T A S C

INTRODUCTION.

Ce programme permet de compiler un programme écrit en APPLESOFT, ce qui permet d'en optimiser la vitesse d'exécution.

Le compilateur supporte le langage APPLESOFT avec quelques exceptions.

Les avantages fournis par TASC sont :

1. Vitesse d'exécution augmentée.
   Les programmes APPLESOFT compilés sont exécutés de deux à vingt fois plus rapidement que le même programme en interpréteur.

2. Communication inter-programmes.
   Les programmes peuvent inter-communiquer avec l'utilisation de variables communes.

3. Arithmétique entière réelle.

4. Sécurité du code source.
   TASC crée un langage machine équivalent au programme APPLESOFT. Cette file en langage machine se trouve protégée contre toute copie.

5. Compilation basee sur disquette.
   Contrairement aux autres compilateurs, TASC écrit le programme en langage machine sur disquette lors de la compilation, ce qui permet de compiler un grand programme.

1.1 COMMENT UTILISER CE MANUEL.

Les trois premiers chapîtres de ce manuel expliquent le système de compilation. Les chapîtres suivants décrivent techniquement le TASC et son utilisation.

Chapitre 1. Introduction - Explique brièvement TASC.

Chapitre 2. Démonstration - Explique la compilation et exécute un programme de démonstration.

Chapître 3. Introduction à la compilation - Donne une introduction au vocabulaire du compilateur, une comparaison entre interpréteur et compilateur, et un aperçu du développement d'un programme avec le compilateur.

Chapître 4. Dépannage avec l'interpréteur - Décrit comment dépanner la file source avec l'interpréteur de l'APPLESOFT avant de le compiler.

Chapître 5. Compilation - Décrit l'utilisation de TASC en détail, ainsi que les diverses options.

Chapître 6. Exécution d'un programme compilé - Décrit comment faire tourner une file objet compilée.

Chapître 7. Comparaison entre l'interpréteur et le compilateur - Ce chapître est à lire avant de tenter de compiler un programme.

Chapître 8. Amélioration du langage - Décrit les extensions qu'apportent TASC à l'APPLESOFT.

Chapître 9. Comment travaille le compilateur.

Chapître 10. Messages d'erreurs et dépannage - Décrit chaque message d'erreur et discute des problèmes communs.

Appendices :

A. Déplace des files binaires avec l'utilitaire ADR - Comment BLOAD et BSAVE des files binaires avec l'utilitaire ADR.

B. Copie de TASC et conversion en DOS 3.3 - Comment copier et convertir le compilateur TASC.

C. Création de disque automatique.

D. Notes sur APPLESOFT.

E. Utilisation de la mémoire par les programmes compilés.

F. Utilisation de la page zéro par les programmes compilés.

Ce manuel présume que l'utilisateur a une bonne connaissance du langage APPLESOFT.

1.2 CONTENU DU PROGRAMME TASC.

1. TASC - Le compilateur APPLESOFT.

2. PASS0,PASS1,PASS2 - Composants internes de TASC.

3. RUNTIME - Librairie de routines en langage machine utilisées par les programmes compilés.

4. ADR - Utilitaire pour files binaires.

5. CREATE ADR - Utilitaire pour créer ADR sur une autre disquette.

6. BALL - Programme de démonstration.

2. DEMONTRATION.

IMPORTANT

Avant de commencer cette démonstration, il est fortement conseillé de faire une copie de cette disquette et de mettre la disquette maitre de côté. S'il est nécessaire de transformer cette disquette en DOS 3.3 , utiliser le programme MUFFIN. Cette procédure est expliquée à l'appendix B.

Ce chapitre fournit pas à pas les instructions pour utiliser TASC. Nous vous recommandons fortement cette démonstration avant de compiler un programme.

Premièrement, bootez la disquette et tapez :

```
RUN TASC
```

Puis répondez aux questions posées :

```
Source file? BALL

objet code file:
(default ball.obj)? <RETURN>
```

La file source est un programme APPLESOFT nommé BALL qui existe déjà sur la disquette. La file objet est en langage machine créée par le compilateur. La file originale est BALL.OBJ.

La file source est supposée être sur la meme disquette que le compilateur, sinon il faudrait le spécifier. Les différents connecteurs ou drives peuvent être spécifiés en utilisant S<numéro de connecteur> et D<numéro de drive>. La compilation est plus rapide si un seul drive est utilisé. Les commandes DOS peuvent être exécutées en tapant <CTRL-D> suivi de <RETURN>.

```
MEMORY USAGE:
DEFAULT CONFIGURATION? <RETURN>

OPTIONS:
DEFAULT CONFIGURATION? <RETURN>
```

Si vous refusez les configurations par défaut ci-dessus, vous devrez expliciter spécifiquement la valeur de plusieurs options de compilation. Ces options sont expliquées au chapitre 5.

Lorsque la compilation commence, le drive est accédé presque constamment soit pour lire la file source, soit pour écrire la file objet. Pendant la compilation, le compilateur liste sur

l'écran le programme source et génère les messages appropriés si des erreurs sont rencontrées. Quand le listing du programme source s'arrête, la première partie de la compilation est terminée et le compilateur affiche :

*****BEGINNING PASS2.

La seconde partie de la compilation utilise le drive davantage. Pour indiquer le processus de compilation, le compilateur affiche un point de temps en temps. Lorsque c'est terminé, il affiche :

*****CODE GENERATION COMPLETE

A ce point, le processus de compilation est terminé. Répondre 'Y' à la question suivante:

```
COMPILATION INFORMATION AND LINE NUMBER
REFERENCE TABLE? YES
```

Cette entrée accepte aussi la commande drive <CTRL=D>. Si vous voulez la liste de compilation sur une imprimante, vous pouvez le faire en entrant:

<CTRL=D>PR#<numéro du connecteur de l'imprimante>

TASC imprime alors l'information désirée et revient à l'interprèteur et affiche le message:

*****COMPILATION COMPLETE

L'augmentation de la vitesse d'exécution du programme BALL est très apparent comparé au même programme exécuté sous APPLESOFT. Comparez les vitesses en lançant le programme interprèté:

```
RUN BALL
```

Puis exécutez le programme compilé:

```
BLOAD RUNTIME
BRUN BALL.OBJ
```

Notez que la librairie RUNTIME doit être BLOAD en mémoire avant que BALL.OBJ puisse être BRUN.

Vous avez maintenant réussi la démonstration complète. Lisez bien le chapitre 7 avant de tenter de compiler un autre programme APPLESOFT.

3. INTRODUCTION A LA COMPILATION.

3.1 VOCABULAIRE.

FILE SOURCE - Le programme APPLESOFT est appelé file source car c'est depuis cette file que le programme en langage machine est créé. Le CATALOG liste les noms des programmes sources APPLESOFT avec la lettre A précédant la taille de chaque file.

FILE OBJET - TASC translate les files sources en files objets en langage machine. Le CATALOG liste les noms des files objets de TASC avec la lettre B précédant la taille de chaque file.

TEMPS DE COMPILATION - Temps mis par le compilateur pour translater une file source en file objet.

TEMPS D'EXECUTION - Temps mis par un programme compilé à s'exécuter.

LIBRAIRIE RUNTIME - Collection de routines en langage machine utilisées par le programme compilé. RUNTIME doit être chargé en mémoire avant l'exécution d'une file objet.

3.2 DIFFERENCES ENTRE COMPILATEUR ET INTERPRETEUR.

Le micro-processeur de l'APPLE n'exécute que ses propres instructions machine. Il n'exécute donc pas un programme APPLESOFT directement. Chaque instruction doit être simulée par des routines écrites en langage machine qui permettent l'exécution de chaque instruction BASIC.

Supposiez que vous ayez un livre en Anglais, langue que vous ne connaissez pas du tout, pour le traduire, vous aurez besoin d'un dictionnaire Français-Anglais. En APPLESOFT, c'est pareil, il doit traduire ses instructions.
Si ce livre est écrit en Français, vous le lirez plus facilement et plus rapidement. C'est ce qui se passe avec le compilateur.

