GUIDE POUR L'UTILISATEUR DU PASCAL UCSD rédigé par Daniel.Smague

THOMSON TO8 /TO9/TO9+
PC émulateur TEO

I - HISTORIQUE DU LANGAGE PASCAL

Le Pascal standard a été créé vers 1970 par Klaus Wirth chercheur suisse.
Par la suite, une nouvelle version du langage 'pascal ucsd' a été élaborée sous l'égide de l'université californienne de San Diego grace à Kenneth Bowles.
Le Pascal UCSD a été rapidement opérationnel sur les micros ordinateurs APPLE II et IBM PC
Son adaptation sur micro-ordinateurs THOMSON TO est restée confidentielle .
Son apprentissage sur TO ouvre des portes à une très bonne approche de Turbo-Pascal,voire Delphi.

LIVRES :

Comme livres on peut conseiller :
- initiation au pascal Editions Radio par jc Guillemot pour les débutants.
- la programmation en pascal éditions interéditions par Peter Grogono qui est l'ouvrage de référence par excellence.
- le langage Pascal UCSD editions technology resources par jc Grattery ,petit recueil pour une utilisation sur PC
- le guide du Pascal éditions Sybex par jacques Tiberghien ,ouvrage totalement exhaustif de tout le langage.
- Topiques pascal par John Colibri éditions memodynes ,énorme volume de 850 pages qui décortique le PASCAL UCSD en utilisation sur Apple II (mais bien utile aussi pour TO).Disponible auprès de l'Institut Pascal 26 rue Lamartine 75009 PARIS
- Pascal par la pratique Pierre.Le Beux éditions Sybex (ouvrage assez axé sur les mathématiques)
- Algorithmique et programmation en Pascal par Patrick Cousot éditions ellipses (école polytechnique) en deux volumes pour les utilisateurs sur MACINTOSH ..
- De nombreux ouvrages traitant de Turbo-Pascal,comme Turbo Pascal mode d'emploi de Douglas Stivison (Sybex) peuvent être aussi consultés car ils donnent en exemple des méthodes de programmation adaptables au PASCAL UCSD.
- Les anciens numéros de Teo n° 7, 8, 9,10,11 (années1986 et 1987) pour quelques informationss supplémentaires sur le langage dans l' environnement THOMSON.

II- MISE EN SERVICE

Le système PASCAL UCSD adapté pour THOMSON fonctionne sur TO9+ TO9 TO8 TO8D et peut-être sur TO7 70 avec lecteur de disquettes.
Il est aussi utilisable avec l'émulateur TEO sous MSDOS (choisir option mode 80 colonnes au démarrage de TEO sous MSDOS)

U.C.S.D. Pascal is a TM of the Regents
of the University of California

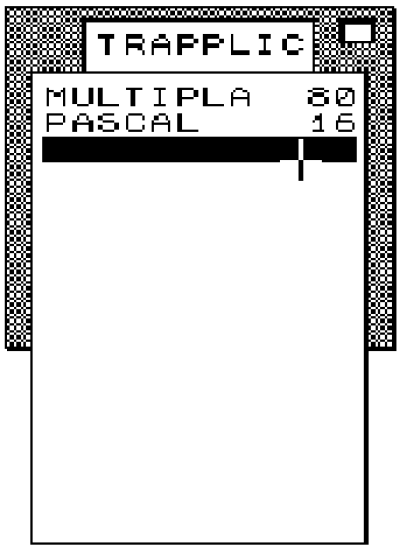
p-System is a trademark of
SofTech Microsystems, Inc.

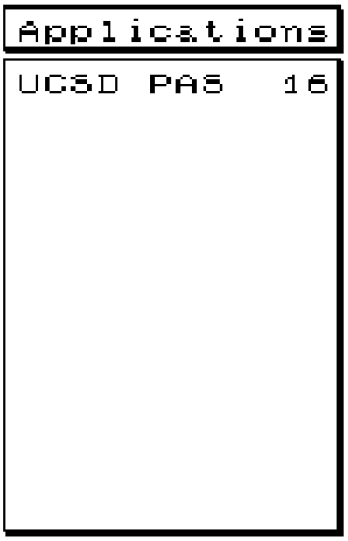
Converted by BUS groupe G.S.I. 1985

(c) copyright THOMSON 1986

Les disquettes contenant les fichiers PASCAL n'étant pas compatibles avec le DOS THOMSON ,elles doivent être utilisées et formatées exclusivement pour le PASCAL UCSD (un peu comme le logiciel LE JOURNALISTE).

On accède au langage en insérant dans le lecteur 0 ,une disquette (au DOS THOMSON) contenant le fichier de lancement de l'application PASCAL CHG
Aprés avoir validé (4) APPEL DE PROGRAMMES au menu du TO ,on valide Pascal dans la fenêtre de gauche avec la souris ou le crayon optique.
Puis, on doit retirer la disquette contenant le fichier de lancement pour la remplacer par la disquette système PASCAL (non compatible DOS Thomson) avant de valider dans la fenêtre de droite UCSD PASCAL .





Il faut procéder à la même manipulation en émulation avec TEO.

Les faces 0 et 2 des lecteurs de disquettes sont gérées mais pas les faces 1 et 3

Une fois sous PASCAL UCSD ,il faudra,autant que possible, laisser dans le lecteur 0 cette disquette de démarrage du PASCAL UCSD , nommée disquette racine (ROOT) et qui doit contenir au minimum les 4 fichiers suivants :

SYSTEM.BOOT
SYSTEM.MAP
SYSTEM.PASCAL
SYSTEM.MISCINFO

Les autres fichiers du système seront indispensables par la suite mais ils peuvent se trouver indifféremment sur la disquette de démarrage ou bien sur une autre disquette dans le second lecteur ou même sur le disque virtuel. (RAM dsk).
A ce propos pour utiliser un RAM disk il faut d'abord le selectionner dans le menu REGLAGES et PREFERENCES du TO et l'initialiser sous PASCAL UCSD avec l utilitaire CONFIG

Ces fichiers systèmes sont

SYSTEM.EDITOR (indispensable pour passer en édition)
SYSTEM.FILER (indispensable pour gérer les fichiers)
SYSTEM.COMPILER (indispensable pour compiler le texte d'un programme source Pascal)
SYSTEM.ASSMBLER (indispensable pour assembler un programme en Assembleur)
SYSTEM.LINKER (indispensable pour chainer divers programmes entre eux)
SYSTEM.SYNTAX (indispensable pour avoir le libellé des erreurs)
SYSTEM.LIBRARY (indispensable si on veut disposer des extensions du langage)

6809.OPCODES et 6809.ERRORS (pour les mnémoniques du micro-processeur)

Par exemple, dés que l'on passe , en EDITION par la touche E(dit
Le système va rechercher le fichier SYSTEM.EDITOR, jusqu'à le trouver, d'abord sur le lecteur principal,puis sur le second lecteur et enfin sur le disque virtuel si celui-ci existe.

Mais attention ,quand on quittera l'éditeur le système vous redemandera d'insérez ,si elle n'y est plus, la disquette racine de démarrage (root) dans le lecteur principal.

D'autres fichiers de type .CODE, comme par exemple CONFIG.CODE ou LIBRARY.CODE ,figurent sur la disquette du pack système PASCAL ;ce sont des utilitaires destinés à certaines taches.

L' écran de présentation

En haut de l'écran, apparait un bandeau,en deux parties, avec des lettres-clés qui correspondent à des commandes .
La touche ? permet de visualiser la partie 2 du bandeau.

- Options du bandeau de commandes

E(dit : permet de rentrer des programmes en étant en mode édition.

R(un : permet l'éxécution du programme présent en mémoire.

Tout programme en Pascal ,contrairement au Basic, existera sous deux formes
La forme texte et la forme code.
Quand vous écrivez un programme source dans l'éditeur vous utiliserez les mots réservés ,la syntaxe et les fonctions du Pascal UCSD
Ensuite,une fois le texte du programme saisi, vous devrez le compiler avant de pouvoir le lancer par RUN
La phase de compilation traduit le langage Pascal (texte) en codes compréhensibles par le micro-ordinateur.
Si vous n'avez pas commis d'erreurs dans votre programme , le compilateur affichera zéro errors et pourra génèrer un programme code éxecutable.
Par la suite ,c'est toujours le programme code qui sera éxécuté.
Le programme texte, à la limite , pourra être éffacé sauf si vous voulez le modifier pour le compiler une nouvelle fois (conseillé).

F(iler donne accés à la gestion des fichiers sur disquettes en ouvrant un autre bandeau.

X(ecute permet l'éxécution d'un programme code quelconque situé dans n'importe quel lecteur (il vous suffit d'indiquer le chemin complet d'accès au programme code).

R(un lance l'éxécution du programme de travail situé sur la disquette de démarrage (ce programme est toujours nommé SYSTEM.WRK.TEXT pour la version texte et

SYSTEM.WRK.CODE pour la version code) ou du dernier programme chargé en mémoire.

L(ink permet de fusionner des programmes codes entre eux (que leur source soit du Pascal ou de l'Assemnbleur).

C'est le cas si le programme principal contient des procédures ou des fonctions,définies dans des UNITs écrites en PASCAL et déjà compilés ou dans des modules écrits en ASSEMBLEUR.Il faudra utiliser cet éditeur de liens.

Si ces units ou modules ont été rangés ,auparavant, dans SYSTEM.LIBRARY les liaisons se feront de manière automatique.
Sinon,il faudra manuellement saisir le noms des programmes à relier par L(ink .

H(alt permet de quitter le PASCAL pour revenir au menu du TO8 / TO9 /TO9+

PREMIER PROGRAMME

Avant d'aller plus loin ,il peut sembler utile de saisir un premier programme pour vous familiariser avec le PASCAL UCSD.

Aprés avoir appelé l éditeur, par la lettre E du bandeau ,un message d'invite vous demande de saisir le nom d'un fichier texte à charger en mémoire ou d'appuyer sur ENT s'il s'agit de l'écriture d'un nouveau programme (ce qui est notre cas)

Un nouveau bandeau apparait .Il concerne toutes les commandes de l'éditeur.

Pour écrire correctement le texte du programme il faut utiliser les différentes commandes de l'éditeur.Ces commandes sont un peu rébarbatives au départ.

Ainsi pour démarrer la saisie du texte du programme il faudra en premier lieu activer la commande
I(nsert,
puis entrer le texte du programme classique ci-dessous en vous servant de la touche ENT pour passer à la ligne.

```
PROGRAM AFFICHE_UN_MESSAGE;

 VAR
    BIENVENUE:STRING ;
  BEGIN
BIENVENUE:=' Bonjour les Thomsonistes  ' ;
  WRITE(BIENVENUE); (* écriture à l'écran *)
END.
```

Ce petit programme comporte 7 lignes, non numérotées, que le système numérotera de 0 a 6, essentiellement pour vous indiquer où se situent les éventuelles erreurs (les lignes sans texte ne sont pas numérotées.)
Vous remarquerez que le programme est déclaré par le mot-réservé PROGRAM.
Chaque instruction est terminée impérativement par un point-virgule
Les commentaires sont encadrés par (* et *).
La fin absolue du prrogramme est indiquée par END suivi d'un point.

Si vous avez commis des erreurs de frappe ,pour effectuer les corrections quittez le mode insertion par CTRL-C et validez la commande D(elete .
Dés que vous aurez terminé votre saisie,en faisant très attention à la ponctuation,sortez du mode insertion en appuyant sur CTRL-C puis quittez l'édition par Q(uit.

Un choix de 4 options vous sera alors proposé et dans notre exemple vous choisirez U(pdate qui signifie mise à jour.
Cette commande a pour effet d'enregistrer la dernière version du texte de votre programme Pascal sur la disquette racine dans le lecteur principal.
Le programme sera automatiquement nommé SYSTEM.WRK.TEXT (fichier de travail texte).

Ayant quitté l'éditeur vous retrouverez donc le bandeau initial et vous lancerez la compilation de votre programme par la commande C(ompile.
Le compilateur va convertir le programme source Pascal en un programme code éxécutable (si il ne signale pas d'erreur).
Puis le programme code sera automatiquement sauvegardé sur disquette par le système avec pour nom SYSTEM.WRK.CODE (version executable du fichier de travail).

Il ne vous reste plus qu'à lancer l' exécution de ce programme par R(un.

Par la suite il vous est vivement conseillé de conserver ce programme dans vos archives , en lui donnant un nom autre que SYSTEM.WRK.CODE sinon il sera écrasé par tout nouveau fichier de travail (SYSTEM.WRK.TEXT ou SYSTEM.WRK.CODE) .

Pour cela appelez le gestionnaire de fichiers par F(iler
puis validez l'option S(ave et donnez par exemple le nom BONJOUR à ce court programme
validez par ENT et un message vous confirmera que SYSTEM.WRK.TEXT est sauvegardé avec le nom BONJOUR.TEXT quant à SYSTEM.WRK.CODE il est enregistré sous le nom BONJOUR.CODE.
Vous pouvez alors effacer par la commande N(ew la version texte et la version code du fichier de travail ,si vous voulez retourner en édition pour saisir un nouveau programme .

COMMENTAIRES SUR CE PROGRAMME

En Pascal ,il est important de savoir que la notion de lignes n'a aucune incidence ;c'est vous qui disposez votre programme comme vous l'entendez, pour qu'il soit le plus lisible possible.
Pour séparer chaque instruction on utilise le point-virgule.On peut donc avoir une ou plusieurs instructions sur une seule ligne.
L'en- tête commence toujours par l'instruction PROGRAM suivi d'un nom
(ici AFFICHE_UN_MESSAGE)
Il ne faut pas laisser d'espace dans un nom de programme mais utiliser à la place le caractère _
Dans la partie déclaration, nous avons déclaré , par VAR la variable BIENVENUE comme étant de type chaine de caractères (STRING).
BEGIN et END ne sont pas des instructions ;ce sont des mots réservés du pascal qui agissent comme des parenthèses et servent à encadrer une suite d'instructions.
Dans tout programme ,aussi simple soit-il il y a donc toujours ,au moins un BEGIN et un END.

exemple:

```
PROGRAM NE_FAIT_RIEN;
BEGIN
END.
```

Ce programme tout à fait correct ne fait rien !!!!

Reprenons l'analyse de notre premier programme
On affecte la chaine de caractères ' Bonjour lesThomsonistes ' à la variable BIENVENUE par : = (et non pas =)
:= équivaut à l'instruction LET du Basic et sert à affecter des valeurs.

Ensuite,on affiche la variable BIENVENUE à l'écran en utilisant l'instruction WRITE .

