit:

- l'architecture mémoire du 8088, du PC, du DOS, de TURBO PASCAL, et des programmes compilés avec TURBO PASCAL (compilation mémoire, disque, programmes avec OVERLAY et chaînage).
- le format de chaque type de données:
 - les types simples: entiers, réels, réels BCD, réels 8087, caractères booléens, énumérés
 - les types structurés: tableau, enregistrement, ensemble, pointeur
 - les fichiers
- le code généré par le compilateur pour les expressions, les instructions de contrôle, les appels de procédures, les appels de fonctions, les constantes typées et les variables absolues
- les variables globales utilisées par un programme
- les routines de la librairie avec leurs paramètres

Plus de **52 schémas** illustrent les **512 pages** de ce livre, qui comporte 149 programmes, dont **19 utilitaires** permettant:

- l'affichage du plan mémoire
- l'affichage du contenu mémoire ou disque
- le décodage des réels
- la visualisation des variables globales
- des modifications du fonctionnement de TURBO PASCAL.

X67, MBA Columbia, **John COLIBRI** a appris la gestion chez Ferrodo, la fabrication et la vente chez TI, le marketing chez Nestlé et la chasse chez Compétences, avant de s'installer à son compte en 1980. Auteur de 10 livres sur la micro informatique, il fonde en 1983 l'INSTITUT PASCAL, où il vous propose plus de 350 livres sur PASCAL, réunit le mercredi le Club des Utilisateurs PASCAL, publie le magazine PASCALISSIME et anime tous les mois ses célèbres séminaires PASCAL.