3.3 PROCESSUS DE DEVELOPPEMENT DU PROGRAMME.

Cela commence avec la création d'un programme source APPLESOFT. Une fois le programme terminé, il doit tourner correctement, sinon le dépanner en utilisant l'interprèteur APPLESOFT. Etant donné que la syntaxe de l'interprèteur et du compilateur sont très similaires, un programme correct en interprèteur doit être correct en compilateur.
Ensuite, ce programme peut être compilé. Si la compilation est réussie, le compilateur fournit une file objet qui peut être exécutée comme un programme en langage machine. Si le compilateur détecte une erreur, le processus revient en mode APPLESOFT.

4. DEPANNAGE AVEC L'INTERPRETEUR D'APPLESOFT.

1. Création d'un programme source.

2. Lancement du programme sous interpréteur pour rechercher et corriger les erreurs.

4.1 CREATION D'UN PROGRAMME SOURCE.

Le programme doit être écrit comme un programme APPLESOFT habituel, puis il peut être sauvé sur disquette avec SAVE. TASC peut seulement compiler des files APPLESOFT sur disquette.

4.2 LANCEMENT DU PROGRAMME AVEC APPLESOFT.

Le programme doit tourner correctement en APPLESOFT avant d'être compilé. Si le programme utilise des possibilités de TASC non disponibles dans l'interpréteur, il sera nécessaire de dépanner le programme avec le compilateur, voir chapître 8.

La compilation permet de retrouver des erreurs qui passeraient inaperçues à l'interpréteur, toutefois, les erreurs de logiques sont plus facilement corrigeables avec l'interpréteur.

5. COMPILATION.

Ce chapître explique plusieurs aspects techniques de la compilation.

1. Options.
2. Terminer la compilation.
3. Compilation d'un grand programme.

5.1 OPTIONS.

La démonstration montrait seulement un programme à compiler. TASC inclue plusieurs options qui peuvent être utilisées pour mieux contrôler la compilation. Pour spécifier les valeurs de ces options, taper 'NO' lorsque le compilateur offre les valeurs par défaut.

5.1.1 Utilisation de la mémoire.

La mémoire utilisée par le fonctionnement du compilateur est divisée en trois zones:

1. Librairie de fonctionnement.
2. Programme objet.
3. Variables.

TASC permet l'adressage séparé de chacun de ces blocs. Les possibilités d'adressage mémoire peuvent être utilisées pour protéger des programmes en langage machine, des tables de formes, ou d'autres parties importantes de la mémoire.

L'ordre d'emplacement par défaut de ces blocs est : librairie, programme, variables. La librairie est située en bas de la mémoire, suivi du programme et des variables. La librairie commence à l'adresse 2051, ou $803. La configuration de la mémoire par défaut est :

haut
- espace chaine de caractères
- variables
- code objet

RUNTIME (librairie)

bas

Les adresses de ces blocs sont simples à spécifier. La nouvelle adresse du bloc librairie doit être entrée comme un nombre, $803 par défaut. Les adresses peuvent être spécifiées en hexadécimal ou en décimal. Les adresses hexadécimales doivent etre précédées du signe dollar ($).

La librairie doit être BLOAD pour que le programme compilé puisse tourner. Elle est chargée en $803 par défaut. A une adresse différente, la librairie doiit être chargée à l'adresse correcte en utilisant l'option 'A' avec la commande BLOAD. Voir appendix A pour davantage d'informations sur le sauvetage et chargement du code objet binaire et de la file RUNTIME.

L'adresse de départ du code objet doit être spécifiée par :

1. Le mot HGR1
2. Le mot HGR2
3. Un nombre décimal ou hexadécimal
4. <RETURN>

HGR1 et HGR2 place simplement le début du programme au-dessus de l'écran haute-résolution approprié. Les 4 K de la librairie RUNTIME sont placés par défaut au-dessous de la première page graphique haute-résolution. Cet emplacement par défaut de la librairie est suggéré pour un programme utilisant le graphique haute-résolution. Une adresse absolue décimale ou hexadécimale peut aussi être spécifiée, mais se méfier de ce qu'il peut détruire en travaillant. Taper <RETURN> pour placer le code objet à la fin de la librairie par défaut.

L'espace des variables peut être spécifié explicitement ou alloué par défaut. L'espace des variables commence à la fin du code objet par défaut.

Les programmes compilés utilisent le pointeur HIMEM normal pour déterminer le sommet de l'espace disponible pour les chaînes de caractères. Le bas de cet espace est placé, ainsi le bloc qui est en haut de la mémoire est protégé. En conséquence, l'ordre normal par défaut (librairie RUNTIME, programme objet, variables) place le bas du stockage des chaines de caractères à la fin de l'espace des variables. Le fait de spécifier qu'un autre bloc réside en haut de la mémoire place le bas du stockage des chaine des caractères à la fin de ce bloc.

5.1.2 Options de compilation.

Il existe cinq options de compilation qui peuvent être spécifiées avant que le processus de compilation commence.

Ces options suivantes sont par défaut :

OPTION DE COMPILATION	DEFAUT
Listing de compilation	YES
Pause sur erreur	YES
Arithmétique entière	YES
Constantes entières	YES
RESUME/code dépannage	NO

Répondre par 'NO' ou 'N' aux questions dont la réponse est 'YES' si vous voulez modifier l'entrée par défaut et par 'YES' ou 'Y' à la dernière question.

Option listing de compilation.

Le compilateur liste normalement la file source. En répondant 'NO' à cette option, il n'y aura pas de listing. Les erreurs, précautions, et messages spéciaux seront affichés comme d'habitude.

Option pause sur erreur.

Normalement, les erreurs stoppent la compilation et permettent à l'utilisateur d'arrêter ou de continuer la compilation. En répondant 'NO' à cette option, la pause sera supprimée et aucun message d'erreur ne sera affiché.

Option arithmétique entière.

L'arthmétique entière permet les opérations sur des entiers en un temps inférieur à la moitié du temps normal. Cette option augmente la vitesse du programme qui utilise les entiers, mais il y a quelques limitations. Voir chapitre 8.

Constantes entières.

Dans un programme compilé, les constantes peuvent être traitées comme des entiers ou comme des nombres à virgule

flottante. La sélection de l'option constantes entières permet de les stocker sous le format entier. Si une constante est aussi nécessaire sous forme virgule flottante, le compilateur inclue les deux formes dans le code compilé. La conversion de constante en fonctionnement est totalement éliminée. L'option constantes entières augmente la vitesse d'exécution du programme.

Les constantes entières prennent deux octets dans la file objet; la représentation en virgule flottante en requiert cinq. L'emploi de constantes entières diminue la taille du code objet, alors que, bien sûr, l'emploi de constantes réelles et entières mélangées l'augmente de façon non négligeable.

RESUME/OPTION DE DEPANNAGE DU CODE.

Cette option permet d'inclure l'instruction RESUME dans le programme objet. L'instruction RESUME, en APPLESOFT, permet, en cas d'erreur, de ne pas stopper l'exécution du programme, mais de revenir sur l'instruction qui a causé l'erreur dès que le programme est terminé. Comme l'APPLESOFT, TASC supporte pleinement l'instruction ONERR GOTO. Le mécanisme est identique à celui de l'APPLESOFT.

Cette option requiert que le compilateur génère un extra code au début de chaque instruction suceptible de générer une erreur. La sélection de cette option entraîne une augmentation de la taille du programme et une diminution de la vitesse d'exécution.

Cette option présente l'avantage d'inclure l'adresse du code objet dans tous les messages d'erreur. Normalement, quelques erreurs générées par la librairie RUNTIME comportent l'adresse du code objet. Cette option peut être utilisée pour dépanner avec le compilateur. Toutefois, ceci diminue la vitesse d'exécution et augmente la longueur du code compilé. Si cette option n'est pas choisie, l'instruction RESUME sera ignorée.

5.2 FIN DE COMPILATION.

Le compilateur tourne en langage machine, donc <CTRL-C> est ignoré et doit être remplacé par <RESET>. Toutefois, le compilateur accède souvent au drive, aussi est-il préférable de ne pas utiliser <RESET>. Pour solutionner ce problème, le commpilateur cherche occasionnellement si un <CTRL-C> a été tapé, et dans ce cas termine la compilation. Le compilateur peut également être stoppé en tapant <CTRL-C> comme premier caractère en réponse à un INPUT, mais ceci ne stoppe pas correctement la compilation.