La fin absolue du programme est indiquée par

```
END.
```

Attention !
comme il faut encadrer une chaine alphanumerique par des ' cela vous obligera à les doubler si votre chaine en comporte elle-même .
exemple ' le bonjour de sophie ' et ' le bonjour d'' alfred '

En utilisant WRITE il n'y a pas de saut de ligne pour l'affichage suivant ; sinon utiliser WRITELN et la prochaine écriture se fera à la ligne en dessous.
A noter qu'on peut mettre des commentaires dans le programme en utilisant .

```
 soit
   (*  CECI EST UN COMMENTAIRE *)
soit
    {ceci est un commentaire }  (les accolades sont moins employées )
```

La phase de compilation ignore ces commentaires.

DIVERSES REMARQUES

On peut saisir son programme indifféremment en majuscules ou minuscules pour ce qui concerne les instructions, les noms des variables ou les fonctions ...
Le compilateur ne fera pas la différence .
Pour la version Thomson du PASCAL UCSD, le compilateur ne prend en compte que les huit premiers caractères des noms de variables, de fonctions, ou de procédures.
(non compris le caractère _)
Chaque variable devra avoir un nom différent (sinon erreur 101 à la compilation)
attention car sur TO une erreur est générée si un nom de variable est identique à l'en-tête du nom du programme .

Complétons notre approche par un petit programme de calcul.

```
PROGRAM ADDITION;
 VAR
PREM_NB,DEUX_NB,SOMME:INTEGER;

  BEGIN
WRITELN('ce programme aditionne deux entiers');
(WRITELN('TAPER LE PREMIER NOMBRE');
READLN(PREM_NB);
WRITELN('TAPER LE SECOND NOMBRE');
READLN(DEUX_NB);
 SOMME:=PREM_NB+DEUX_NB;
WRITELN('LA SOMME VAUT ',SOMME)
 END.
```

(Integer signifie variable entière numérique comprise entre -32767 et +32767)

Le point-virgule n'apparait pas à la fin de l'avant dernière ligne car comme on vous l'avez appris END n'est pas une instruction mais une sorte de parenthèse donc il n'est pas nécessaire de mettre un point-virgule entre WRITELN ('LA SOMME VAUT',SOMME) et END . .

Pour avoir des résultats avec des nombres décimaux ,on pourra utiliser les réels en simple précision (REAL)
Le Pascal UCSD peut aussi gérer des nombres comportant jusqu'à 36 chiffres : les entiers longs.

Il faut ,simplement, les déclarer par
INTEGER[n] où n peut varier de 1 à 36

Dans le cas des réels on peut formater la présentation du résultat
exemple :

écrire WRITE('somme vaut',SOMME:10:2);
L'affichage occupera 10 caractères, signe compris avec 2 chiffres significatifs après la virgule.
Cela est annoncé en théorie, mais en pratique nous avons constaté des bugs à l'affichage pour le formatage des réels sur THOMSON.

INDEX GENERAL DU PASCAL UCSD

I- BANDEAU PRINCIPAL

Sur la première ligne de l'écran on passe alternativement à chacune des deux parties du bandeau de commande par la touche [?]
Le système démarre automatiquement en mode 80 colonnes par défaut.

En mode 80 colonnes vous voyez l'affichage de:
E(dit,R(un ,F(ile, C(omp, L(ink,X(ecute,A(ssem,D(ebug,U(ser restart,
I(nit,H(alt

On peut activer le mode 40 colonnes en lançant l'utilitaire CONFIG .
Dans ce cas l'affichage sera uniquement composé de la première lettre de chaque commande du bandeau
E,R,F,C,L,X,A,D,U,I,H
Quelle que soit la portion du bandeau visible sur l'écran l'appui sur la touche correspondant à une lettre-clé d'une fonction ,active la fonction.

II- FONCTION HALT

Elle provoque le retour au système d'exploitation du micro-ordinateur .On peut ,alors , revenir directement au Pascal en tapant l'option 6 du menu du TO8 si la disquette de démarrage est restée dans le lecteur principal.

III- FONCTION RUN

Elle charge en mémoire le fichier de travail (SYSTEM.WRK.) et lance son execution.Si le fichier de travail est encore a l'état de texte, il est d'abord compilé (voir compilateur) puis immédiatement executé.
Lors de la premiére éxécution, RUN ouvre,si nécessaire, d'autres fichiers situés dans des bibliothèques,et crée automatiquement ou non les différents liens avec le programme principal.

IV- FONCTION EXECUTE

Cette commande permet l'éxécution d'un programme code déja compilé et présent sur une disquette .

La commande X(excute) affiche l'invite ci-dessous
EXECUTE WHAT FILE?
vous devrez indiquer le descriptif et le chemin complet du fichier à éxécuter.
exemple:
VOL15:PROG3 lancera l'exécution du programme PROG3.CODE situé sur la disquette ayant pour nom VOL15 (cette disquette pourra se trouver dans n'importe quel lecteur.)

Pour suspendre l'éxecution du programme taper CTRL-B (BREAK).pour la reprendre,refaire CTRL-B
CTRL-R --> provoque un arrêt temporaire de quelques secondes
Pour suspendre l'affichage à l'écran -> CTRL-F (FLUSH).puis reprise avec CTRL-F
CTRL-W provoque une sortie du programme avec un retour au bandeau de commandes du système Pascal UCSD.

MESSAGES D'ERREURS A L'EXECUTION
(ces messages ne sont à confondre ni avec les messages d'erreurs à la compilation, ni avec les erreurs d'entrées/sorties --> voir COMPILATION

0 ERREUR SYSTEME FATALE (REDEMARRAGE A FROID)
1 INDEX INVALIDE HORS LIMITE
2 PAS DE SEGMENT
3 PROCEDURE ABSENTE
4 DEBORDEMENT DE PILE
5 DEBORDEMENT D'ENTIER
6 DIVISION PAR ZERO
7 REFERENCE MEMOIRE INVALIDE
8 ARRET DU FAIT DE L'UTILISATEUR
9 ERREUR SYSTEME E/S FATALE (REDEMARRAGE A FROID)
10 ERREUR D'UTILISATION EN E/S
11 INSTRUCTION NON IMPLEMENTEE
12 ERREUR EN OPERATION FLOTTANTE
13 CHAINE TROP LONGUE
14 POINT D'ARRET PAR HALT
15 MAUVAIS BLOC

En général ces erreurs ont pour effet de ré-initialiser le système .
Les erreurs fatales causent un redemarrage a froid.

V- FONCTION U(SER RESTART

Elle provoque une nouvelle éxécution du dernier programme code exécuté
(cela évite d'avoir à retaper son nom)

VI- FONCTION INITIALIZE

Cette commande n'est pas une commande de formatage.

Elle est peu documentée et,a priori ne sert qu'à repasser au bandeau de commandes pour reprendre la main lors d'une éxécution d'un programme posant problème.
Egalement,lors de la création d'une nouvelle bibliothèque SYSTEM.LIBRARY par l'utilitire LIBRARIAN quand vous avez terminé, et que vous revenez au menu il est recommandé de faire I(nitialize

VII- FONCTION DEBUG

Appelle un programme de mise au point mais n'est disponible que pour certaines versions UCSD.Pour la version THOMSON cette commande est inactive et il peut même y avoir plantage en l'activant dans certains cas .

VIII- FONCTION ASSEMBLE

Il est possible d'écrire des modules source Assembleur dans l'éditeur en utilisant les mnémoniques du micro-processeur 6809.

Une fois le programme-source saisi ,on devra l'assembler par la commande
A(ssemble .
Le module code ainsi créé pourra être appelé à partir d'un programme écrit en Pascal en utilisant la directive EXTERNAL.

exemple:

```
PROCEDURE JOYSTK(S:INTEGER; D:BOOLEAN);EXTERNAL;
```

JOYSTK étant le nom d'une procédure déclarée dans un programme assembleur.
S et D sont deux paramètres transmis à ce module.

(voir des exemples dans magazine TEO N° 7 OCT 1986 et dans les bulletins 14 et 16 de CONTACTHOMS)

IX- FONCTION LINK

Le compilateur effectue automatiquement la liaison entre le programme principal et les unités si celles-ci sont situées dans SYSTEM.LIBRARY.
Mais vous pouvez créer vous-même vos propres unités, soit en Pascal soit en Assembleur, et vous n'êtes pas forcés de les ranger dans SYSTEM.LIBRARY.
Dans ce cas,les liaisons devront s'effectuer manuellement à l'aide de la commande L(ink
aprés la compilation de votre programme principal..

Tapez la lettre-clé L
A la question : HOST FILE? entrez le nom du programme principal.

Le linker ,ouvre alors le fichier contenant le programme principal,vérifie qu'il s'agit bien d'un fichier code et qu'il contient une directive d'appel à l'éditeur de liens (VOIR COMPILATION)
Il vous réclame ensuite le nom de la première librairie ,puis de la seconde etc.là où se trouvent les différentes unités que vous voulez relier à votre programme principal.
Quand vous avez appelé toutes les librairies tapez ENT
Le linker vérifie que chaque fichier appelé a bien une structure de librairie (voir paragraphe consacré aux unités et aux librairies)

Puis, en fin d'opération ,la question MAP FILE ? vous demande si vous désirez avoir un fichier journal sur les tailles et emplacements des segments des unités (répondre par Yes ou No)
L'éditeur de liens (linker) effectue alors le travail,en affichant à l'écran les differentes étapes.
Puis il vous demande un nom pour ce nouveau fichier-programme code qui regroupe donc le code du programme principal et celui des unités.
Appuyer sur ENT si vous voulez écraser la version du programme principal avant l'edition des liens (sinon entrez un nouveau nom en mentionnant le suffixe .CODE.)

Dès le retour au bandeau de commandes , on pourra lancer l'éxecution du programme code avec la touche X : X(cute).

X- GESTIONNAIRE DE FICHIERS (le filer)

Pour accèder au gestionnaire de fichiers tapez la lettre -clé F
On accède alors à un sous-bandeau de commandes.
et alternativement par la touche ? aux deux parties de ce sous-bandeau.

En mode 80 colonnes vous voyez l'affichage de
G(et,S(ave,W(hat, N(ew, L(dir,R(em,C(hng,T(rans,D(ate,Q(uit
Et sur la seconde moitié du bandeau l'affichage de:
B(ad blocks,M(ake,K(runch etc....

* commande Q(uit
elle permet de quitter le gestionnaire de fichiers avec retour au bandeau principal des commandes

* commande V(olumes
elle permet de visualiser l'ensemble des péripheriques connectés et des volumes disponibles.
Tous les péripheriques ,sauf les lecteurs de disquettes ,sont des volumes.en revanche les disquettes sont des volumes.

Pour un système hardware complet on aura:
0_ non utilisé
1_ CONSOLE:(avec echo)
2_ SYSTERM:(sans echo)

3_ GRAPHIC:(terminal graphique)
4_ unité lecteur principal de disquettes
5_ unité second lecteur de disquettes
6_ PRINTER: (imprimante)
7_ REMIN: (interface MODEM entrée)
8_ REMOUT: (interface MODEM sortie)
9_disque virtuel (RAM DISK)
10 à 12 _ autres unités de disquettes (ne concerne pas THOMSON)

La spécification d'un volume est désigné habituellement par un nom de 7 caractères maxi suivi de deux points
exemple:
DISK39: désigne la disquette (volume) nommée DISK39 et activera la recherche du lecteur où se trouve cette disquette .
On peut aussi utiliser # +numéro d'unité.
Dans ce cas #4 (ou #4:)désignera la disquette qui se trouve dans le lecteur principal.

remarques sur l'affichage de la fonction V(olumes

Un # signifie que le nom situé sur la même ligne,désigne un volume structuré (par exemple une disquette avec ses secteurs)

COMMANDE P(refix

Le volume racine (root) désigne le nom du volume (nom de la disquette) qui a permis le demarrage du système .On peut aussi la désigner par un * pour éviter la saisie complète du nom

Le volume préfixe designe le nom du volume utilisé par defaut .
Au départ le volume préfixe (ou courant) est identique au volume racine.
on peut le désigner ,en abrégé, par (:) .(voir mode abrégé)
N'importe quelle disquette peut être désignée comme volume courant,en utilisant la commande P(refix.
puis en tapant son nom suivi de : (ex DISK40:)
ou en indiquant un lecteur précis (par exemple #5 désignera le second lecteur, à condition qu'à cet instant il soit vide donc bizarrement sans disquette à l'intérieur.)

COMMANDE E(xtended directory

Elle affiche le contenu du repertoire d'un volume de manière detaillèe.
a la question DIR LISTING OF WHAT VOL ?
entrez le nom complet de la disquette concernée
exemple : DISK28 (+ ENT) permettra d'avoir son répertoire si bien sûr la disquette se trouve dans l'un des deux lecteurs.
En mode abrégé si on veut le répertoire de la disquette de démarrage il suffira de taper le signe *

De la même façon ,en abrégé ,on obtiendra le catalogue de la disquette courante par :
Autre façon de procéder: en tapant #5: on aura le répertoire de n'importe quelle disquette se trouvant dans le lecteur secondaire.
Si le message d'erreur
' NO SUCH VOL ON LINE' apparait c'est que la disquette appelée ne se trouve dans aucun des lecteurs .

Voici comment se présente ce répertoire à l'écran.

en haut on aura le nom de la disquette en colonne 1 le nom des fichiers .
en colonne 2 le nombre de blocs occupés par le fichier.en colonne 3 la date de mise à jour du fichier
en colonne 4 l'emplacement du debut du fichier.en colonne 5 le nombre d'octets utilisés dans le dernier bloc occupé
en colonne 6 le type du fichier text code ou data.la ligne du bas indique le nombre de fichiers listés par rapport au nombre total de fichiers .
Puis l'espace total occupé par les fichiers listés,ensuite l'espace inutilisé et enfin la grandeur du plus grand espace inutilisé (blocs consécutifs)
les blocs libres sont désignés par UNUSED.

Vous pouvez imprimer ce répertoire
Appelez la commande E(xtended dir puis si l'imprimante est en service
Entrez --> NOM DE VOTRE DISQUETTE,PRINTER: et validez par ENT
OU NOM DE LA DISQUETTE:,#6
Mais on peut avoir aussi se permettre des écritures plus complexes
Ainsi --> DISK30:,DISK31:REPERT.TEXT
aura pour effet de copier le repertoire de la disquette DSK30,dans un fichier texte nommé REPERT qui sera sauvegardé sur la disquette DSK31.

COMMANDE L(IST DIRECTORY.
Elle donne une liste du repertoire mais de manière plus simplifiée.
Elle accepte,tout comme E(xtended directory les critères de selection en mode abrégé qui sont présentés ci-dessous :
Il peut être intéressant de ne pas avoir le listage complet de tous les fichiers de la disquette.
Grace aux caractères d'abréviation du Filer on peut créer des filtres
Ainsi :
B= désigne tous les fichiers commençant par B
=S tous les fichiers se terminant par S
P=S tous les fichiers commençant par P et finissant par S
=.TEXT tous les fichiers de type TEXT
R=.CODE tous les fichiers commençant par R et de type code

à la place de = on peut utiliser ? mais dans ce cas il faudra confirmer la commande(ceci est surtout valable pour les commandes d effacement).

COMMANDE GET

Elle permet de sélectionner un fichier .TEXT pour qu'il devienne le fichier de travail (par exemple quand on veut reprendre un programme pour le modifier).
A la question GET WHAT FILE ? entrez le nom du fichier .
Si ce fichier existe il est chargé en mémoire.
Si une version code existe elle est également chargée
le message 'nom.text & nom.code loaded' indique le bon deroulement du chargement.
Les deux versions du programme seront alors nommées SYSTEM.WRK.TEXT et SYSTEM.WRK.CODE pour redevenir les fichiers de travail.
Bien entendu ,les versions originales restent sur la disquette avec leur nom d'origine.
Mais si un ancien fichier de travail se trouve encore sur la disquette,au moment ou vous appelez par G(et un nouveau fichier, un message d'avertissement vous alerte :
THROW AWAY CURRENT WORKFILE ? (y/n) vous demandant si vous voulez bien l'écraser.

LA COMMANDE WHAT

Elle renseigne justement sur la présence ou l'absence d'un fichier de travail sur la disquette en donnant 4 types de réponses (plus ou moins claires à vrai dire)

no workfile :
il n'y a pas de fichier de travail (code ou text)
not named :
le volume racine contient un fichier de travail qui n'a pas encore été sauvegardé sous un nom particulier.
workfile is nom volume:nom fichier:
le fichier designé était le fichier de travail mais il n'y a plus ni SYSTEM.WRK.TEXT NI SYSTEM.WRK.CODE sur le volume racine.
workfile is nom volume:nom fichier (not saved)
SYSTEM.WRK est sur le volume racine et provient du fichier'nom de fichier' mais les éventuelles modifications n'ont pas été sauvegardées

LA COMMANDE NEW

Elle efface les fichiers de travail présents sur la disquette racine.

LA COMMANDE SAVE

Elle permet à tout moment de sauvegarder les contenus de SYSTEM.WRK.TEXT et éventuellement SYSTEM.WRK.CODE, quand une compilation du fichier de travail texte a eu lieu.
Pour obtenir différentes versions de sauvegarde il faudra donner des noms différents lors des appels successifs de S(ave.

COMMANDE REMOVE

Elle permet de détruire un fichier.
Elle doit être confirmée,pour plus de sécurité par Yes à la question UPDATE DIRECTORY ?
Cette commande accepte aussi les abréviations vues plus haut comme ? ou =
exemple : REMOVE WHAT FILE ? --> TEST ? provoquera la destruction de TEST.TEXT et TEST.CODE du volume prefixe mais en demandant une double confirmation.
On peut également effacer tout un groupe de fichiers
par exemple
MEMO.TEXT,JEU.CODE,NOTE.DATA pour détruire en une fois ces trois fichiers

LA COMMANDE CHANGE

Elle permet de renommer un fichier ou une disquette
Il faut répondre à l'invite
CHANGE WHAT FILE ? nom de fichier
puis
CHANGE TO WHAT ? nouveau nom

Autre possibilité
En repondant par exemple :
VOL5:MEMO.TEXT,NOTE.TEXT on renommera le fichier MEMO DU VOL5 en NOTE

Cette commande permet aussi de changer le nom d'une disquette
exemple
en entrant NOIR:,BLANC:
on donnera un nouveau nom à la disquette NOIR pour l'appeler BLANC
CHANGE accepte les abrégés comme ? $ et =
exemple pour renommer les deux fichiers
NOTE.TEXT ET NOTE.CODE en MEMO.TEXT ET MEMO.CODE
Taper plus simplement NOTE?,MEMO?

LA COMMANDE TRANSFER

Elle permet de copier un fichier ou un groupe de fichiers vers une autre disquette.
Si vous n'avez qu'un seul lecteur mettez la disquette source , choisissez T(ransfer puis attendez l'ordre d'introduction de la disquette destination.

transfer what file ?
VOL4:DEMO
to where? ---> VOL2:NOTE
copiera le fichier DEMO du VOL4 sur le VOL2 en le renommant NOTE
autre possibilité
transfer what file ? LETTRE.TEXT,VOL3:ECRIT.TEXT
évite une seconde question en effectuant directement la copie sous un nouveau nom.

autre syntaxe:

transfer what file? VOL4:JEU.TEXT,VOL2:GAME.TEXT[10]
pour copier JEU du VOL4 sur VOL2 en le renommant GAME (GAME sera placé dans la première zone contenant au moins 10 blocs libres consécutifs)
autre variante
transfer what file ? *DEMO.TEXT,:$
copiera le fichier DEMO du volume racine(*) vers le volume prefixe en lui conservant le même nom.
on notera l'emploi des abrégés * : et $
attention ne pas oublier le $ car sinon vous risqueriez de detruire l'integralité du volume prefixe !

Autre exemple d'emploi d'abréviations avec T(ransfer
TRANSFER WHAT FILE ?
réponse ----> MEMO?,VOL3:$
réalisera une copie de MEMO.CODE et MEMO.TEXT situés sur le volume par défaut vers la disquette VOL3 en gardant pour noms MEMO.TEXT ET MEMO.CODE
Le point d'interrogation active la demande de confirmation par Yes pour chaque operation.

Hélas,si vous n'avez qu'un seul lecteur vous ne pourrez pas utiliser les abreviations avec la commande T(ransfer.

METHODE POUR COPIER L'ENSEMBLE D'UNE DISQUETTE

Avec la commande T(ransfer vous pouvez réaliser la sauvegarde d'un disque complet
TRANSFER WHAT FILE ?
Réponse --> VOL1:,VOL6:
affichage de TRANSFER N BLOCKS?
répondre y puis message: --> destroy VOL6 ? repondre Y
on obtiendra un duplicata de la disquette VOL1 (le contenu ,le nom et le repertoire de la disquette destination VOL6 sont, bien entendu, écrasés.).

On peut aussi, déplacer un fichier vers une autre zone de la disquette

TRANSFER WHAT FILE ?
-->ESSAI.TEXT,$[8] -> ici on notera que les disquettes source et destination sont identiques ce qui aura pour effet de copier ESSAI dans une nouvelle zone et de supprimer l'ancienne.
De plus,ESSAI.TEXT sera copié dans la première zone de 8 blocs libres si bien entendu sa taille est inférieure ou égale a 8 .
Si le nombre entre crochets n'est pas mentionné, le transfert se fait vers la plus grande zone libre de la disquette.

Pour obtenir une duplication d'un même fichier sur une disquette il faut lui donner un nouveau nom.
exemple --> ESSAI1.TEXT,ESSAI2.TEXT

UTILISATION DE LA COMMANDE TRANSFER POUR IMPRIMER

Pour lister un programme sur imprimante la façon la plus simple est d'employer la commande T(ransfer.
transfer what file? VOL1:MEMO.TEXT,PRINTER:
permet la sortie du fichier MEMO.TEXT sur imprimante.
Comme tout périphérique est aussi un volume,on pourra,de la sorte, éditer un fichier .TEXT à l'écran par

transfer what file? VOL1:MEMO.TEXT,CONSOLE:
ou VOL1:MEMO.TEXT,#1:

COMMANDE KRUNCH

En Pascal UCSD, les fichiers sont sauvegardés dans des emplacements comportants un nombre de blocs suffisants et contigus sur les secteurs de la disquette.
Ainsi même s'il reste 3 blocs non contigus ,et que le fichier occupe deux blocs ,on ne pourra pas le sauvegarder.
Il est donc utile, de temps en temps , de défragmenter les disquettes pour regrouper les espaces libres .

Si une commande E(xtended dir fait apparaitre beaucoup de blocs isoles inutilises (UNUSED) alors activez K(runch

A l'invite répondre à la premiere question en indiquant le nom de la disquette a défragmenter.
Une seconde question vous demande si vous voulez ranger les emplacements libres en fin de disque; si vous repondez N(on alors une troisième question vous demandera d'entrez le numèro du bloc ,à partir duquel vous voulez que commence l'espace libre sur la disquette.
Dans le cas du choix de la dernière option, les blocs non vides seront répartis avant et après la zone vide nouvellement désignée.
L'opération se termine avec l'apparition du message :
NOM VOLUME:CRUNCHED

LA COMMANDE BAD BLOCKS
Elle permet de vérifier ,si sur un volume ,des secteurs ne sont pas endommagés.
On pourra l'exécuter avant la défragmentation ou si on a un doute sur l'état d'une disquette.

première question de l'invite de la commande :
Rentrer le nom de la disquette
la seconde invite vous propose une vérification
Soit de tous les blocs ,si vous repondez Yes
Soit d'un nombre fixe de blocs a tester.

Au fur et à mesure de la vérification les mauvais blocs sont indiqués à l'utilisateur avec en fin le total des mauvais blocs et la liste des fichiers endommagés du fait de ces anomalies.

LA COMMANDE E(X)AMINE

Elle permet parfois de restaurer les mauvais blocs signalés par la commande précedente.
Tapez d'abord le nom de la disquette puis la limite inférieure et la limite superieure séparés par un trait d'union entre,pour indiquer l'amplitude de l'opération (exemple: 20-30 pour essayer de réparer les blocs 20 à 30)
Ensuite on vous demande une confirmation par FIX THEM ?
à chaque bloc traité,le système envoie un message qui sera soit
- irréparable (IT'S BAD) ou
- semble reparé(MAY BE OK)
Dans le cas d'un bloc irréparable,il vous sera demandé de marquer le bloc incriminé pour le rendre désormais indisponible en écriture sur la disquette.
Si le bloc marqué constituait une partie d'un fichier ce fichier disparaitra

LA COMMANDE MAKE

Elle est assez peu utilisée et sert a réserver de la place sur une disquette pour un usage ultérieur.
On entrera le nom d'un fichier virtuel, occupant la zone a réserver suivi entre crochets du nombre de blocs.
exemple : VOL4:PLACE[12]
réservera sur la disquette VOL4 une zone de 12 blocs identifiés par le nom PLACE.
Si vous omettez la taille entre crochets ,la réservation portera sur la plus grande taille disponible.
si vous mettez [*] la réservation occupera la moitie de la plus grande zone libre ou la totalité de la zone libre venant en second .

LA COMMANDE ZERO

Elle permet de nommer une disquette en initialisant son répertoire.
Auparavant toute disquette neuve qui aura été formatée sous Pascal UCSD avec l'utilitaire CONFIG aura pour nom FMT:

A la première question de l'invite
répondre FMT: (ou #4 pour désigner le lecteur principal)
Puis le système indique que le catalogue de FMT: sera détruit
(ce qui n'est pas génant puisque la disquette est vide de fichiers)
Le nombre total de blocs de la disquette est alors indiqué (640 blocs) et un nouveau nom est demandé à l'utilisateur.
Eventuellement on pourra conserver une trace de l'ancien répertoire d'une disquette renommée par la commande Z(ero en répondant y(es à la question
Duplicate Dir ?

LA COMMANDE DATE
Elle permet de rentrer la date du jour qui sera ensuite indiquée devant chaque fichier du répertoire créé ou modifié ce jour là.
Si le jour change mais le mois reste identique taper seulement le chiffre du nouveau jour puis ENT
si le jour et le mois changent mais pas l'année taper
par exemple 6-MAR
Le système n'accepte que l'orthographe en anglais pour les abréviations des mois.
Jan,Feb,May etc

XI- FONCTION EDIT

On accède à l'éditeur par la lettre -clé E .

Dès l'entrée en mode EDIT il y a chargement du fichier de travail en mémoire et son affichage à l'écran .
Ce fichier de travail existe si le fichier SYSTEM.WRK.TEXT est sur le volume racine ou si un autre fichier a été designé comme fichier de travail par la commande GET.
En cas d'absence de ce fichier (no workfile present) vous avez trois solutions
1_ taper un nom de fichier à éditer (sans spécifier le suffixe TEXT) et il se chargera en mémoire ,sauf si vous lisez le message NOT PRESENT.
2_ appuyer sur ENT et vous serez alors en mode de création d'un nouveau fichier sur une nouvelle page-écran vierge.
3_ appuyer sur ESC (CTRL [pour thomson) qui vous ramenera au bandeau de commandes .

LES TOUCHES UTILISABLES EN EDITION.

DEPLACEMENT DU CURSEUR :
Les touches fléchées gauche,droite,haut et bas deplacent le curseur .
La barre d'espace déplace le curseur vers la droite .
CTRL-H (backspace) déplace le curseur vers la gauche.
la touche ENT déplace le curseur jusqu'au début de la ligne suivante
Une tabulation est obtenue par CTRL-T .
Les touches fléchées sont paramétrables avec le pavé numérique
(ex 18 <- fait reculer le curseur de 18 caractères)

INDICATEUR DE DIRECTION (>)
L'action de ENT et de la barre d'espace peut être inversée selon le sens de l'indicateur de direction.
Celui-ci est placé en haut de l'écran à gauche du mot EDIT et il est par défaut dirigé vers l'avant (>)
Pour l'inverser,il faut se trouver en mode EDIT OU DELETE et appuyer sur les touches suivantes > < + ou -

AUTRES TOUCHES

La touche = (peu employée), amène le curseur au dernier endroit du texte modifié par REPLACE, ou selectionné par FIND ou inséré par INSERT (voir ces commandes)
La touche P (page) fait soit avancer le texte d'une page ou soit remonter d'une page selon le sens de l'indicateur de direction.
En tapant 4 suivi de P le texte se déplace de 4 pages-écran.

COMMANDE J(UMP

Elle permet de sauter rapidement à un endroit précis du texte .
Début (BEGINING), fin (END) OU position marquée (MARKER)voir détails à marker .On abandonne par CTRL [cette commande.

COMMANDE INSERT

Ce mode permet soit d'écrire un nouveau texte,soit d'ajouter ou d'inclure des lignes dans un texte éxistant.
La taille du texte dépend de la mémoire disponible et on peut connaitre la quantité de mémoire capacité restante par l'option E(nvironnement de la commande SET.
L'insertion débute à l'endroit où se trouve le curseur .
Pour corriger une faute de frappe en mode insertion on utilise la touche fléchée ARR ou CTRL-H quand on doit corriger dans la ligne courante si elle n'a pas été validée par ENT .
Si une correction est à faire sur une autre ligne il faut quitter le mode insertion et utiliser les autres commandes DELETE etc...
Sur Thomson la touche DEL n'a pas l'air de fonctionner en mode insertion
On quitte le mode insertion par (etx) soit CTRL-C sur Thomson et le texte est rangé dans une mémoire tampon.