Puisque l'arrêt de la compilation laisse la file objet incomplète, TASC annule la file objet si la compilation est annulée. TASC modifie le DOS, ainsi, l'utilisation de <RESET> ou de <CTRL-C> dans une entrée laisse le DOS dans un état modifié.

Le DOS doit être rechargé si le compilateur est stoppé anormalement. Le fait de taper <CTRL-C> hors d'une entrée est la seule manière correcte de terminer la compilation. Lorsqu'il est est correctement stoppé, le compilateur remet le DOS dans son état normal.

5.3 COMPILATION D'UN GRAND PROGRAMME.

Le code objet compact, l'utilisation basée sur la disquette et les variables communes permettent déjà de compiler des programmes de bonne taille sans modification. Toutefois, si un programme est simplement trop important pour être compilé ou pour tenir en mémoire disponible, des mesures spéciales doivent être prises.

5.3.1 Réduction de l'espace de la table des symboles.

La table des symboles du compilateur doit contenir les informations sur les variables, les fonctions, les constantes, les numéros des lignes de référence, et attributions communes.

La table des symboles est examinée avant chaque nouvelle entrée et cet examen prévient la duplication de l'information. La première utilisation d'une variable requiert que le compilateur crée une nouvelle entrée, mais les références suivantes ne requièrent aucun espace additionnel. Similairement, bien qu'initialement un numéro de ligne référencée dans une instruction de branchement crée une entrée pour la ligne, des références multiples ne requièrent aucune entrée supplémentaire.

Les fonctions requièrent quatre emplacements dans la table des symboles, et les numéros de lignes référencées en requièrent trois. Les variables simples requièrent cinq emplacements et les variables de tableau en requièrent six plus une pour chaque dimension. Les variables communes requièrent deux emplacements supplémentaires par variable. En pratique, l'élimination des fonctions, lignes référencées ou variables est difficile mais possible.

Les très longs programmes qui dépassent l'espace de la table des symboles donnent le message d'erreur "SYMBOL TABLE FULL" durant la compilation. Il existe plusieurs solutions à ce problème.

La solution la plus simple est de ne pas choisir l'option constantes entières. En effet, avec cette option, les constantes sont initialement stockées comme des entiers. Si plus tard, elles doivent être utilisées comme nombres à virgule flottante, elles seront converties et entrées dans la table des symboles comme constantes à virgule flottante. L'entrée comme entier initiale occupe cinq emplacements. Si l'entrée en virgule

flottante additionnelle est requise, elle occupe huit emplacements.

Sans cette option, les constantes sont stockées sous forme virgule flottante, ce qui sauve cinq emplacements par variable. Par contre, ceci diminue la vitesse d'exécution, donc est à utiliser le moins possible. Si le message d'erreur subsiste, il faut séparer le programme en deux parties.

5.3.2 Séparation d'un programme en plusieurs parties.

Quand le code objet d'un programme ne tient pas dans la mémoire disponible ou dont l'espace de la table de symboles est trop important, le programme peut souvent etre séparé en deux programmes plus petits. On peut utiliser la disquette comme file commune où le premier programme rentre des données et où le second programme les lit.

Tasc permet également d'utiliser des variables communes. Les variables sont simplement déclarées instructions communes dans les deux programmes et le compilateur alloue le stockage et peut sauver aussi les valeurs sur disquette si nécessaire. C'est la technique que TASC utilise pour communiquer entre ses trois parties : PASS0,PASS1, et PASS2. Voir chapître 8 pour plus d'informations.

6. EXECUTION D'UN PROGRAMME COMPILE.

Bien qu'exécuté de façon identique à un programme en APPLESOFT et chargé comme file binaire, un programme compilé nécessite un mécanisme de chargement et de mise en route différent.

1. Un programme interprèté est stocké comme file APPLESOFT et est indiqué par la lettre 'A' dans le catalogue de la disquette. Il est lancé en tapant RUN <nom de file>. Au contraire, un programme compilé est stocké comme file binaire et est indiqué par la lettre 'B' dans le catalogue de la disquette.

Puisque le programme compilé n'est plus une file APPLESOFT, il ne peut plus être mis en route par la commande RUN qui donnerait le message FILE TYPE MISMATCH, et doit donc être lancé en tapant

BRUN <nom de file>

Quand un programme est BRUN, la librairie RUNTIME doit déjà être en mémoire. Si elle y était déjà avec le programme précédent, elle n'a pas besoin d'être rechargée.

La séquence normale pour exécuter un nouveau programme compilé est :

```
BLOAD RUNTIME
BRUN <nom de file>
```

Un programme compilé ne peut être exécuté que si l'interprèteur APPLESOFT est en mémoire et ne peut travailler avec le BASIC ENTIER.

2. Utilisation de l'ampersand (&) - Une fois que le programme compilé a été chargé et exécuté, il peut être réexécuté en tapant un '&' suivi d'un RETURN. Le programme compilé utilise le '&' comme vecteur pour pointer au début du code objet lors de l'exécution, ainsi ce vecteur peut être utilisé aussi longtemps que le dernier programme tourne. L'utilisation de ce vecteur est par ailleurs plus facile d'emploi que l'utilisation de CALL puisqu'avec cette commande, il faudrait connaitre l'adresse de départ du code objet.

3. Arrêt d'exécution d'un programme compilé - Puisque le code compilé est exécuté directement et n'est plus sous la supervision de l'interprèteur, <CTRL-C> ne fonctionne plus durant l'exécution du programme. Sauf en cas d'un INPUT, tout caractère tapé sera ignoré.

Toutefois, en tapant <CTRL-C> comme réponse à un INPUT, la fonction est identique à l'interprèteur, ce qui signifie que le programme compilé doit être stoppé par un <RESET>. <RESET> ne réinitialise pas correctement l'interprèteur APPLESOFT, donc une commande NEW doit être utilisée après interruption d'un

programme compilé.

4. Commande NEW - Cause la réinitialisation des pointeurs de l'interprèteur, mais n'efface pas le programme en mémoire. En conséquence, un programme compilé peut être réexécuté si aucune ligne de programme n'a été tapée et stockée dans l'espace du programme.

5. Commandes immédiates. Le code compilé ne tient pas une liste des variables, donc l'interprèteur ne peut pas trouver la valeur des variables utilisées dans un programme compilé. L'exécution d'un programme compilé qui utilise la variable A et le fait de taper la commande immédiate PRINT A retourne une valeur qui n'a aucune relation avec la variable A utilisée par le programme compilé.

7. COMPARAISON ENTRE LE LANGAGE INTERPRETE ET COMPILE.

Ce chapître décrit les différences qui doivent être prises en compte lors de la compilation d'un programme. Si un programme compilé ne tourne pas correctement, voir le chapître 10 pour plus d'information.

7.1 INSTRUCTIONS NON EXECUTEES.

La plupart des instructions APPLESOFT sont exécutées sans modification. Les instructions suivantes ne sont pas inclues dans TASC :

CONT	DEL	LIST	LOAD
LOMEM:	NOTRACE	RECALL	SAVE
SHLOAD	STORE	TRACE	&

7.2 INSTRUCTIONS SUPPORTEES AVEC DES LIMITATIONS.

DEF FN DIM <CTRL-C>

7.2.1 Utilisation de DEF FN.

TASC et APPLESOFT permettent la définition d'un argument simple, fonction arithmétique réelle avec l'instruction DEF FN. En plus, l'interprèteur permet de redéfinir les fonctions en utilisant le meme nom de fonction. TASC ne supporte pas cette redéfinition.

Dáns l'interprèteur, un DEF FN est dèfini du commencement à la fin du programme et peut se trouver à n'importe quel endroit. Le compilateur cherche ces définitions à la compilation, par conséquent, il ne peut y avoir qu'une définition par variable.

S'il y a plusieurs définitions de même nom, le compilateur sort un message d'erreur '?UNDEF'D FUNCTION'. De plus les fonctions devenues indéfinies sont signalées durant la compilation.

7.2.2 L'instruction de DIMemsionnement.

L'interprèteur fournit trois méthodes de dimensionnement d'un tableau :

1. Dimensionnement constant - Exécution d'une instruction de dimensionnement dans laquelle les dimensions spécifiées occupent le même espace pour le tableau chaque fois que le programme tourne.

2. Dimensionnement dynamique - Exécution d'une instruction de dimensionnement dans laquelle les dimensions spécifiées sont des expressions arthmétiques occupant un espace de stockage dépendant du calcul de ces expressions.