Si on quitte le mode insertion par CTRL [le texte est perdu.

L'insertion gère une auto-indentation des lignes .Quand vous appuyez sur ENT le curseur passe à la ligne suivante, en venant se placer sous le premier caractère non blanc, de la ligne d'ou il vient.
On peut modifier manuellement l'auto-indentation des touches CNT-H,BARRE D'ESPACE etc....
Il est possible d'écrire du texte au delà de la limite droite de l'écran .
Si c'est le cas ,vous en êtes averti ,par un point d'exclamation (!) juste avant la marge droite (voir option SET)

Pour voir le reste d'une ligne hors écran ,positionner le curseur sur le dernier caractère visible à droite et repasser en mode insertion.

COMMANDE EXCHANGE

Ce mode permet de corriger des petites erreurs en remplaçant un caractère par un autre.Pour cela placer le curseur sur le caractère erroné, validez la touche X pour activer la commande EXCHANGE ,et tout nouveau caractère saisi remplacera l'ancien.

Si vous faites une erreur en corrigeant, revenez en arrière,les anciens caractères réapparaitront .Si vous quittez la fonction par CTRL-C toutes les modifications seront validés ;au contraire, par CTRL [(esc) rien ne sera pris en compte.
Le mode EXCHANGE se limite à une seule ligne à la fois,et n'accepte pas la frappe de ENT.

COMMANDE DELETE

Ce mode permet d'effacer du texte .Quand vous êtes en mode DELETE ,le déplacement du curseur gomme les caractères, mais ceux-ci réapparaitrons si vous revenez en arrière avec les touches fléchées.
Le déplacement du curseur s'opère soit par les touches fléchées, soit par CTRL H,la barre d'espace ou ENT .La frappe de ces touches peut être precedée d'un argument numerique: .(exemple : 4+ ENT efface 4 lignes.)
la flèche d'indication de direction est également prise en compte.
L'appui sur CTRL-C, valide l'effacement .
L'appui sur CTRL [ignore l'effacement.
Dans les deux cas ,le texte effacé sera rangé dans une mémoire tampon.
La relecture de cette mémoire tampon est possible par la fonction COPY (voir cette fonction)

COMMANDE COPY

Elle permet de copier le contenu de la mémoire -tampon à partir de la position du curseur par l'option B(uffer.
Rappelons que la mémoire- tampon contient le texte ,le plus récent, d'une insertion validee par CTRL-C ou d'un effacement valide par CTRL-C ou non validé par CTRL [.
l'option F(rom file permet de copier tout ou partie d'un fichier sur disquette à partir de la position courante du curseur.
Positionner le curseur à l'endroit, a partir duquel vous voulez inserez le texte.
L'option, entre crochets, qui peut figurer après le nom du fichier indique que seule la partie du fichier delimitée par les marques sera copiée.
exemple : VOL1:JEU[PART1,PART2] fournira la copie du fichier texte JEU de la disquette VOL1 pour la partie comprise entre les marques PART1 ET PART2
On peut avoir comme variante [,PART2] du debut jusqu'à PART2 ou [PART3,] de la marque PART3 à la fin du fichier.
Pour mettre des marques dans un fichier voir la fonction SET.
On quitte la fonction COPY par CTRL [(esc).

COMMANDE ADJUST

Elle permet de rectifier la disposition des lignes de programmes pour rendre la presentation plus lisible.
placer le curseur sur la ligne à ajuster et sélectionner A(djust.
En utilisant L la ligne est amenée contre la marge gauche,avec R la ligne est amenée contre la marge droite et avec C la ligne est centrée.

Ces ajustements peuvent être complétés par l'appui sur les touches fléchées haut et bas pour ré-aligner ensuite d'autres lignes sur le modèle.
Les touches flèchées gauche et droite déplacent la ligne d'une position vers la droite ou la gauche.
Mais on peut faire précèder l'appui sur l'une des quatre touches flèchées par un argument numérique pour étendre leur action sur plusieurs lignes a la fois.
(7 + appui sur touche fléchée -haut provoque l'alignement des sept lignes situees au-dessus de la ligne-modèle sur laquelle se trouve le curseur.

On quitte le mode adjust par CTRL-C

COMMANDE FIND

Cette commande permet de rechercher une ou plusieurs occurences de chaines de caractères dans un texte
F(ind opère a partir de la position du curseur,vers l'avant ou l'arrière selon l'indicateur de direction.
Il est possible d'entrer un paramètre n avant l'appui sur F(ind ;alors la recherche se poursuivra jusqu'à trouver un nombre n d'occurences , le curseur se positionnant sur la nième occurence(l'argument numérique apparaitra entre crochets dans le bandeau de la fonction) .
Si vous tapez / puis F(ind, la recherche ira jusqu'au bout et indiquera la dernière occurence trouvée.
Si vous tapez F(ind, sans préciser de paramètre, la recherche s'arrêtera sur la première occurence trouvée.

On vous demandera de spécifier la chaine cible (ou chaine recherchée) et vous devrez l' encadrer par un délimiteur
(exemple: /CLIENT/) les délimiteurs ne doivent être ni des lettres,ni des chiffres,ni un espace.
Selon que la chaine cible sera entrée en majuscules ou en minuscules la recherche s'arrêtera sur le mot en majuscules ou minuscules mais pas sur les deux .

OPTIONS POUR LA COMMANDE FIND
La commande opère en mode litteral L(it ou en mode T(oken.
Ces deux modes sont sélectionnables en bascule par appui sur la lettre-clé correspondante. .Si dans le bandeau apparait L(it c'est que vous êtes en mode T(oken (et réciproquement).
exemple:
En mode T(oken, si on recherche /PROGRAMME/ la recherche s'arrêtera sur PROGRAMME mais pas sur PROGRAMMEUR.
En mode L(it , si on recherche /PROGRAMME/ le système s'arrêtera sur PROGRAMME mais également sur PROGRAMMEUR.

La chaine cible (target) figurant entre les délimiteurs est conservée dans une zone cible-tampon.En poursuivant la recherche, inutile de retaper la chaine cible entrez simplement la lettre S (sans délimiteurs) qui signifie same.
Si la recherche n'aboutit pas vous aurez le message:
PATTERN NOT IN FILE , taper alors la barre d'espace pour continuer.

On retourne au menu général de l'édition par CTRL-C

COMMANDE REPLACE

Cette commande est complementaire de F(ind car elle permet à la fois recherche et substitution.
Vous devez entrer une chaine cible et une chaine de substitution
exemple: /SOMME//TOTAL/
On pourra remplacer une ou plusieurs occurences de chaines de caracteres dans un texte par la chaine de substitution ;cela grace à un argument numérique à donner avant d'activer R(eplace.
R(eplace opère à partir de la position du curseur,vers l'avant ou l'arrière selon l'indicateur de direction.
Si vous tapez / puis R(eplace le remplacement aura lieu jusqu'à la fin du texte.
La commande opère en mode littéral L(it ou en mode T(oken (voir FIND)
De plus on dispose de l'option V(erify, en tapant V, avant la cible ou avant la substitution qui oblige à confirmer chacune des substitutions.

Avec cette option le remplacement se fait en deux fois .Le curseur se place a la fin de la chaine en concordance avec la cible et on vous demande de taper R pour confirmer le remplacement.Si vous tapez espace le remplacement n'a pas lieu et le curseur se place sur la concordance suivante.

Pour éviter de taper plusieurs fois le nom de la cible (ou substitution) on peut utiliser l'option same (S sans delimiteurs),en réponse à l'invitation <TAR> <SUB>.
Le contenu des zones tampons cible et substitution, peut être visualisé par la commande SET (option :environnement).

Pour revenir au menu EDITt taper CTRL [

COMMANDE ZAP

Cette commande est dangereuse car elle detruit tout le texte compris entre le curseur et le premier caractère du texte le plus récemment trouvé ,insèré ou remplacé à l'aide des commandes FIND,INSERT ou REPLACE .
Néanmoins si la destruction concerne plus de 80 caracteres un message vous en avertit.

COMMANDE VERIFY

Elle n'apparait pas dans le menu EDIT mais on y accède par lettre-clé V
Elle provoque l'affichage du texte contenu dans la mémoire-tampon .

COMMANDE QUIT

Elle permet de quitter le mode édition pour retrouver le bandeau de commandes.
Auparavant,en quittant l'éditeur il est proposé 4 possibilités:

U --> UPDATE (mise à jour) qui sauvegardera automatiquement l'écriture du tampon texte sur la disquette racine en nommant le fichier SYSTEM.WRK.TEXT .
E --> EXIT provoque un retour au bandeau de commandes et ignore le tampon-texte.
R --> RETURN permet un retour au mode édition sans écriture sur disquette.
W --> WRITE permet la sauvegarde du tampon-texte , vers n'importe quelle disquette, avec un nom de fichier choisi par l'utilisateur

COMMANDE SET

Cette commande permet un contrôle des paramètres de l'éditeur et la pose de marques de repérages.
On abandonne la fonction SET par CTRL [.

a) OPTION MARKER
Placer en premier lieu le curseur à l'endroit où vous voulez inscrire une marque.
taper M et à la question:
SET WHAT MARKER ? entrez un nom ou des chiffres (8 maxi) puis ENT
Le texte est désormais marqué à cet endroit.
On peut mettre un maximum de 10 marques dans un texte .ces marques invisibles à l'écran seront affichées en utilisant l'option environnement de SET.
Pour déplacer une marque,déplacer le curseur et entrer à nouveau son nom lors de l'invitation SET WHAT MARKER?"
Si vous essayez de placer plus de 10 marques,la liste des marques s'affiche et on vous demande de taper le numéro de la marque à supprimer, puis le nom de la dernière marque .

b) OPTION ENVIRONNEMENT
Elle permet l' affichage d'un important tableau de paramètres et de renseignements concernant l'éditeur.
dans la première moitié du tableau s'affichent 7 paramètres
AUTO.INDENT gére l'auto-indentation
s'il est à true,le curseur s'aligne sur le premier caractère de la ligne précédente; sinon la ligne reprend à l'extrème gauche de l'écran après la frappe de ENT.
FILLING gère l'affichage du texte au bord droit de l'écran.
s'il est a false vous pouvez entrer du texte au-dela de la limite droite de l'ecran et un ! vous l'indique
s'il est a true ,quand un mot dépasse la marge droite il est automatiquement rejeté à la ligne suivante sans avoir a utiliser la touche ENT.

LEFT MARGIN détermine la marge gauche
(spécifier un nombre).Il ne fonctionne que si auto-indent=false et filling=true.
RIGHT MARGIN détermine la marge droite
(spécifier un nombre) ne fonctionne que si auto-indent=false et filling=true
PARA MARGIN détermine la position de la première ligne d'un paragraphe.
ne fonctionne que si auto-indent =false et filling=true
TOKEN DEF fixe le mode proposé par défaut (soit token soit litteral) pour les fonctions FIND et REPLACE.
il prend la valeur false ou true
COMMAND CH option non disponible sur THOMSON

Dans la deuxième moitié du tableau s'affichent divers renseignements.

BYTES : le nombre d'octets utilisés dans le tampon de texte et ceux qui sont encore disponibles est indiqué.
PATTERNS: affiche le contenu des dernières cibles et substitutions.
MARKERS : indique le nom de toutes les marques
DATES : indique la date de création du fichier et la date de sa dernière modification.

L'intèrêt des paramètres est de pouvoir utiliser l'éditeur à la fois comme éditeur de programmes pascal et comme traitement de texte.
-En mode 'programmation' il faut que auto-indent soit vrai (t) et filling faux (f)
-En mode 'traitement de texte' il faut que auto-indent soit faux (f) et filling vrai (t)"
En mode 'traitement de texte' , la commande MARGIN est directement accessible à partir de EDIT en tapant M .Cette commande permet de reformater un paragraphe et elle n'affecte que le paragraphe contenant le curseur

Pour quitter l'option environnement,et permettre la validation des modifications, utiliser la barre d'espace ou CTRL-C

XII- FONCTION COMPILATEUR

a) GENERALITES

Vous ne pouvez vous-même essayer de créer directement un fichier .CODE .
Vous devez écrire un fichier-source texte qui sera ensuite traduit en fichier code éxécutable par le compilateur .

En activant la fonction de compilation (appui sur C dans le bandeau principal) le systéme charge le fichier de travail SYSTEM.WRK.TEXT (ou éventuellement un autre fichier texte que vous aurez choisi avec Get) et le traduit en fichier CODE (compilation)

La compilation consiste à transcoder complètement un fichier source en langage pascal en un fichier de p-code .
Le p-code sera ensuite traduit en langage-machine (processeur 6809) par un interprèteur lors de l'exécution du programme;

Toute erreur détectée,lors de la compilation,en arrétera le déroulement et nécessitera une correction des erreurs par l'utilisateur.
Toutefois si le programme est correct sur le plan syntaxe ,il sera compilé et traduit en programme code sans pour autant être certain que son execution répondra aux attentes du programmeur (il pourra comporter des erreurs logiques ou des omissions) .
Le processus de compilation nécessite la présence dans l'un des lecteurs ou sur le disque virtuel du fichier SYSTEM.COMPILER .
Le fichier code SYSTEM.WRK.CODE, sera nommé puis sauvegardé automatiquement dans la plus grande zone inoccupée de la disquette racine.

Pour choisir un autre emplacement,en compilant un fichier autre que SYSTEM.WRK.TEXT, paramétrez le en entrant un nombre entre crochets
exemple: ESSAI[10] executera une sauvegarde vers un emplacement qui sera la premiere zone libre de dix blocs de la disquette.

b) PROCESSUS DE COMPILATION

En cours de compilation , un certain nombre d'informations apparaissent à l'écran.
Le nom des procédures ,des fonctions du programme principal s'affichent au fur et à mesure de leur traitement par le compilateur.
les nombres entre chevrons(exemple <50>) indiquent quelle ligne du programme source le compilateur est en train de traiter et chaque point indique la progression d'une nouvelle ligne.
L'indication xxxwords exemple: [2106 words] signifie qu'a cette phase de la compilation, il reste un nombre indiqué de mots de 16 bits disponibles.Si le nombre de mots disponibles devient inférieur a 550 la compilation s'arrête.

```
Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, D(ebug,? [II.0]




PASCAL Compiler [II.0.A.1]
(   0)...........
DECOMPOS [10050 words]
(  11).......................
34 lines
Smallest available space = 10050 words
```

On peut remédier à cet inconvénient en ajoutant au programme la directive de compilation swap (voir directives de compilation)

En fin de compilation, au bas de l'écran s'affiche le nombre de lignes compilées et le plus petit espace mémoire disponible en cours de compilation (smallest available space).
Dès lors,vous pouvez tester le programme compilé par R(un .

c) ERREURS DE SYNTAXE
Si une erreur de syntaxe est détectée,au cours de la compilation ,celle-ci s'arrête après un l'émission d'un bip sonore.
Sur l'écran quatre flèches <<<< signalent la partie du programme ayant provoquée l'erreur (le numéro de ligne suivi du numéro de l'erreur sont affichés).