3. Dimensionnement par défaut - Si un tableau est rencontré avant une instruction de dimensionnement, le tableau est donné pour une valeur de 10 maximum par défaut pour chaque dimension. Exemple : un tableau à trois dimensions produit (10,10,10). Etant donné que l'APPLESOFT permet l'utilisation du 0, le tableau se trouve réellement dimensionné à 11 éléments.

Le compilateur ne supporte pas le dimensionnement dynamique des tableaux. Toute instruction de dimensionnement doit comporter des constantes entières, et non en virgule flottante, ni des expressions arthmétiques. Les instructions de dimensionnement peuvent se situer n'importe où dans le programme, mais doivent toujours précéder le tableau dimensionné. Les tableaux peuvent être dimensionnés plusieurs fois, mais toujours les dimensions spécifiées doivent être identiques. Un tableau non dimensionné reçoit la dimension de 10 par défaut pour chaque élément souscrit.

Une erreur est générée si un tableau est référencé avec un nombre d'éléments différent du nombre utlisé en premier pour la même variable.

Seule la taille mémoire limite la dimension des tableaux.

7.2.3 Utilisation de <CTRL-C> pour stopper un programme compilé

L'APPLESOFT permet l'utilisation de <CTRL-C> pour stopper un programme. En plus, le fait de taper <CTRL-C> en réponse à un INPUT suivi d'un RETURN stoppe également le programme.

Le code compilé ne recherche pas le <CTRL-C> durant l'exécution. Il ne peut etre stoppé que par <RESET>. Une commande NEW est alors nécessaire pour réinitialiser l'APPLESOFT. Voir chapître 6.

Puisque le code compilé ne recherche pas <CTRL-C> pendant l'exécution, il en supporte l'usage pendant une instruction INPUT. Le fait de taper <CTRL-C> suivi d'un RETURN en réponse à un INPUT stoppe le programme et affiche le message :
'BREAK IN ####".

7.3 AUTRES DIFFERENCES DE LANGAGE.

Quelques instructions APPLESOFT doivent être modifiées pour être exécutées correctement.

7.3.1 Utilisation des mots IF/THEN.

En APPLESOFT, l'instruction IF/THEN ne peut pas comparer directement des chaînes de caractères. Toutefois, l'emploi de telles chaînes dans cette instruction ne cause pas d'erreur, car elles sont transformées en valeurs numériques pour être testées. L'utilisation d'un IF/THEN avec une expression chaîne de caractères utilisée plus de deux à trois fois dans un programme cause l'erreur : '?FORMULA TOO COMPLEX'. De plus, la valeur numérique retournée n'est pas représentative de la chaîne.

Voici des exemples d'expressions considérées comme chaine :

```
CHR$(3)      A$+B$      STR$(I*J)
```

Les exemples suivants sont évalués numériquement et sont par conséquent des expressions numériques :

```
A$<B$+C$    A$<CHR$(2)    FLAG AND A$<C$
```

Le compilateur supporte pleinement l'utilisation de IF/THEN dans toutes ses formes avec des arguments numériques, mais un argument chaine de caractères est indiqué comme erreur durant la compilation.

7.3.2 Le GET numérique.

Normalement, l'APPLESOFT permet le GET essentiellement pour entrer un caractère sous la forme chaîne de caractères. Toutefois, il est possible de l'utiliser avec une variable numérique. Seulement, si un GET numérique reçoit une autre entrée qu'une entrée numérique, l'interpréteur donne un message d'erreur et stoppe l'exécution du programme. Le compilateur élimine ce problème en retournant la valeur 0 pour toute entrée non numérique, et ne génère pas de message d'erreur.

7.3.3 Le READ numérique.

L'APPLESOFT ne permet pas aux chaînes de caractères d'être lues dans des variables numériques. Par exemple, bien que les instructions DATA suivantes soient traitées identiquement quand elles sont lues dans des variables chaînes de caractères, la

seconde génère un '?SYNTAX ERROR' quand elle est lue dans une variable numérique :

```
10 DATA 1234
20 DATA "1234"
```

Le compilateur éiimine cet inconvénient. Toutefois, la version compilée d'un INPUT génère un message '?REENTER' quand une virgule est entrée.

7.4 DIFFERENCES OPERATIONNELLES.

7.4.1 La pile.

Les programmes compilés et interprétés utilisent une partie de la mémoire comme pile pour stocker les informations nécessaires aux instructions GOSUB/RETURN et FOR/NEXT. Les routines utilisées par le compilateur sont plus efficaces que celles de l'interpréteur. Certains programmes tournant avec des erreurs de surcharge de pile peuvent tourner sans problème une fois compilés.

Il y a 254 octets de libre pour la pile. L'entrée d'un RETURN en prend deux, et le FOR en prend 16. Il y a surcharge s'il y a plus de 127 appels par GOSUB ou plus de 15 boucles FOR/NEXT. Une partie de la pile est également utilisée par les routines PRINT et STR$. L'utilisation de ces instructions occupe les 16 derniers octets disponibles dans la pile. Donc, il faut dans la mesure du possible s'en tenir aux restrictions de l'interpréteur

7.4.2 ONERR GOTO et la pile.

L'instruction ONERR GOTO est la source de la plupart des problèmes avec la pile dans les programmes compilés et interprétés. Les problèmes ont lieu lorsque l'interpréteur interne ou une routine sort avec une condition d'erreur. La plupart des routines internes utilise une partie de l'espace de la pile pour un stockage temporaire. La sortie avec une condition d'erreur peut laisser quelques paramètres stockés dans la pile. Quand l'ONERR n'est pas en action, le programme stoppe son exécution et l'information sur la pile ne cause aucun problème.

Quand les erreurs sont tenues par ONERR GOTO, les extra-entrées restent dans la pile et le contrôle est transféré à la routine de maintenance des erreurs. Si la maintenance des erreurs se termine avec RESUME, la pile est restaurée et les extra-octets ne causent plus aucun problème. Si la routine de maintenance des

erreurs ne se termine pas par RESUME, la pile n'est pas restaurée. La répétition de ce processus provoque la mise en dépassement de la pile.

Le manuel APPLESOFT inclue un programme en langage machine qui restaure la pile. Cette routine restaure aussi la pile quand elle est utilisée avec un programme compilé. Toutefois, le code compilé ne sauve pas toujours le pointeur de pile avant chaque instruction. Quand la routine essaie de restaurer la pile, le pointeur n'est pas placé correctement. Pour prévenir ce problème, le code compilé doit sauver le pointeur de pile avant chaque instruction. L'instruction RESUME requiert également que le code compilé sauve l'information avant chaque instruction. La génération d'un extra-code RESUME/dépannage est une option de compilation, puisqu'il ralentit l'exécution et augmente la longueur du programme.
Cette option DOIT être inclue dans le programme compilé pour que la routine d'effacement de la pile du manuel APPLESOFT puisse fonctionner correctement. L'oubli d'inclure cette option provoquera le saut du programme compilé dans le monitor ou rencontrera d'autres ennuis. Bien sur, cette option ne doit pas être utilisée sans nécessité absolue.

7.4.3 Appels spéciaux au langage machine.

Certains programmes en APPLESOFT utilisent des techniques spéciales pour passer des informations en langage machine. La méthode la plus commune est d'inclure un extra-texte suivant l'appel au langage machine. Par exemple, les instructions suivantes peuvent être utilisées :

```
; USR(0)"3,5,6"
  CALL 520"PROGRAMME2"
```

Ces instructions travaillent avec l'interpréteur car le programme en langage machine peut changer le pointeur de l'interpréteur dans l'instruction courante et empêcher que les caractères additionnels soient vus. Puisque le compilateur scrute toutes les instructions à la compilation, l'extra-texte sera signalé comme erreur de syntaxe. Voir le chapître 9 pour plus d'informations à ce sujet.

7.4.4 Utilisation de MAXFILES dans un programme compilé.

La commande MAXFILES réserve le nombre d'emplacements mémoire tampons (buffers) pour les files. Le nombre de buffers disponibles détermine combien de files peuvent être ouvertes simultanément. Puisque TASC supporte pleinement toutes les commandes du DOS, la commande MAXFILES peut être utilisée dans un programme compilé. Toutefois, le DOS n'exécute pas les

opérations additionnelles pour exécuter correctement MAXFILES dans un programme compilé. Le DOS change la valeur stockée dans HIMEM, mais il n'altère pas les autres pointeurs qui doivent être changés pour qu'un programme compilé puisse être conforme à la nouvelle valeur de HIMEM.