Le programmeur dispose alors de trois possibilités
1: soit abandonner la compilation par CTRL [et revenir au bandeau de commandes.
2: soit poursuivre la compilation en appuyant sur la barre d'espace .
bien entendu,si d'autres erreurs sont décelées,un nouveau bip vous redemandera un choix.
3: soit revenir en mode édition par E(dit, ce qui vous permettra de corriger l'erreur et de relancer la compilation depuis le début du programme.
En revenant en édition,le curseur se positionnera a l'endroit où l'erreur a été détectée,et un texte explicitant le type d'erreur apparait en haut de l'ecran.
Pour corriger une erreur,il faut d'abord appuyer sur la barre d'espace.
Seule une compilation ,sans aucune erreur, aboutira à la création d'un fichier code éxecutable.
L'indication de l'endroit où se situe l'erreur et le type d'erreur n'est pas fiable à 100% (comme en Basic, une erreur peut découler d'une omission ou d'une erreur de programme préalable en amont dans le programme).

ERREURS DE COMPILATION
---

1 ERREUR DANS UN TYPE SIMPLE
2 IDENTIFICATEUR ATTENDU
3 ERREUR NON IMPLEMENTEE
4) ATTENDUE
5 : ATTENDU
6 SYMBOLE ILLEGAL OU MANQUE ; PLUS-HAUT
7 ERREUR DANS LA LISTE DES PARAMETRES
8 OF ATTENDU
9 (ATTENDUE
10 ERREUR DE TYPE
11 [ATTENDU
12] ATTENDU
13 END ATTENDU
14 ; ATTENDU
15 ENTIER ATTENDU

16 = ATTENDU
17 BEGIN ATTENDU
18 ERREUR DANS LA DECLARATION
19 ERREUR DANS UNE LISTE DE CHAMPS
20 . ATTENDU
21 * ATTENDU
22 INTERFACE ATTENDU
23 IMPLEMENTATION ATTENDU
24 UNIT ATTENDU
25->49 non définies
50 ERREUR DANS UNE CONSTANTE
51 := ATTENDU
52 THEN ATTENDU
53 UNTIL ATTENDU
54 DO ATTENDU
55 TO OU DOWNTO ATTENDU
56 IF ATTENDU
57 FILE ATTENDU
58 EXPRESSION INCORRECTE
59 ERREUR DANS UNE VARIABLE
60 OBLIGATOIREMENT DE TYPE SEMAPHORE
61 OBLIGATOIREMENT DE TYPE PROCESSID
62 PROCESSUS NON AUTORISE A CE NIVEAU
63 SEULES LES TACHES PRINCIPALES PEUVENT DEMARRER DES PROCESSUS
64-> 100 non définies
101 IDENTIFICATEUR DECLARE DEUX FOIS
102 BORNE INFERIEURE PLUS GRANDE QUE BORNE SUPERIEURE
103 IDENTIFICATEUR DE LA MAUVAISE CLASSE
104 IDENTIFICATEUR NON DECLARE
105 SIGNE NON AUTORISE
106 NOMBRE ATTENDU
107 TYPES INTERVALLES INCOMPATIBLES
108 FICHIER NON AUTORISE ICI
109 TYPE REEL NON AUTORISE
110 ETIQUETTE DOIT ETRE DE TYPE SCALAIRE OU INTERVALLE
111 INCOMPATIBLE AVEC L'ETIQUETTE DECLAREE
112 INDEX NE PEUT ETRE REEL
113 INDEX TYPE SCALAIRE OU INTERVALLE ATTENDU
114 LE TYPE DE BASE NE PEUT PAS ETRE REEL
115 TYPE DE BASE DEVANT ETRE SCALAIRE OU INTERVALLE
116 ERREUR DE TYPE D'UN PARAMETRE
117 REFERENCE EN AVAL NON SATISFAITE
118 IDENTIFICATEUR NE POUVANT ETRE UNE VARIABLE
119 NE PAS RE-SPECIFIER LES PARAMETRES
120 UN RESULTAT DE FONCTION DOIT ETRE SALAIRE,INTERVALLE OU POINTEUR
121 FICHIER PASSE PAR VALEUR (NON AUTORISE)
122 NE PAS RESPECIFIER LE TYPE DE RESULTAT

123 TYPE DE RESULTAT MANQUANT DANS DECLARATION DE FONCTION
124 FORMAT F POUR REELS SEULEMENT
125 ERREUR DE TYPE D'UN PARAMETRE D'UNE PROCEDURE STANDARD
126 LE NOMBRE DE PARAMETRES DIFFERE DE LA DECLARATION
127 SUBSTITUTION DE PARAMETRE ILLEGALE
128 TYPE DE RESULTAT NE CORRESPONDANT PAS A LA DECLARATION
129 CONFLIT DE TYPE ENTRE OPERANDES
130 L'EXPRESSION N'EST PAS DU TYPE ENSEMBLE
131 SEULS LES TESTS D'EGALITES SONT AUTORISES
132 INCLUSION STRICTE NON AUTORISEE
133 COMPARAISONS DE FICHIERS NON AUTORISEES
134 TYPE D'OPERANDES ILLEGAL
135 TYPE D'OPERANDES BOOLEEN ATTENDU
136 UN ELEMENT D'ENSEMBLE DOIT ETRE SCALAIRE OU INTERVALLE
137 LES TYPES DES ELEMENTS DOIVENT ETRE COMPATIBLES
138 LA VARIABLE N'EST PAS DE TYPE TABLEAU
139 LE TYPE DE L'INDEX EST INCOMPATIBLE AVEC LA DECLARATION
140 LLA VARIABLE N'EST PAS DE TYPE ENREGISTREMENT
141 LE TYPE DE LA VARIABLE DOIT ETRE FICHIER OU POINTEUR
142 PARAMETRE ILLEGAL
143 VARIABLE DE BOUCLE ILLEGALE
144 EXPRESSION DE TYPE ILLEGAL
145 CONFLIT DE TYPE
146 ASSIGNATION DE FICHIERS NON AUTORISEE:
147 TYPE D'ETIQUETTE INCOMPATIBLE AVEC EXPRESSION DE SELECTION
148 LES BORNES D'INTERVALLES DOIVENT ETRE SCALAIRES
149 LE TYPE DE L'INDEX DOIT ETRE ENTIER
150 ASSIGNATION A DES FONCTIONS STANDARD NON AUTORISEE
151 ASSIGNATION A DES FONCTIONS FORMELLES NON AUTORISEE
152 CHAMP ABSENT DE L'ENREGISTREMENT
153 ERREUR DE TYPE EN LECTURE
154 LE PARAMETRE ENVOYE DOIT ETRE UNE VARIABLE
155 VARIABLES DE CONTROLE NE POUVANT ETRE FORMELLES OU GLOBALES
156 ETIQUETTE DE CASE DEFINIE PLUSIEURS FOIS
157 TROP DE CAS POUR CASE
158 VARIANTE INCONNUE
159 CHAMPS REELS OU CHAINE NON AUTORISES
160 LA DECLARATION PRECEDENTE N'ETAIT PAS EN AVAL
161 DECLARATION FORWARD DEJA FAITE
162 LA TAILLE DES PARAMETRES DOIT ETRE CONSTANTE
163 VARIANTE MANQUANTE DANS LA DECLARATION
164 SUBSTITUTION D'UNE PROCEDURE/FONCTION STANDARD NON PERMISE
165 ETIQUETTE DEFINIE PLUSIEURS FOIS
166 ETIQUETTE DECLAREE PLUSIEURS FOIS
167 ETIQUETTE NON DECLAREE
168 ETIQUETTE NON DEFINIE
169 ERREUR DANS UN ENSEMBLE
170 PARAMETRE PAR VALEUR ATTENDU

171 FICHIER STANDARD REDECLARE
172 FICHIER EXTERIEUR NON DECLARE
173 PROCEDURE OU FONCTION FORTRAN ATTENDUE
174 PROCEDURE OU FONCTION PASCAL ATTENDUE
175 UN SEMAPHORE DOIT ETRE PASSE EN PARAMETRE PAR VARIABLE
176->181 non définies
182 IMBRICATIONS D'UNITS INTERDITES
183 DECLARATIONS 'EXTERNAL' NON AUTORISEES A CE NIVEAU
184 DECLARATIONS 'EXTERNAL' NON AUTORISEES DANS SECTION INTERFACE
185 DECLARATIONS DE SEGMENTS NON AUTORISEES EN SECTION INTERFACE
186 ETIQUETTES NON AUTORISEES EN SECTION INTERFACE
187 ERREUR A L'ESSAI D'OUVERTURE DE BIBLIOTHEQUE
188 UNIT NON DECLARE PAR UN USES
189 USES NON AUTORISE A CE NIVEAU
190 UNIT ABSENTE DE LA BIBLIOTHEQUE
191 LA DECLARATION ANTICIPEE N'ETAIT PAS UN SEGMENT
192 USES DOIT FIGURER DANS SECTION INTERFACE
193 PAS ASSEZ DE PLACE POUR CETTE OPERATION
194 OPTION DE COMPILATION A FAIRE FIGURER EN DEBUT DE PROGRAMME
195 UNIT NON IMPORTABLE
196 -> 200 non définies
201 ERREUR DANS UN NOMBRE REEL
202 UNE CONSTANTE CHAINE NE PEUT DEPASSER LA LIGNE SOURCE
203 CONSTANTE ENTIERE HORS LIMITES
204 8 OU 9 NE PEUVENT FIGURER DANS UN NOMBRE OCTAL
205 -> 249 non définies
250 TROP DE NIVEAU D'IMBRICATION D'IDENTIFICATEURS
251 TROP DE PROCEDURES OU FONCTIONS IMBRIQUEES
252 TROP DE REFERENCES ANTICIPEES DE PROCEDURES
253 PROCEDURE TROP LONGUE
254 TROP DE CONSTANTES DANS CETTE PROCEDURE
255 non définie
256 TROP DE REFERENCES EXTERNES
257 TROP D'OBJETS EXTERNES
258 TROP DE FICHIERS LOCAUX
259 EXPRESSION TROP COMPLIQUEE
260 -> 299 non définies
300 DIVISION PAR ZERO
301 VALEUR NON DEFINIE DANS CASE
302 EXPRESSION D'INDEX HORS BORNES
303 LA VALEUR A ASSIGNER EST EN DEHORS DES BORNES
304 LE RESULTAT DE L'EXPRESSION EST HORS LIMITES
305 -> 397 non définies
398 RESTRICTION D'IMPLEMENTATION
399 RESTRICTION D'IMPLEMENTATION (COMPILATEUR PAS DEFINI POUR CELA)
400 CARACTERE ILLEGAL DANS LE TEXTE

401 FIN DE PROGRAMME INATTENDUE
402 ERREUR D'ECRITURE DU FICHIER CODE (PAS ASSEZ DE PLACE)
403 ERREUR A LA LECTURE D'UN FICHIER A INCLURE
404 PAS ASSEZ DE PLACE POUR ECRIRE LE LISTING DE COMPILATION
405 IMPOSSIBLE D'ACCEDER A CETTE PROCEDURE ICI
406   INCLUSION DE FICHIER ILLEGALE
407 TROP DE BIBLIOTHETHEQUES APPELEES
408   SECTION INTERFACE ABSENTE D'UN FICHIER
409   NOM D'UNIT RESERVE POUR SYSTEME
410   ERREUR DISQUE
411 à 499   non définies
500   ERREUR DE L'ASSEMBLEUR

d) LES DIRECTIVES DE COMPILATION

Elles seront incluses dans le programme source pour indiquer au compilateur certaines orientations ou dispositions.
Elles sont placées entre accolades (ou entre parenthèses et astérisques) et sont précédées du caractère $ .elles sont désignées par une seule lettre .

exemple : {$Q+} active la directive Quiet .La désactivation se faisant par {$Q-} .
On peut aussi remplacer les accolades par (* et *) exemple : (*$Q+*) et (*$Q-*)
Ne pas laisser d'espaces blancs entre les caractères.
on peut aussi grouper dans certains cas plusieurs directives.
exemple (*$G+,$R-*)

 Liste des directives disponibles.

DIRECTIVE LIST (*$L+*) :

Elle indique au compilateur qu'il doit aussi produire un listing du programme compilé .
Ce listing nommé SYSTEM.LST.TEXT sera automatiquement sauvegardé sur le volume racine.
(*$LCONSOLE:*) affiche le listing à l'écran et (*$LPRINTER:*) imprime le listing.
(*$Lnom de fichier*) est aussi utilisable .Pour desactiver cette directive ecrire dans le programme (*$L-*)

Indications fournies par le listing de compilation:

 -la première colonne du listing de compilation indique le numéro de la ligne du programme texte.
 - la deuxième colonne du listing de compilation indique le numéro du segment de programme.ce sera 1 si le programme n'est pas segmenté.
 -la troisième colonne du listing de compilation indique le numero de la procédure .le programme principal ayant le numéro 1 .

- la quatrième colonne du listing de compilation indique qu'il s'agit d'une partie déclaration si elle vaut D ,sinon s'il s'agit d' une ligne éxécutable on verra l'affichage d'un nombre.Ce nombre indique le degré d'imbrication de l'instruction exécutable.;les lignes de degré n étant controlées par les lignes de degré n-1
- la cinquième colonne indique soit le nombre d'octets de p-code deja compilés si la ligne est une ligne exécutable, soit le nombre de mots de 16 bits alloués aux variables de la procédure,s'il s'agit d'une ligne de déclaration.

DIRECTIVE INCLUDE

Elle permet d'inclure un fichier .TEXT, au programme principal à partir de l'endroit où elle est inscrite.
exemple :(*$IVOL2:MATH*) indique au compilateur qu'à cet endroit il doit rechercher le fichier MATH sur la disquette VOL2 ,l'associer au programme principal et le compiler.
un fichier include peut contenir lui-meme une nouvelle directive include .on dira alors que la profondeur d'imbrication de l'inclusion est de un.
(la version I V.0 ucsd permet un niveau d'imbrication jusqu'à une profondeur de 3,mais,les versions antérieures,comme la version THOMSON,n'autorisent qu'un seul niveau.)

Remarque:
la directive INCLUDE étant suivie d'un nom de fichier ,si ce nom venait à avoir + ou - comme premier caractère (ce qui est rare !) il faudrait laisser un espace entre I et ce premier caractère.

DIRECTIVE QUIET
(*$Q+*) Elle permet d'allèger l'affichage en supprimant l'affichage standard ,décrit plus haut au moment de la compilation.
Pour rétablir cet affichage taper (*$Q-*)

DIRECTIVE SWAP
(*$S+*)Elle doit être placée au debut du programme texte et permet de libérer plus de place en mémoire pour permettre la compilation de gros programmes.
L'utiliser si la compilation s'arrête en affichant :
STACK OVERFLOW
L'utiliser aussi dans les cas où le programme appelle plusieurs unités.
Si ce n'est pas suffisant employer (*$S++*)
Les directives (*$S+*) et surtout (*$S++*) ont pour inconvenient de ralentir la vitesse de compilation.

DIRECTIVE GOTO
(*$G+*) Elle doit être placée au debut d'une procédure si on a l'intention d'utiliser une instruction GOTO (branchement inconditionnel) à l'intérieur de cette procédure.
Rappelons que l'emploi de GOTO est très restrictif en PASCAL et necessite l'emploi d'étiquettes.

DIRECTIVE I/OCHECK

Elle permet la suppression des interruptions d'un programme lorsque survient une erreur d'entrée/sortie pendant l'éxécution .
activée par (*$I+*) et désactivée par (*$I-*)

DIRECTIVE PAGE
ELLE crée un saut de page sur le listing lorsque celui-ci est demandé.
(*$P*)

DIRECTIVE COPYRIGHT
Elle permet l'inscription dans le fichier CODE (au bloc 0) d'une indication de copyright .
(*$CPIERRE DUPONT (1992)*)
Elle doit être placée avant le mot eéservé PROGRAM

DIRECTIVE USES LIBRARY
Lorsque le programme appelle une unité (UNIT) et que celle-ci se trouve dans un fichier CODE ,autre que SYSTEM.LIBRARY (librairie principale),il faut indiquer au compilateur le nom du fichier code qui contient l'interface des unités appelées.
exemple:
Si le programme principal appelle les unités U1 et U2 contenues dans SYSTEM.LIBRARY ;il suffit de mettre après le nom du programme USES U1,U2
Si le programme principal appelle les unités U1 et U2 qui se trouvent dans une librairie personnelle nommée LIBPERSO.CODE copiée sur la disquette VOL8:
alors après le nom du programme il faudra écrire l'appel de cette manière:

USES (*$UVOL8:FICH.CODE*)U1,U2

DIRECTIVE RANGE CHECK

Elle permet de supprimer les tests de vérification de validité des intervalles de valeurs des variables
on l'active par (*$R-*)
ainsi lorsqu'on accède à une variable de type intervalle,le compilateur génère du code pour vérifier si celle-ci est bien entre les bornes.
Cette directive doit être activée avec précaution et uniquement pour accéder aux cellules d'un tableau dont la taille est inconnue au moment de la compilation (tableaux dynamiques),
Elle permet d'accélérer la vitesse d'exécution d'un programme et d' éviter des erreurs d'éxécution stoppant le déroulement d'un programme.

DIRECTIVE RESIDENT

Nous rappelons que lorsqu'un programme est découpé en segments ,le code de chacun des segments n'est chargé en mémoire qu'au moment où le programme en a besoin (voir SEGMENTATION).
Mais dans certains cas (accès disquette trop fréquent) il peut s'avérer plus commode de forcer le programme à conserver le code du segment en mémoire même en dehors des périodes d'utilisation.
Ceci est possible en activant la directive R

(*$Rnom du segment*) est à placer juste après le BEGIN de la procédure.

DIRECTIVE USER

Permet de compiler au niveau lexical (*$U-*)
possibilité de placer des unités après la déclaration des variables globales,activation automatique de G+,R-,I-.
Ces programmes ne peuvent être éxécutés par X(ecute mais doivent être assemblés en utilisant le LIBRARIAN et placés dans SYSTEM.PASCAL.

Cette directive semble très complexe à utiliser.

DIRECTIVE VERIFY

Quand elle est désactivée par (*$V-*) ,le compilateur ne contrôle plus la longueur des chaines déclarée par STRING[n] .Comme il y a risque d'écrasement de zones mémoires ,cette directive est assez dangereuse d'emploi.

DIRECTIVE E
Elle est activée par (*$E+*) et permet de déclarer des fichiers dans la partie implémentation d'une unité.

DIRECTIVE FLIP-BYTE

Sur TO8/TO9 le micro-processeur 6809 implante les mots mémoires dans l'ordre poids fort,poids faible .
Cette directive permet de générer des adresses ayant l'octet faible en premier pour avoir une portabilité des fichiers codes sur 6502,8080,8086.
mettre la directive à (*$F-*).

DIRECTIVE NO LOAD

Elle s'active par (*$N+*) et évite que le code des unités appelées par le programme principal ne soit présent en mémoire du début à la fin du programme inutilement.

e) LA SEGMENTATION

On est parfois confronté à un problème de place disponible en mémoire si on doit compiler un important programme.
Nous avons vu plus haut qu'il faut parfois recourir à la directive de compilation SWAP pour y parvenir,mais celle-ci n'est pas une panacée.

Il peut être plus judicieux ,à ce moment,d'écrire son programme pour obtenir non pas un seul gros fichier code mais plusieurs.

Un segment est une petite unité de p-code qui peut être chargée en mémoire centrale uniquement lorsque c'est nécessaire,puis ensuite être retirée pour libérer de la place.Le Pascal UCSD gère cela.

Il faudra remplacer dans le programme les fonctions ou procédures ordinaires par des segments procédures ou segments fonctions qui seront dans des segments indépendants du code du programme principal

Les segments procédures et segments fonctions seront écrites de la même façon que les fonctions ou procedures ordinaires.
Il faut simplement spécifier SEGMENT comme mot reservé dans l'en-tête de la procédure.

```
exemple: SEGMENT PROCEDURE CALCUL(VAR ...)
        VAR X,Y: INTEGER;
        BEGIN .....
```

Les segments procédures et les segments fonctions doivent imperativement être declarées dans le programme avant toute instruction éxecutable de procédure ou fonction ordinaires.
Par conséquent, les définitions de segment procédures précedent obligatoirement les definitions des autres procedures.et si une segment procédure appelle une procédure ordinaire celle -ci devra apparaitre,avant,dans une partie déclaration,suivie du mot-réservé FORWARD pour préciser que la partie la définissant est située plus loin dans le programme.

f) MESSAGES D'ERREURS POUR LES ENTREES/SORTIES

0 AUCUNE ERREUR
1 BLOC DOUTEUX
2 NUMERO D'UNITE ILLEGAL
3 MODE INADAPTE
4 ERREUR SYSTEME INDEFINIE
5 UNITE NON CONNECTEE
6 FICHIER PERDU NON RETROUVE EN DIRECTORY
7 TITRE ILLEGAL
8 PLACE INSUFFISANTE SUR DISQUETTE
9 UNITE ABSENTE
10 FICHIER ABSENT SUR L'UNITE CHOISIE
11 DEUX FICHIERS PORTENT LE MEME NOM
12 ESSAI D'OUVERTURE D'UN FICHIER NON FERME
13 ACCES IMPOSSIBLE SI FICHIER FERME
14 MAUVAIS FORMAT ERREUR EN LISANT UN REEL OU UN ENTIER
15 MEMOIRE TAMPON SATUREE

La suppression des messages d'erreurs entrées/sorties est possible en mettant la directive iocheck (*$I-*) dans le programme.Avec (*I+*) on retrouve l'affichage de ces

messages d'erreur.IORESULT retourne une valeur correspondant au numero de l'erreur d'entree/sortie.

XIII- LES UNITES (UNITS)

Dans l'écriture d'un programme on aura parfois recours,à des procédures ou fonctions qui ont déjà été écrites et compilés ultérieurement et qu'il serait fastitieux d'avoir à ré-écrire.
Ainsi,un grand nombre de procédures mathématiques,graphiques etc. ,spécifiques aux TO8 TO9 sont déjà fournies sur la disquette du langage Pascal UCSD .
Ces procédures,sont rangées dans des unités.
Les unités sont elles-mêmes groupées et rangées soit
-dans SYSTEM.LIBRARY (librairie système principale)
- ou ailleurs dans une quelconque librairie personnelle .

Nous avons vu déjà la différence que cela entraine lors de la compilation où l'accès à SYSTEM.LIBRARY est automatique tandis que l'accès aux librairies personnelles necessite d' activer la commande L(ink .

Le programmeur peut écrire ses propres fonctions ou procédures et les ranger dans une librairie personnelle ou même dans SYSTEM.LIBRARY s'il reste de la place disponible.

L'écriture d'une unité obéit à certaines règles de syntaxe et différe un peu de l'écriture d'un programme classique.
Vous pouvez vous reporter aux explications données dans le bulletin du club CONTACTHOMS n°10 (avril 92).

a) STRUCTURE D'UNE UNITE

Une unité comporte quatre parties
L'en-tête UNIT nom de l'unité
La deuxième partie qui débute par le mot INTERFACE
comportant les déclarations habituelles (constantes,types,variables ,procédures,fonctions)
La troisième partie qui débute par le mot IMPLEMENTATION contient toutes les fonctions et instructions locales suivies de la définition des fonctions et procédures de la partie INTERFACE.
la quatrième partie dite d'initialisation contient un bloc d'instructions encadrées par BEGIN et END.
Si l'initialisation n' est pas necessaire il faut néanmoins mettre un begin puis un end pour cette quatrième partie.
Une foisle texte source de l'unité écrite ,il faut la compiler en procédant comme avec n'importe quel programme ordinaire (voir COMPILE)
L'unité pourra être rangées dans une librairie qu'il vous appartiendra de nommer (ces librairies sont des fichiers codes) (voir LIBRARIAN)

b) UNITES ECRITES EN ASSEMBLEUR

Il est parfois pratique pour certaines routines du programme d'appeler des routines en ASSEMBLEUR (voir ASSEMBLE).
On peut ainsi créer des unités à l'aide de modules écrits en langage assembleur et les ranger dans SYSTEM.LIBRARY ou une autre librairie.
Les unités fournies à l'origine dans SYSTEM.LIBRARY (EXTRA,UTILS etc) et que nous décrivons plus loin sont d'ailleurs ,pour la plupart écrites en assembleur 6809.

c) LE LIBRARIAN

Cet utilitaire permet de ranger ses propres unités dans SYSTEM.LIBRARY (ainsi l'unité sera automatiquement reliée au programme principal appelant lors du premier RUN.)
on accède au librarian par X(ecute LIBRARY .
Au préalable,votre unité devra être compilée,nommée et figurer sur une disquette.
exemple: GRAFIC.CODE
L'utilitaire LIBRARIAN est conçu de telle sorte que pour ajouter votre unité à SYSTEM.LIBRARYY (qui en contient déjà plusieurs),il faut :
créer une nouvelle bibliotheque (que l'on nommera évidemment SYSTEM.LIBRARY) pour y transférer toutes les anciennes unités puis y ajouter la nouvelle.
Il va de soi, qu'il faut faire cette opération sur une copie de la SYSTEM.LIBRARY originale car il y a écrasement des données .

Dans un premier temps, le librarian vous demande le nom de la nouvelle bibliothèque à créer par OUTPUT CODE FILE ? tapez SYSTEM.LIBRARY
pour le rangement dans une autre librairie la procédure est identique
puis le librarian vous demande le nom du fichier à ajouter par
LINK CODE FILE ? tapez une nouvelle fois SYSTEM.LIBRARY et ENT pour valider.
Une grille s'affiche alors sur l'écran avec le contenu de SYSTEM.LIBRARY chaque unité étant précédée d'un numero
Il faut placer toutes les unités (ou une partie seulement si vous le voulez) dans la nouvelle version de SYSTEM.LIBRARY
Pour cela taper le numéro correspondant puis la barre d'espace.
On vous demande,ensuite, le numéro de l'emplacement où ira se ranger,l'unité, dans la nouvelle version de SYSTEM.LIBRARY
indiquer ce numéro d'emplacement en réponse à
SEG TO LINK INTO ? tapez un numéro et barre d'espace.
Repétez l'opération pour transfèrer les unités de l'ancienne SYSTEM.LIBRARY vers la nouvelle version.
Enfin pour y inclure votre nouvelle unité tapez N(ew file et entrez son nom suivi du suffixe .CODE.
Votre unité ira se ranger,dans un emplacement de l'ancienne version de SYSTEM.LIBRARY et vous devrez enfin la transférer vers la nouvelle par la meme procédure que pour les autres unités décrite ci-dessus.
Dans le transfert des unités vers la nouvelle version de SYSTEM.LIBRARY ,réserver l'emplacement 0 pour l'unité OUTERLEV et ne laissez pas d'emplacement vide entre deux emplacements pour économiser de la place en mémoire.
Si un message vous indique qu'il n'y a plus assez de place , pour ranger votre unité ,ce qui est possible car SYSTEM.LIBRARY(version thomson) contient deja 11 unités.Alors,

supprimez les unités qui vous sont moins indispensables .d'où l'obligation d'avoir une copie des 11 unités originales sur une disquette a part.

Quand vous aurez inclus votre unité personnelle dans la nouvelle SYSTEM.LIBRARY. tapez Q(uit pour quitter le LIBRARIAN.
Le message NOTICE? demande d'entrer un mot de passe; vous pouvez simplement taper ENT (pas de mot de passe).

Il est recommandé,en dernier lieu, d'activer la commande I(nitialize dès le retour au bandeau de commandes.

XIV- AUTRES UTILITAIRES

a) CONFIG.CODE

Utilitaire indispensable pour formater les disquettes sous Pascal.
(Malheureusement un bug interdit la gestion des disquettes double_face ;il faut se contenter d'une seule face par disquette).
Il permet aussi de passer de 80 colonnes en mode 40 colonnes .
Il permet,également, de déclarer par la commande Z(éro un disque virtuel sous Pascal (celui_ci devra cependant avoir été choisi au menu réglage du TO avant de passer sous pascal) .Le disque virtuel prend pour nom RAM: et pour numéro #9.

b)LIBMAP.CODE

Utilitaire permettant d'afficher le contenu d'une librairie avec l'ensemble des unités et les procédures et fonctions qui s'y rattachent .