MAXFILES, dans un programme compilé, doit être suivi par l'instruction HIMEM afin de placer tous les pointeurs correctement. Ceci peut être accompli par l'instruction suivante :

```
HIMEM:PEEK(115)+256*PEEK(116)
```

Ces deux instructions détruisant les valeurs chaines de caractères, elles doivent être employées dans les premières lignes du programme.

7.4.5 Utilisation de RUN avec COMMON.

L'instruction RUN est, en APPLESOFT, normalement utilisée comme commande immédiate depuis l'éditeur, mais peut aussi être utilisée dans un programme. RUN efface toutes les variables et réexécute le programme entier, mais avec un numéro de ligne, commence l'exécution à cette ligne.

TASC inclue également cette instruction. Sans numéro de ligne, le programme est exécuté en sautant au vecteur 'ampersand'. L'effet est identique à l'APPLESOFT. Si le programme n'a pas de variablè commune, ou a des variables spécifiées avec USECOMMON, seules les variables locales seront effacées. Si le programme spécifie DEFCOMMON, alors même les variables communes seront effacées.

L'utilisation d'un RUN<numéro de ligne> provoque un CLEAR, suivi d'un GOTO au numéro de ligne spécifié. Les variables communes ne seront jamais effacées, puisque la commande CLEAR compilée initialise seulement les variables locales. Voir chapître 8 pour plus d'informations.

7.4.6 NEW, END et STOP.

Les instructions NEW et END compilées fonctionnent comme en interpréteur. STOP imprime le message 'BREAK IN ####', mais le numéro spécifié est une adresse du code objet et non un numéro de ligne. Les trois commandes effacent tout programme interprété se trouvant en mémoire, initialisent correctement les pointeurs APPLESOFT et reviennent en interpréteur.

7.4.7 Arrêt anormal d'un programme compilé.

L'utilisation de la page zéro diffère suivant que le programme est compilé ou interprété. L'exécution d'un programme compilé commence par un appel à une routine placée en page zéro. Lorsqu'un programme compilé exécute un END, STOP ou NEW, ou est stoppé par un <CTRL-C> durant une instruction INPUT, il réinitialise la page zéro pour l'interpréteur avant de terminer l'exécution. Ceci termine l'exécution correctement.

L'utilisation de RESET pour arrêter un programme compilé ne réinitialise pas la page zéro pour l'interpréteur. Il n'est donc pas possible d'utiliser l'interpréteur à ce point. Même si les instructions paraissent fonctionner normalement, l'interpréteur peut détruire le DOS ou provoquer d'autres ennuis. Une commande NEW est alors nécessaire pour réinitialiser correctement l'interpréteur.

7.4.8 Pointeurs APPLESOFT préservés par le code compilé.

Il y a deux pointeurs utilisés par l'interpréteur qui sont préservés pendant l'exécution d'un programme compilé. Ces deux pointeurs sont MEMSIZ et TXTTAB. MEMSIZ est le pointeur du sommet de la mémoire affecté par l'instruction HIMEM, et TXTTAB est le pointeur de début de programme. Ils résident respectivement en $73-74 (115-116 en décimal) et $67-68 (103-104 en décimal).

Les programmes compilés utilisent MEMSIZ comme l'interpréteur et l'instruction HIMEM change le contenu de MEMSIZ.

Les programmes compilés n'utilisent pas TXTTAB, mais l'interpréteur l'utilise pour pointer au début du programme en mémoire. L'interpréteur et le DOS l'utilisent aussi pour décider où le programme LOADé commencera. Les programmes compilés préservent TXTTAB, ainsi un programme compilé pourra aisément RUNner un programme interprété utilisant le DOS.

Le format de stockage d'un programme utilisé par l'interpréteur de l'APPLESOFT requiert que l'emplacement juste au-dessous de la zone de programme pointée par TXTTAB contienne zéro. Un programme peut encore être entré quand cet emplacement n'est pas à zéro, mais la tentative de lancer le programme produira un '?SYNTAX ERROR'. Cette restriction tient aussi en utilisant le DOS pour charger ou lancer un programme interprété. TXTTAB pointe à son emplacement par défaut $800. $801 est mis à zéro quand l'APPLESOFT est initialisé, laissant ainsi TXTTAB à sa valeur normale.

Pour maintenir cet avantage, le compilateur laisse l'emplacement $800 protégé. La préservation de cet emplacement permet à un programme compilé de lancer un programme interprété sans avoir à

stocker un zéro en $800. $801-$802 sont protégés pour des raisons similaires, aussi l'adresse par défaut pour la librairie est $803 (2051 en décimal).

7.4.9 Chaînage entre programmes compilés.

Les programmes interprétés sont chaînés en utilisant la commande du DOS RUN. Les programmes compilés sont chaînés en utilisant la commande BRUN. Pour faciliter le chaînage, TASC permet des variables communes (COMMON). Un programme exécuté en séquences peut utiliser les variables COMMON pour passer les informations.

7.4.10 Opérations sur les chaînes de caractères.

TASC gère les chaînes différemment. L'interpréteur duplique habituellement les valeurs de chaînes dans une instruction telle que : A$ = B$. Si trente chaines doivent avoir la même valeur, l'interpréteur stocke trente fois la même chaîne en mémoire, ce qui ralentit l'exécution du programme.

TASC élimine les copies de chaînes en permettant plusieurs chaînes en un point de même valeur dans la mémoire. Ceci élimine la nécessité de dupliquer les chaînes et rend les opérations plus rapides. En échange, pour être très rapide en assignement, TASC est légèrement plus lent pour construire et prendre les chaînes séparément. LEFT$, RIGHT$ ET MID$ sont moins efficaces. Puisque ces fonctions sont moins utilisées que les assignements, la méthode utilisée par TASC reste plus efficace dans son ensemble.

Les chaînes inutilisées doivent être éliminées. La fréquence du 'nettoyage' est déterminée par deux facteurs : l'espace disponible et le taux de chaînes inutiles. Le nettoyage est généralement une opération assez longue dont la durée dépend de la quantité de chaînes utilisées dans un programme. Chaque fois que le nettoyage devient nécessaire, l'exécution du programme est suspendue pendant cette opération.

Le nombre de fois que le nettoyage est nécessaire peut être réduit en diminuant le taux de chaînes inutiles ou en augmentant l'espace disponible pour ces chaînes. Davantage d'espace peut être obtenu en réduisant la taille des tableaux, en raccourcissant le programme ou en plaçant MAXFILES à un nombre plus petit. Le taux de chaînes inutiles peut être diminué en réduisant l'emploi de LEFT$, RIGHT$, MID$ et les concaténations. Puisque TASC ne duplique pas les chaînes pour les assignements, leur nombre n'a pas d'influence. Le temps requis au nettoyage est grossièrement proportionnel au carré du nombre de chaînes, donc en diminuant ce nombre de moitié, le nettoyage se fera quatre fois plus rapidement.

8. AMELIORATIONS DU LANGAGE.

TASC fournit deux améliorations majeures à l'APPLESOFT qui offrent une subtantielle augmentation de la vitesse et une puissance de programmation :

1. Arithmétique entière réelle, boucles FOR/NEXT entières et
2. Variables communes (COMMON).

Pour ce faire, TASC inclue cinq nouvelles instructions :

CLEAR CHAIN	CLEAR COMMON	DEFCOMMON
INTEGER	USECOMMON	

Ces nouvelles instructions et les améliorations de langage qu'elles permettent vont être étudiées dans les sections suivantes.

8.1 ARITHMETIQUE ENTIERE.

Ceci permet une vitesse d'exécution plus rapide qu'avec l'interpréteur.

8.1.1 Arithmétique entière dans les programmes en interpréteur.

L'APPLESOFT inclue l'utilisation de variables entières, mais il les convertit en nombres à virgule flottante avant toute opération, ce qui ralentit la vitesse d'exécution. Le seul avantage d'employer ce type de variables réside dans le fait que lés variables entières employées dans les tableaux occupent deux octets par élément au lieu de cinq en virgule flottante. De plus l'interpréteur ne permet pas de variables entières comme index pour les boucles FOR/NEXT.