(exemple : le contenu de SYSTEM.LIBRARY donné en fin de ce mode d'emploi)

c)PATCH80.CODE

PATCH est un utilitaire qui permet de manipuler, d'examiner et éventuellement d'altérer les fichiers. Patch a été écrit pour faire du rapiéçage de fichier au niveau du bit et autres opérations douteuses, bien que pouvant se révéler parfois utiles. Il fut d'abord écrit en tant qu'utilitaire personnel mais fut rapidement incorporé à l'ensemble standard des outils systèmes.
PATCH est destiné à être utilisé de manière interactive avec un écran vidéo.
Il y a deux facilités principales dans Patch: un mode permettant l'édition des fichiers au niveau de l'octet et un mode permettant leur visualisation (DUMP) selon différents formats.
La possibilité d'édition par octet permet à l'utilisateur d'éditer non seulement des fichiers texte, mais aussi de faire des modifications rapides de fichiers code et également de créer des fichiers spécialisés dans certains tests.
La visualisation du contenu des fichiers peut se faire dans différentes bases arithmétiques. Le contenu de la mémoire peut être également visualisé.

Lorsqu'il exécute PATCH pour la première fois, l'utilisateur se trouve en mode EDIT. Le mode DUMP s'obtient en frappant 'D'. Aucune information n'est perdue lors du passage d'un mode à l'autre.
EDIT permet à l'utilisateur d'ouvrir un fichier ou un volume, de lire les blocs qu'il aura sélectionné (en donnant leurs numéros relatifs) dans un tampon (buffer) d'édition et ainsi d'examiner ou de modifier ce tampon (au moyen de la commande TYPE) et, pour finir, d'écrire le bloc ainsi modifié sur le fichier. Les tampons (buffers) sont affichés à l'écran dans le format désiré, et édités d'une manière tout à fait similaire à celle de l'éditeur d'écran.
Les commandes individuelles de EDIT sont expliquées en détail ci-après. Lorsqu'il lui est impossible d'exécuter une commande, PATCH affiche un message auto-explicatif.
Les lignes de commande de EDIT sont:

EDIT : D(ump, G(et, R(ead, S(ave, M(ix, T(ype, I(nfo, F(or, B(ack, ?
EDIT : V(iew, W(ipe, Q(uit, ?

D(ump appelle DUMP
G(et ouvre le fichier ou le disque que l'on veut examiner et charge le bloc zéro dans le tampon.
R(ead lit le bloc spécifié dans le fichier courant.
S(ave écrit le contenu du tampon sur le bloc physique courant.
M(ixed change le format d'affichage du bloc courant.En entrant 'M' on active alternativement l'un des deux formats suivants
Mixed qui affiche les caractères ASCII imprimables et l'equivalent en Hexadecimal des caractères non-imprimables.
HEX qui affiche les blocs en hexadecimal.
I(nfo affiche des informations relatives au fichier courant.Ces infos comprennent:
-Le nom du fichier
-la longueur du fichier
-le numéro du bloc courant
-l'état d'ouverture du fichier
-la possibilité de faire des UNITREADs
-le numéro d'unité (-1 si UNITIO est à False)
-le sexe d'octet de la machine
F(orward permet de lire le bloc suivant du fichier
B(ackward permet de lire le bloc précédent du fichier
V(iew affiche le bloc courant
W(ipedisplay efface l'affichage du bloc à l'écran
Q(uit permet de sortir du programme PATCH80
T(ype permet d'entrer dans le mode permettant une modification du tampon.

Mode TYPE

TYPE permet, tout comme avec l'éditeur, de modifier l'information affichée sur l'écran en déplaçant le curseur et en écrasant les informations existantes. Si vous faites des erreurs en utilisant TYPE, n'utilisez pas la commande S(ave pour sauvegarder le

tampon comme lorsque vous êtes en mode EDIT, mais utilisez la commande R(ead pour relire le bloc et recommencer.
La ligne de commande de TYPE est:

TYPE : C(har, H(ex, F(ill, U(p, D(own, L(eft, R(ight, <vector arrows>, Q(uit

C(haracter change les caractères du tampon par les caractères ASCII tels qu'ils sont frappés, à partir de la position courante du curseur, jusqu'à ce qu'un <etx> soit frappé. Seuls les caractères imprimables sont acceptés.
H(ex change les caractères du tampon par les chiffres hexadécimaux tels qu'ils sont frappés, à partir de la position courante du curseur, jusqu'à ce qu'un 'Q' soit frappé. Les chiffres hexadécimaux peuvent être frappés en majuscule ou en minuscule.
F(ill remplit une partie du bloc courant avec le même caractère. F(ill accepte soit les caractères ASCII, soit les chiffres hexadécimaux. Lorsque la commande est achevée, le curseur se positionne après le dernier octet.

Les commandes suivantes permettent de déplacer le curseur à l'intérieur du bloc de données affiché. Le curseur se trouve toujours positionné sur un octet particulier. Plutôt que d'effacer l'écran le curseur saute d'un côté à l'autre et du haut vers le bas de l'écran.

U(p déplace le curseur d'une ligne vers le haut.
D(own déplace le curseur d'une ligne vers le bas.
L(eft déplace le curseur d'une colonne à gauche.
R(ight déplace le curseur d'une colonne à droite.
<flèche>. Les flèches sont utilisées comme dans l'éditeur d'écran. Elles ont les mêmes effets que les commandes U,D,L,R.
Q(uit permet de sortir du mode TYPE et de retourner au mode EDIT.

Mode DUMP

Les dumps peuvent être produits dans les formats suivants: décimal, hexadécimal, octal, caractères ASCII (s'ils sont imprimables), caractères décimaux (BCD) et caractères octaux.
DUMP est également capable de mettre les octets dans un mot avant de l'afficher, ou d'afficher simultanément une liste de mots.
L'entrée de DUMP peut être un fichier disque spécifié par l'utilisateur ou, directement, la mémoire principale (il était initialement utilisé pour examiner l'interpréteur et le BIOS).
La taille de la ligne de sortie peut être contrôlée; une ligne peut contenir un certain nombre de mots machine. 15 mots remplissent une ligne de 132 caractères et 9 mots une ligne de 80 caractères.
Lorsque l'utilisateur entre dans DUMP, l'écran affiche une brève ligne de commande (D(o et Q(uit), ainsi qu'un menu plus long, qui permet de modifier les spécifications de formatage en frappant la lettre désirée et en entrant ensuite la spécification.
Les spécifications sont les suivantes:

A) l'entrée: un fichier résidant sur disque ou une unité périphérique.

B) le numéro de bloc à partir duquel le dump commence. Si (A) est un périphérique, ce numéro n'est pas vérifié.
C) le nombre de blocs à imprimer. Si ce nombre est trop grand, le dump se termine lorsqu'il n'y a plus de bloc à éditer.
D) le dump commence au moment où l'on frappe 'D'.
E) cette sous-commande est une bascule: si elle est mise à True, DUMP imprime le contenu de la mémoire principale; si elle est mise à False, l'entrée se fait à partir du fichier défini par (A).
F) cette sous-commande est un déplacement: le dump commence à partir de l'octet 0 <= (F) <= maxint.
G) cette sous-commande donne le nombre d'octets à imprimer (entre 0 et maxint).
H) cette sous-commande donne le fichier de sortie, qui est ouvert comme un fichier .TEXT.
I) cette sous-commande donne la longueur de la ligne d'édition, exprimée en mots machine (1 à 15).

Les six spécifications suivantes ont trois paramètres associés qui doivent être spécifiés par: USE, FLIP et BOTH.
USE indique à DUMP s'il doit utiliser ou non le format associé à chaque spéfification.
FLIP indique à DUMP s'il faut ou s'il ne faut pas flipper, c'est à dire permuter, les octets avant d'afficher les mots dans le format qui a été précédemment indiqué.
BOTH indique à DUMP d'afficher simulltanément les versions flippée et non flippée de la ligne. Si un BOTH est mis à True, la valeur de FLIP n'a pas d'importance.

J) affiche chaque mot sous forme d'un entier décimal.
K) affiche chaque mot sous forme d'un nombre hexadécimal, dans l'ordre des octets.
L) affiche chaque mot sous forme d'un entier octal. C'est l'équivalent octal de (J).
M) affiche chaque mot en caractère ASCII, dans l'ordre des octets. Les caractères non imprimables sont affichés sous forme de nombres hexadécimaux.
N) affiche chaque mot en caractères décimaux (BCD) dans l'ordre des octets.
O) affiche chaque mot sous forme de nombres octaux, dans l'ordre des octets.
Q) permet de retourner au mode EDIT. DUMP se souvient de la spécification initiale.
S) imprime une ligne blanche après la version non flippée de la ligne.
T) met des lignes blanches entre les différents formats de lignes.

Les modes EDIT et DUMP se souviennent des informations pertinentes qu'ils contiennent quand l'autre mode est activé.

La notion d'octet flippés correspond à la disposition suivante: lorsque le sexe d'octet de la machine est inversé, c'est à dire lorsque l'octet le moins significatif (ou de moindre poids) d'un mot se trouve à l'adresse la plus basse, il est commode d'afficher le contenu de ces octets dans l'ordre naturel (octet de poids le plus élevé écrit à gauche). C'est ce que permet la commande FLIP, de façon à rendre plus agréable la lecture des dumps.

Tous les nombres fournis à PATCH par l'utilisateur sont lus sous forme de chaîne de caractères et ensuite convertis en entier. Seuls les cinq premiers caractères de la chaîne sont pris en considération. S'il y a un caractère non numérique dans la chaîne de caractère la valeur de l'entier est mise à zéro. Si un débordement de nombre entier

apparaît, la valeur de l'entier est MAXINT (dans la mesure où le débordement sur les nombres entiers est détecté par la présence d'un nombre négatif, les entiers de l'intervalle 65536..98303 seront pris modulo 32768).

d) COPYDUPDIR .CODE

Il est souvent utile de dupliquer le catalogue d'un disque. Cela peut permettre, dans certains cas, de régénérer les informations contenues dans le catalogue, informations qui avaient pu être perdues ou endommagées, et aider à restaurer un disque ou à restaurer les fichiers qui s'y trouvent dans l'état désiré. La commande Z(ero du Filer permet de créer un double du catalogue; l'utilitaire MARKDUPDIR décrit ci-après le permet aussi. Le Filer tient à jour le double du catalogue, après qu'il ait été, en même temps que le catalogue initial. L'utilitaire COPYDUPDIR recopie le double du catalogue à la place du catalogue initial.

Ce programme copie le double du catalogue du disque sur le catalogue initial. On exécute COPYDUPDIR par la commande eX(ecute. COPYDUPDIR demande le numéro du disque sur lequel la copie s'effectue (4 ou 5). Si le disque ne possède pas de double du catalogue, COPYDUPDIR le signale. S'il trouve le double du catalogue, COPYDUPDIR demande si l'on veut toujours écraser le catalogue initial, qui réside sur les blocs 2 à 5. Un 'Y' permet d'exécuter la copie; tout autre caractère fait abandonner le programme.

e) MARKDUPDIR.CODE

MARKDUPDIR permet de créer, sur un disque, un double du catalogue. L'utilisateur doit s'assurer que les blocs 6..9 peuvent être utilisés librement. S'ils ne le sont pas l'utilisateur doit réarranger ses fichiers sur le disque de façon à libérer les blocs 6 à 9. On peut s'assurer de leur disponibilité en demandant le catalogue du disque par la commande E(xtended du Filer et en regardant où commence le premier fichier. Si le premier fichier commence au bloc 6 ou s'il commence au bloc 10 mais qu'il y a un ensemble de quatre blocs inutilisés au début du disque, le disque ne possède pas de double du catalogue. Si, par contre, le premier fichier commence au bloc 10 et qu'il n'y a pas de bloc inutilisé en tête du catalogue le catalogue du disque a déjà été dupliqué.

Exemple:

```
SYSTEM.PASCAL        31   30-Aug-78     6      Codefile
```

ou:

```
<UNUSED>             4                  6
SYSTEM.PASCAL        31   30-Aug-78     10     Codefile
.
.
.
```

... les deux exemples ci-dessus indiquent que le disque ne possède pas de double du catalogue. L'exemple ci-dessous montre le catalogue dupliqué d'un disque:

```
SYSTEM.PASCAL        31   30-Aug-85    10      Codefile
.
.
.
```

Ce programme s'exécute en frappant 'X' suivi de MARKDUPDIR. Il demande quel est le disque sur lequel on veut écrire un double du catalogue (4 ou 5). MARKDUPDIR vérifie l'état des blocs 6 à 9. S'ils semblent libres, il demande confirmation en affichant la ligne de commande:
are sure they are free ?
Si l'on frappe un 'Y', on effectue la duplication, tout autre caractère fait abandonner le programme. Il faut être sûr que cet espace est libre avant de dupliquer le catalogue, car sinon les informations qu'il contenait seraient irrémédiablement perdues.

f)FLIPDIR.CODE

g)FLIPCODE.CODE

h)RECOVER.G.CODE

i)DISASM.II.CODE

XV- GESTION DE L'IMPRIMANTE

On pourra brancher et utiliser une imprimante de type PR 90-600 ou PR 90-612 avec un TO sous PASCAL UCSD.

Pour le moment il n'est pas possible d'imprimer sur une imprimante PC en utilisant Pascal UCSD en émulation avec TEO (contacter Mr Eric Botcazou pour de plus amples renseignements)

Nous avons déjà vu qu'en tant que périphérique standard l'imprimante se voit désigner par le système de deux façons par

- un numéro de périphérique --> #6
-un nom prédéfini --> PRINTER:

On peut imprimer à partir du gestionnaire de fichiers (Voir F(ile et T(ransfer).
Mais on peut aussi gérer l'impression à partir d'un programme spécifique écrit en Pascal (voir le programme de Thierry Chamoret dans les bulletins CONTACTHOMS n°11 et 12)

AUTRES EXEMPLES :

Voici un programme pour imprimer du texte :

```
PROGRAM IMPRIME_SALUTATIONS;
VAR PR:INTERACTIVE;
 MESSAGE:STRING;
BEGIN
RESET(PR,'PRINTER:');
MESSAGE:='SALUT A TOUS';
WRITELN(PR,MESSAGE);
 END.
```

il est possible d'envoyer des codes à l'imprimante à partir du Pascal UCSD, il suffit de déclarer une variable de type INTERACTIVE et d'ouvrir un fichier vers l'imprimante.

```
PROGRAM J_IMPRIME;
  VAR
    PR : INTERACTIVE ;
BEGIN
  RESET(PR,'PRINTER:');
  WRITE(PR,CHR(27),'C');
  WRITELN('BONJOUR');
                etc...
```

L'envoi de ESCAPE C produira l'impression de BONJOUR en mode condensé sur l'imprimante THOMSON PR-90612.