8.1.2 Arithmétique entière dans les programmes compilés.

Les programmes compilés avec TASC peuvent augmenter leur vitesse d'exécution si des variables réelles sont changées en variables entières. Toutefois, la conversion d'un programme en ajoutant le signe % pour chaque variable prend du temps. Pour éliminer ce problème, TASC inclue une instruction de déclaration INTEGER. INTEGER permet de définir des variables entières sans ajouter le signe %. Par exemple, la variable réelle I peut être déclarée comme entière en incluant simplement l'instruction INTEGER I.

8.1.3 L'instruction INTEGER.

Puisque l'interpréteur ne reconnait pas l'instruction INTEGER, le fait de l'inclure dans une instruction normale produirait une erreur de syntaxe si le programme était lancé sous APPLESOFT. Pour éviter ce problème, l'instruction INTEGER doit être inclue dans une instruction spéciale REM 'active'. Pour permettre au compilateur de reconnaître ces REM spéciales actives, elles seront distinguées des REM normales en utilisant un point d'exclamation après le mot REM. Par exemple, l'instruction suivante déclare I comme variable entière :

```
10 REM!INTEGER I
```

Les autres instructions additionnelles décrites plus tard seront également distinguées dans des REM actives. Ces nouvelles instructions seront ignorées lors de la mise en route sous interpréteur, mais seront reconnues par le compilateur et traitées comme des instructions normales.

L'instruction INTEGER peut déclarer des tableaux ou des variables simples. Les tableaux sont déclarés en incluant le nom et les dimensions dans la même forme qu'ils le seraient dans une instruction DIM. Les dimensions spécifiées doivent être identiques aux dimensions des autres instructions INTEGER, DEFCOMMON, USECOMMON ou DIM.

Les variables multiples peuvent être déclarées comme entières en séparant les noms de variables par une virgule. Les espaces sont permis entre les noms de variables et la virgule mais pas dans les noms de variables ou dans les dimensions de tableau.

Puisque plusieurs programmes peuvent déclarer toutes leurs variables comme entières, TASC inclue une option pour INTEGER. Si le premier caractère différent d'un espacement suivant INTEGER est un astérisque (*), toutes les variables numériques seront traitées comme entières.

Bien que de multiples déclarations de variables entières sont permises, seule la première aura un effet. Les instructions suivantes sont toutes acceptables :

```
10 REM! INTEGER I,J,K,L
20 REM! INTEGER I,J , AB(3,9),R(3)
30 REM! INTEGER*
```

Les déclarations suivantes incluent des espaces dans les noms de variables et sont donc inacceptables :

```
10 REM! INTEGER A% ,A 2,A 3
20 REM! INTEGER BA( 3, 7), D %(3,7)
```

Les instructions INTEGER peuvent se suivre ou être mélangées avec des REM inactives, mais doivent précéder toutes autres instructions du programme. Le compilateur recherche toutes les REM du programme et ignore les messages qui ne commencent pas par un point d'exclamation. Les REM ne contenant qu'un des mots-clés INTEGER, COMMON, CLEAR COMMON ou CLEAR CHAIN sont aussi ignorées. Pendant la compilation, le compilateur indique à l'utilisateur qu'il exécute une REM active en affichant 'RECOGNIZED'. Ce message est affiché même si l'option de non-affichage a été sélectionnée. Ces messages doivent être bien surveillés, car une REM incorrectement active est difficile à détecter en dehors du message 'RECOGNIZED'.

8.1.4 Boucles FOR/NEXT entières.

L'interpréteur ne permet pas l'emploi de variables notées du signe (%) dans les boucles. L'instruction INTEGER permise par TASC permet aux boucles FOR/NEXT de maintenir la compatibilité avec l'intepréteur.

TASC utilise des variables entières déclarées avec l'instruction INTEGER. Par exemple :

```
10 REM! INTEGER I
20 FOR I = 1 TO 10 : PRINT I : NEXT I
```

L'instruction INTEGER résoud le problème de la compatibilité avec l'interpréteur, puisque la variable de contrôle est traitée comme une variable réelle par l'interpréteur. TASC reconnaît la variable de boucle comme une variable entière et produit un code spécial pour cette boucle. La valeur de la variable doit être comprise entre -32767 et +32767.

8.1.5 Opérations entières.

Les valeurs entières dans une expression ne sont pas toujours calculées dans le mode entier. Si l'opération contient également des variables réelles, ou si l'entier est utilisé comme argument d'une fonction qui suppose une valeur réelle, l'entier serait converti en mode réel.

L'utilisation de quelques opérations sur entiers est controlée par le choix de l'option arithmétique entière. Les addition, soustraction, multiplication et division peuvent générer des dépassements dans l'accumulateur d'entiers, ainsi leur emploi peut etre explicitement controlé. Si l'option est choisie, les opérations sur entiers sont exécutées si les deux opérandes sont entiers.

Si l'option n'est pas choisie, les opérations sur entiers ne seront exécutées que pour des comparaisons et tests logiques, étant donné que ces opérations ne peuvent générer de surcharge et leur utilisation est automatique.

Cette option ne devrait être éliminée qu'en dernier ressort. Si l'inclusion de l'arithmétique entière ne produit un dépassement que dans peu de cas, les expressions qui produisent ce dépassement peuvent être forcées en mode réel.

Les opérations suivantes supposent des valeurs entières :

CHR$	COLOR=	DRAW
FOR (entier)	HCOLOR	HLIN
HPLOT	HTAB	IN#
LEFT$	LET (entier)	MID$
ON GOSUB/TO	PDL	PLOT
POKE (2)	PR#	RIGHT$
ROT=	SCALE=	SCRN
SPEED=	SPC	souscrits
TAB	VLIN	VTAB
WAIT (2&3)	XDRAW	

Les nombres entre parenthèses indiquent quels paramètres sont traités comme des entiers dans les opérations mixtes.

Les divisions sont toujours exécutées en mode fractionnaire puisqu'elles fournissent des résultats fractionnaires. Les opérations suivantes peuvent être exécutées en entier si l'option est choisie :

addition	multiplication
négation	soustraction

Les opérations suivantes peuvent être faites soit en réel soit en entier sans forçage de la conversion :

AND	FRE	IF/THEN
NOT	OR	POS

Toutes les autres opérations supposent des valeurs réelles. Les variables devront être déclarées réelles ou entières suivant les opérations les plus employées dans le programme. Les opérations suivantes retournent des valeurs entières :

ASC	LEN	PDL
PEEK	POS	SCRN

8.2 CHAINAGE AVEC COMMON.

TASC fournit aussi la possibilité de passer des variables communes entre programmes chaînés. L'addition de 'COMMON' permet à plusieurs programmes de passer des valeurs communes sans avoir à les stocker et à les recharger. Un bloc mémoire est réservé et protégé des programmes BRUNnés successivement et chaque programme se réfère au même bloc de valeurs qu'ils exécutent. Les variables déclarées communes (COMMON) dans un programme sont allouées dans le bloc dans l'ordre spécifique correspond à leur emploi dans les différents programmes.

Les noms actuels utilisés pour référencer ces variables n'ont aucune importance. L'emplacement mémoire commun est responsable de la correspondance. Par exemple, supposons que le PROGRAMME 1 déclare la variable A1 comme sa première variable commune et que le PROGRAMME 2 déclare la variable A2 comme sa première variable commune, l'emplacement de ces deux variables dans le bloc commun est le même, la valeur laissée dans la variable A1 à la fin du PROGRAMME 1 est la valeur initiale de la variable A2 au commencement du PROGRAMME 2.