On peut aussi utiliser la procédure UNITWRITE (voir le livre de John Colibri Topiques Pascal)

```
 CODE[0]:=27;CODE[1]:=67;
UNITWRITE(6,CODE,2,0,12);
```

où CODE est de type ARRAY[0,1] OF 0..255"
6 désigne l'imprimante,2 le nombre d'octets à passer,0 signifiant que l'E/S n'est pas une disquette,12 une constante système.

Cette procédure ne controle pas les risques d'erreur en entrées-sorties

ANNEXE I

MOTS RESERVES DU PASCAL UCSD

AND ARRAY BEGIN CASE CONST DIV DO DOWNTO ELSE END
FILE FOR FORWARD FUNCTION GOTO
IF IN INTERFACE LABEL IMPLEMENTATION
MOD NIL NOT OF OR

PACKED PROCEDURE PROGRAM RECORD REPEAT
SEGMENT SET THEN TO TYPE
UNIT USES VAR WHILE WITH

TYPES ET FONCTIONS PREDEFINIES :

ABS ARCTAN ATTACH BOOLEAN CHAR
CLOSE CONCAT CONSOLE COPY COS
DELETE EOF EOLN EXIT EXP
FALSE FILLCHAR GET GOTOXY HALT
INPUT INSERT INTEGER INTERACTIVE IORESULT
KEYBOARD LENGTH LN LOG MARK
MAXINT MEMAVAIL MOVELEFT MOVERIGHT NEW
ODD OPENNEW OPENOLD ORD OUTPUT
PAGE POS PRED PRINTER PUT
PWROFTEN READ READLN REAL RELEASE
REMOTE RESET REWRITE ROUND SCAN
SEEK SEMAPHORE SEMINIT SIGNAL SIN
SIZEOF SQR SQRT START START
STR STRING SUCC TEXT TRUE
TRUNC UNITBUSY UNITCLEAR UNITREAD UNITWAIT
UNITWRITE WAIT WRITE WRITELN

ANNEXE II

L'utilitaire LIBMAP permet de visualiser le contenu de toutes les librairies et en particulier celui de SYSTEM.LIBRARY qui est la librairie système fournie sur la disquette du Pascal UCSD.

```
------------------------------------------------------------------------
 LIBRARY MAP FOR SYSTEM.LIBRARY

    BUS INFORMATIQUE (DISKTO7) LE 2 AVRIL 1986


Segment # 0:  OUTERLEV  completely linked segment

    ------------------------------------------------------------------

Segment # 1:  DECOPS    separate procedure segment
   DECOPS   separate proc  P #1

    ------------------------------------------------------------------

Segment # 2:  PASCALIO  separate procedure segment
```

```
   FSEEK    separate proc  P #1
   FREADREA separate proc  P #2
   FREADDEC separate proc  P #4
   FWRITERE separate proc  P #3
   FWRITEDE separate proc  P #5

    ---------------------------------------------------------------------

Segment # 3:  TRANSCEN  library unit

FUNCTION SIN(X:REAL):REAL;
FUNCTION COS(X:REAL):REAL;
FUNCTION EXP(X:REAL):REAL;
FUNCTION ATAN(X:REAL):REAL;
FUNCTION LN(X:REAL):REAL;
FUNCTION LOG(X:REAL):REAL;
FUNCTION SQRT(X:REAL):REAL;




    ---------------------------------------------------------------------

Segment # 4:  UTILS     library unit


 TYPE BYTE=0..255;

ITEM=(ZERO,UN);

COULEUR=0..15;

     COULPOINT=-16..15;

DIRECTION=(CENTRE,NORD,NORD_EST,
EST,SUD_EST,SUD,SUD_OUEST,
OUEST,NORD_OUEST);

ZONE=PACKED ARRAY[0..79] OF CHAR;


 PROCEDURE POKE(ADRESSE,VALEUR:INTEGER);

 FUNCTION PEEK(ADRESSE:INTEGER):BYTE;

                                                                  FUNCTION
FORMAT(UNITE,ENTRELACEMENT:INTEGER;UNEFACE:BOOLEAN):BOOLEAN;
```

```
 FUNCTION PTRIG:BOOLEAN;

 PROCEDURE INPEN(VAR X,Y:INTEGER);

 PROCEDURE LINE(X,Y:INTEGER);

 PROCEDURE PSET(X,Y:INTEGER);

 PROCEDURE PENCOLOR(COLORI:COULPOINT);

 FUNCTION POINT(X,Y:INTEGER):COULPOINT;

 PROCEDURE COLOR(FORME,FOND:COULEUR);

 PROCEDURE SCREEN(FORME,FOND,TOUR:COULEUR);

 PROCEDURE INVERSE(LOCAL:BOOLEAN);

 FUNCTION CHARACTER(X,Y:INTEGER):CHAR;

 FUNCTION STICK(WHICH:ITEM):DIRECTION;

 FUNCTION STRIG(WHICH:ITEM):BOOLEAN;

 PROCEDURE PLAY(STR:STRING);

 PROCEDURE INCRUST(FLAG:BOOLEAN);

PROCEDURE
SETPAL(NUMCOUL:COULEUR;VALEUR:INTEGER;INCRUST:BOOLEAN);

 FUNCTION READPAL(NUMCOUL:COULEUR):INTEGER;

 FUNCTION MTRIG(WHICH:INTEGER):BOOLEAN;

 PROCEDURE INMOUSE(VAR X,Y:INTEGER);



    ----------------------------------------------------------------

Segment # 5:  APOINT    separate procedure segment
   APOINT   separate proc  P #1
   ACHARACT separate proc  P #2
   ASTICK   separate proc  P #3
   ASTRIG   separate proc  P #4
   APOKE    separate proc  P #5
```

```
   APEEK    separate proc  P #6
   APTRIG   separate proc  P #7
   APSET    separate proc  P #8
   AINPEN   separate proc  P #9
   ALINE    separate proc  P #10

    ---------------------------------------------------------------------

Segment # 6:  AFORMAT   separate procedure segment
   AFORMAT  separate proc  P #1
   ASETPAL  separate proc  P #2
   AREADPAL separate proc  P #3
   AINCR    separate proc  P #4
   ATO9INV  separate proc  P #5
   AWIDTH   separate proc  P #6
   APLAY    separate proc  P #7
   AMTRIG   separate proc  P #8
   AINMOUSE separate proc  P #9

    ---------------------------------------------------------------------

Segment # 7:  EXTRA     library unit


 const coln80=25447{$6367};
xxxxx=24993{$61A1};
yyyyy=24995{$61A3};
      extra_err=25389{$632D};

 type ex_typtrace=(normal,transparent,inversion,trace3,trace4,trace5,
trace6,trace7);

ex_teinte=(ex_fd_orange,ex_ciel_bleu,
ex_fd_parme,ex_fnd_bleu_clair,ex_fd_sable,
ex_fnd_vert_clair,ex_fd_rose,ex_fd_gris,
ex_fd_blanc,ex_fd_cyan,ex_fd_magenta,
ex_fd_bleu,ex_fd_jaune,
ex_fd_vert,ex_fd_rouge,ex_fd_noir,
ex_noir,ex_rouge,ex_vert,ex_jaune,
ex_bleu,ex_magenta,ex_cyan,
ex_blanc,ex_gris,ex_rose,
ex_vert_clair,ex_sable,
ex_bleu_clair,ex_fm_parme,
ex_blue_ciel,ex_orange);
     ex_byte=0..255;
     typattern=packed array[0..3] of integer;

 var pat:typattern;
```

```
procedure resetc;

procedure extradxy(d,x,y:integer);

function ex_error:integer;

procedure ex_poke(adresse,valeur:integer);

function ex_peek(adresse:integer):ex_byte;

procedure ex_dpoke(adresse,valeur:integer);

procedure window(xl,yb,xr,yt:integer);

procedure choix(tratyp:ex_typtrace;avec:boolean;couleur1:ex_teinte);

procedure paint(x,y:integer);

procedure fill(plein:boolean);

procedure pattern(patt:typattern;adr_physique:integer);

procedure psetxy(abs1,ord1:integer);

procedure linexy(destx,desty:integer);

procedure box(xl,yb,xr,yt:integer);

procedure circle(xcentre,ycentre,rayon:integer);

procedure ellipse(xcentre,ycentre,axeh,axev:integer);

procedure portion(camflg:boolean;angle_depart,angle_fin:real);

procedure initortue(adr_tortue:integer);

procedure show(visible:boolean;adr_tortue:integer);

procedure fwd(longueur,adr_tortue:integer);

procedure head(angle,adr_tortue:integer;absolu:boolean);

procedure rot(angle,adr_tortue:integer;absolu:boolean);

procedure zoom(agrandissement,adr_tortue:integer;absolu:boolean);

procedure trace(baisse:boolean;adr_tortue:integer);
```

```
 procedure anime(rapide:boolean;adr_tortue:integer);
```

Segment # 8: ARESETC   separate procedure segment
   ARESETC separate proc P #1
   ACOMSLOT separate proc P #2
   APATTERN separate proc P #3
   AEXTRADX separate proc P #4
   ASTOD   separate proc P #5
   EXPMAN   separate proc P #6
   SGNEXP   separate proc P #7
   AMOVELEF separate proc P #8
   APOKE   separate proc P #9
   APEEK   separate proc P #10

Segment # 9: MATHS   library unit

```
 const notascient=1;
sgn_apr_nb=4;
force_plus=8;
etoiles=32;
using=128;

 type d_real = record
facexp,facho,facmo,faclo,
dfacho,dfacmh,dfacml,dfaclo,
facsgn:0..255;
 end;

 procedure mresetc;

 procedure mextradxy(d,x,y:integer);

 function mex_error:integer;

 procedure d_readreal(var dreal1:d_real);

 procedure print_using(pumask,avant_pt,apres_pt:integer);

 procedure d_writereal(d_real1:d_real);
```

```
procedure str_to_d_real(s:string;var dreal1:d_real);

procedure s_to_d_real(reel1:real;var dreal1:d_real);

function d_to_s_real(dreel:d_real):real;

procedure d_addreal(re1,re2:d_real;var d_somme:d_real);

procedure d_subreal(re1,re2:d_real;var d_difference:d_real);

procedure d_mulreal(re1,re2:d_real;var d_produit:d_real);

procedure d_divreal(re1,re2:d_real;var d_quotient:d_real);

procedure d_sin(arg:d_real;var d_result:d_real);

procedure d_cos(arg:d_real;var d_result:d_real);

procedure d_atan(arg:d_real;var d_result:d_real);

procedure d_tan(arg:d_real;var d_result:d_real);

procedure d_ln(arg:d_real;var d_result:d_real);

procedure d_exp(arg:d_real;var d_result:d_real);
```

Segment #10: SCREENHA library unit

```
 TYPE
sc_chset        = SET OF CHAR;

sc_misc_rec     = PACKED RECORD
height, width : 0..255;
                  can_break, slow, xy_crt, lc_crt,
can_upscroll, can_downscroll : BOOLEAN;
                END;

sc_date_rec     = PACKED RECORD
month : 0..12;
day :   0..31;
year :  0..99;
```

```
END;

sc_info_type    = PACKED RECORD
sc_version : STRING;
sc_date : sc_date_rec;
  spec_char : sc_chset; {Characters not to echo}
  misc_info : sc_misc_rec;
END;

sc_long_string  = STRING[255];

sc_scrn_command = (sc_whome, sc_eras_s, sc_erase_eol, sc_clear_lne,
sc_clear_scn, sc_up_cursor, sc_down_cursor,
sc_left_cursor, sc_right_cursor);
sc_key_command  = (sc_backspace_key, sc_dc1_key, sc_eof_key, sc_etx_key,
sc_escape_key, sc_del_key, sc_up_key, sc_down_key,
sc_left_key, sc_right_key, sc_not_legal);

    sc_choice       = (sc_get, sc_give);

{ Textport handling facilities }

sc_status_set   = (tx_open, tx_closed); { indicates status of a txport }

sc_tx_port      = RECORD
row, col,            { screen relative}
height, width,       { size of txport (zero based)}
cur_x, cur_y  : INTEGER;
{ cursor positions relative to the txport }
status       : sc_status_set;
END;

{Types from here on are temporarily here, do not use !!}

sc_tx_ptr       = ^sc_tx_array; { points to a dynamic txport array }

sc_tx_array     = RECORD                { element of the dynamic array }
port_num : INTEGER;   { txport identifying number }
link    : sc_tx_ptr;
{ link to the next element of the array }
specs   : sc_tx_port;  { txport specifications }
END;

 VAR  {Vars are public only temporarily, do not use}
sc_tx_root   : sc_tx_ptr;    { root of the dynamic array }
sc_tx_point  : sc_tx_ptr;    { points to the currently active txport }

 PROCEDURE sc_use_info (do_what : sc_choice; VAR t_info : sc_info_type)
```

```
 PROCEDURE sc_erase_to_eol (x, line : INTEGER);
 PROCEDURE sc_init (just_booted : BOOLEAN);
 PROCEDURE sc_left;
 PROCEDURE sc_right;
 PROCEDURE sc_up;
 PROCEDURE sc_down;
 PROCEDURE sc_getc_ch (VAR ch : CHAR; return_on_match : sc_chset);
 PROCEDURE sc_clr_screen;
 PROCEDURE sc_clr_line (y : INTEGER);
 PROCEDURE sc_home;
 PROCEDURE sc_eras_eos (x, line : INTEGER);
 PROCEDURE sc_goto_xy (x, line : INTEGER);
 PROCEDURE sc_clr_cur_line;
 FUNCTION  sc_find_x : INTEGER;
 FUNCTION  sc_find_y : INTEGER;
 FUNCTION  sc_scrn_has (what: sc_scrn_command) : BOOLEAN;
 FUNCTION  sc_has_key (what:  sc_key_command) : BOOLEAN;
 FUNCTION  sc_map_crt_command(VAR k_ch: CHAR): sc_key_command;
 FUNCTION  sc_prompt (line : sc_long_string; x_cursor, y_cursor, x_pos,
where : INTEGER; return_on_match : sc_chset;
no_char_back : BOOLEAN; break_char : CHAR) : CHAR;
 FUNCTION sc_check_del (inp : CHAR; VAR s_inx : INTEGER; VAR str : STRING)
: BOOLEAN;
```