8.2.1 USECOMMON et DEFCOMMON.

Ces instructions sont utilisées pour déclarer des variables communes, elles peuvent être inclues dans des REM actives :

```
10 REM! DEFCOMMON I,J,K
```

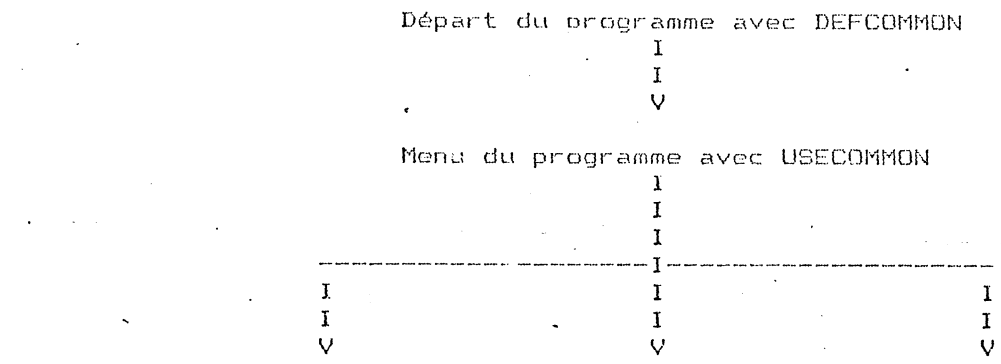
Des instructions multiples COMMON sont permises. Toute déclaration INTEGER doit précéder les instructions COMMON et les deux doivent précéder les autres instructions, sauf les REM inactives. Un ordre incorrect produirait une erreur DECLARATION durant la compilation. COMMON peut spécifier tout type de variables simples ou tableaux.

Les variables peuvent être déclarées en INTEGER et COMMON. La déclaration d'une variable commune par l'instruction COMMON plusieurs fois dans le programme causera l'erreur DECLARATION durant la compilation. Toute instruction DIM doit spécifier les mêmes dimensions pour le tableau.

DEFCOMMON place un bloc COMMON et initialise les variables s'y trouvant, tandis que USECOMMON accepte un bloc COMMON déjà mis par un autre programme sans effacer le bloc. DEFCOMMON présente un problème quand plusieurs programmes sont raccordés à un menu principal. L'utilisation de DEFCOMMON dans le menu produirait l'effacement de l'information venant du sous-programme, mais USECOMMON n'initialiserait pas du tout le bloc.

La solution est d'ajouter un autre programme contenant DEFCOMMON qui chaînerait au menu principal, et utiliserait alors USECOMMON

dans le menu :



Le programme démarre en plaçant initialement le bloc COMMON, puis il va au menu principal et ne retourne plus au début. Cette disposition prévient tout effacement par DEFCOMMON chaque fois que le menu est rappelé.

8.2.2 CLEAR CHAIN et CLEAR COMMON.

La version compilée de l'instruction CLEAR en APPLESOFT n'efface pas les variables communes. TASC le permet par son instruction CLEAR COMMON qui n'affecte pas les variables locales. CLEAR COMMON doit être utilisée dans une REM active.

L'instruction CLEAR CHAIN doit être inclue immédiatement avant l'instruction BRUN. CLEAR CHAIN permet de sauvegarder le stockage des chaînes de caractères pendant le chaînage. Si cette instruction est omise, les valeurs des chaînes peuvent être perdues ou modifiées. CLEAR CHAIN force le nettoyage de la mémoire et réinitialise les variables locales, cette instruction doit être contenue dans une REM active sur la ligne avant la commande BRUN.

8.2.3 Comment travaillent les variables communes.

L'espace du bloc commun est alloué dans l'ordre de déclaration des variables communes. Les entiers occupent 2 octets chacun et le nombres réels en occupent cinq. Les variables chaînes de caractères occupent deux octets chacunes, mais puisqu'elles sont allouées en portions séparées du bloc, il n'est pas possible de

les mélanger avec les variables numériques. Le format de ces variables est différent en compilé. L'octet de longueur pour une chaîne est stocké au début de la valeur de la chaîne au lieu d'avoir un pointeur de chaîne.

Les tableaux sont alloués avec le souscrit le plus à droite variant le plus vite. Par exemple : A(1,1) A(1,2) A(2,1) A(2,2). Chaque élément occupe deux à cinq octets suivant son type.

L'espace dans le bloc commun est divisé en deux sous-blocs, pour variables numériques et chaînes. Dans chaque sous-bloc, l'ordre de l'allocation de l'espace des variables est déterminé par l'ordre dans lequel les variables sont déclarées par les instructions COMMON. Quand le code compilé est exécuté, il s'assure que la taille du bloc numérique et chaîne est identique à la taille des blocs passés par le programme précédent. Le compilateur recherche si, par exemple, dix octets déclarés comme deux fois cinq octets de variables numériques réelles dans un programme ne sont pas devenues cinq fois deux octets de variables entières dans un autre programme. Ce type d'erreur peut être évité facilement en plaçant les déclarations COMMON identiques dans les deux programmes. Le mélange de taille du bloc COMMON produit l'erreur '?TYPE MISMATCH' à l'exécution.

COMMON peut aussi être utilisé pour passer des paramètres en langage machine. Le début du bloc et les positions des variables dans le bloc sont fixés, ainsi les programmes en langage machine peuvent être référencés à des emplacements statiques pour les variables. Le programme compilé peut aussi POKE les valeurs des variables dans un emplacement prédéterminé avant d'appeler le programme en langage machine. Certaines routines désignées pour les programmes interprétés placent leurs variables dans une liste des variables. Ces routines ne peuvent travailler avec un programme compilé, puisque la liste des variables est éliminée. Ces programmes doivent donc être modifiés par l'usage de COMMON ou par des valeurs de POKE explicites.

8.2.4 Notes sur COMMON.

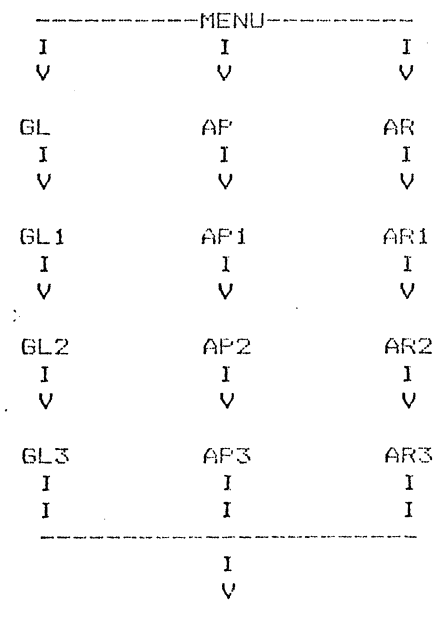
L'espace pour le bloc COMMON est alloué au commencement de l'espace déclaré pour le stockage du programme. Les programmes formant un bloc COMMON doivent tous être compilés avec la même adresse de départ pour l'espace programme. En recherchant les instructions COMMON, le compilateur incrémente l'adresse mémoire de départ du programme compilé pour laisser la place au-dessous pour les variables communes. Le fait d'inclure un GOTO ou une REM de déclaration de bloc COMMON fait que le programme compilé saute jusqu'au bloc COMMON, produisant des résultats indéfinissables.

Puisque COMMON n'est pas inclu dans le langage APPLESOFT, l'utilisation de l'interpréteur pour dépanner les programmes utilisant COMMON est difficile. Toutefois, le DOS fournit un programme en langage machine qui peut simuler certaines possibilités du chaînage compilé. Voir le fonctionnement du DOS avec la commande CHAIN.

Une méthode utilisant les files disquette permet de simuler le bloc COMMON. En imprimant (PRINT) et en entrant (INPUT) les variables communes dans le même ordre qu'elles le seront dans le bloc commun, vous pourrez détecter les erreurs d'ordre.

8.2.5 Création d'un système de programmes.

Les instructions DEFCOMMON et USECOMMON servent à créer de grands systèmes APPLESOFT qui communiquent chacun entre eux. Un exemple simple est décrit ici pour démontrer les interactions possibles. La distinction entre BRUN avec COMMON et un simple BRUN est aussi démontré.



Ce système est controlé par un menu principal.

Les programmes communiquant tous entre eux, la taille du bloc commun doit être identique dans chacun de ces programmes. Deux solutions sont possibles :

1. Utiliser les mêmes déclarations COMMON dans tous les programmes ainsi que toutes les informations communes s'y rattachant, ou

2. Utiliser les mêmes déclarations COMMON dans chacun des trois ensembles, sans information commune avec les autres ensembles ou le menu principal. Dans ce cas, il y a trois séries de déclarations COMMON, une pour chaque ensemble.

Les fragments de programme suivants démontrent comment les programmes peuvent être chaînés. Ils montrent aussi l'emploi d'un BRUN avec COMMON et d'un BRUN simple :

```
   MENU

    -
    -
1000 INPUT "QUEL ENSEMBLE ? ";N
1010 IF N = 1 THEN PRINT D$"BRUN GL"
1020 IF N = 2 THEN PRINT D$"BRUN AP"
1030 IF N = 3 THEN PRINT D$"BRUN AR"

   GL

10  REM! DEFCOMMON A, B(3,4), C$
    -
    -
1000 REM! CLEAR CHAIN
1010 PRINT D$"BRUN GL1"

   GL1

10  REM! USECOMMON A1, B1(3,4),C1$
    -
    -
1000 REM! CLEAR CHAIN
1010 PRINT D$"BRUN GL2"

   GL2

10   REM! USECOMMON A2, B2(3,4),C2$
    -
    -
1000 REM! CLEAR CHAIN
1010 PRINT D$"BRUN GL3"

   GL3

10   REM! USECOMMON A3, B3(3,4),C3$
    -
    -
1000 PRINT D$"BRUN MENU"
```

Ces exemples montrent l'ensemble GL. Les deux autres ensembles

sont similaires. Remarquer que le MENU n'a pas de déclaration commune, puisqu'il ne passe ni ne reçoit d'information. GL a une déclaration DEFCOMMON car il doit passer l'information à GL1, mais il n'en reçoit pas du menu. La déclaration DEFCOMMON place le bloc commun et l'initialise.

GL utilise un CLEAR CHAIN avant le BRUN, puisque GL doit passer l'information à GL1 qui déclare USECOMMON puisqu'il doit recevoir et ne pas initialiser le bloc commun passé par GL.

GL1 inclue un CLEAR CHAIN pour passer l'information à GL2 qui est similaire à GL1, et passe l'information à GL3. GL3 utilise également USECOMMON pour accepter le bloc passé par GL2 mais ne contient pas de CLEAR CHAIN avant BRUN puisque qu'aucune information n'est à passer au MENU.

9. COMMENT LE COMPILATEUR TRAVAILLE.

La première fonction d'un compilateur est de translater un programme source en langage machine. Ce processus est divisé en deux parties :

1. Analyse de la syntaxe - La reconnaissance des instructions APPLESOFT que le compilateur prend comme entrée.

2. Génération du code - La production du langage machine équivalent aux instructions APPLESOFT.

L'étude des ces deux parties majeures est précédée par une description du programme qui les exécute.

9.1 PASS0, PASS1, ET PASS2.

TASC est un compilateur à deux 'passes', puisqu'il compile en deux parties majeures. La PASS0 prend seulement les entrées utilisateur et place les paramètres de compilation et n'est donc pas le processus actuel de compilation.

Les PASS0 et PASS1 chaînent aux PASS1 et PASS2, respectivement. Les trois passes sont écrites en APPLESOFT et TASC les utilise pour se compiler.

La première passe est la PASS1 qui exécute les analyses de syntaxe et génère le plus de code. Comme elle examine aussi le programme, la PASS1 collecte l'information sur les variables et les numéros de ligne et stocke l'information dans la table des symboles.

La table des symboles est utilisée pour stocker toute information sur le programme. La compilation de toute cette information et son insertion élimine le temps nécessaire à la recherche durant l'interprétation.

Puisque PASS1 ne peut allouer le stockage des variables ou connaître les adresses de tous les numéros de ligne jusqu'à ce que tout le programme est été passé en revue, il ne peut insérer l'adresse actuelle dans le code qu'il génère. A la place, il doit laisser une place où les adresses pourront être placées plus tard. PASS2 exécute le placement après que PASS1 ait terminé le processus sur la file source.

PASS2 utilise l'information fournie par PASS1 pour allouer le stockage des variables. PASS1 tient un enregistrement de l'usage des variables dans la table des symboles, ainsi PASS2 utilise la table de symboles pour allouer le stockage. PASS2 utilise les types de variables stockées pour décider combien d'espace allouer, puis sauve l'adresse du stockage alloué à chaque

variable avec les autres informations sur ces variables.

Les numéros de ligne sont tenus légèrement différemment. La tenue de tous les numéros de ligne et adresses en mémoire requiert trop de mémoire, donc PASS1 stocke les adresses des numéros de ligne dans la file disquette 'CLINENUM" en générant le code. PASS2 utilise cette file pour remplacer les numéros de lignes référencées à leur adresse actuelle. Cette adresse déterminée, PASS2 doit chercher toutes les références laissées indéfinies dans PASS1. Plutôt que de conserver une liste énorme de tous les emplacements dans le programme où chaque variable est utilisée, PASS1 rassemble toutes les références pour chaque variable ou numéro de ligne dans une chaîne. Chaque référence contient l'adresse de la référence précédente au lieu d'une adresse inconnue.

Puisque la file est convertie avant, ces chaînes sont mises à leur position courante après le commencement de la file. PASS1 garde la référence la plus récente dans la table des symboles qui pointent sur la suivante plus récente et ainsi de suite.

Ce chaînage laisse PASS2 avec la fin de la chaîne pour chaque place de référence et PASS2 utilise ce chaînage pour remonter et placer les références indéfinies. Ce processus est le seul but de PASS2. Exécuter ce système sur disquette est long, mais permet de s'affranchir des limitations de longueur de programme que réclamerait un système travaillant en mémoire.

9.2 ANALYSE DE LA SYNTAXE.

C'est le premier pas dans le processus de compilation, puisque le compilateur doit être capable de comprendre le programme source avant de pouvoir générer un code. L'anayseur de syntaxe doit examiner la ligne courante et décider quelle instruction elle represente avant que le générateur de code puisse dire quel code produire.

Le processus de reconnaissance d'instructions peut être séparé en deux parties : analyse lexicale et analyse grammaticale. Le groupement de lettre dans des mots est une analyse lexicale; le groupement de mots dans une phrase est une analyse grammaticale.

9.2.1 Analyse lexicale.

Elle vérifie les caractères dans une ligne entrée et sépare les mots qui font partie des instructions APPLESOFT. Ces mots, comme PRINT, FOR, etc, sont appelés mots-clés. Une analyse lexicale prend 'FOR1=ABTOC' et produit 'FOR 1 = AB TO C'. Cette analyse substitue un code numérique pour chaque mot-clé trouvé, permettant de les retrouver plus tard. La reconnaissance de 'FOR', '=', et 'TO' dans les exemples précédents produiront

'<129> I <208> AB <193> C'.

L'interpréteur analyse lexicalement chaque ligne de programme comme elle est tapée, ainsi les programmes APPLESOFT sont stockés dans un format reconnu. Quand un programme est listé, l'interpréteur substitue le mot approprié pour chaque code de reconnaissance, et le programme devient lisible. Les programmes stockés sur disquette sont aussi dans un format code de reconnaissance.

9.2.2 Analyse grammaticale.

Puisque l'interpréteur reconnaît ses programmes, TASC n'a pas à exécuter d'analyse lexicale. Toutefois, TASC doit encore analyser grammaticalement l'entrée. Cette analyse est normalement accomplie en utilisant une de ces deux approches : du haut vers le bas ou du bas vers le haut. Les deux méthodes produisent le même résultat, elles prennent une entrée qui a été analysée lexicalement et identifient les groupes logiques d'information. La méthode haut vers bas suppose d'abord que l'entrée est une certaine instruction, puis tente de trouver les parties constituantes. La méthode bas vers haut examine d'abord les parties, puis en déduit quel type d'instruction elles représentent. La plupart des instructions APPLESOFT peuvent être identifiées en regardant le premier caractère, aussi TASC analyse les instructions de haut en bas.

Les expressions doivent aussi être analysées grammaticalement. Même si elles ne sont pas composées de lettres et de mots, elles incluent encore des symboles que le compilateur doit pouvoir organiser et comprendre. Cette analyse doit examiner l'expression et décider comment l'évaluer. Elle utilise les règles de précédence et toutes parenthèses présentes pour décider quelles opérations doivent être exécutées en priorité.

Le code générateur ne considère pas la précédence, aussi l'anayse grammaticale doit réordonner les opérations qu'elle pourra sortir séquentiellement durant la génération du code. Les expressions seront analysées du bas vers le haut, et réordonnées à l'aide d'une pile.

Les expressions réordonnées sont passées au générateur de code dans la forme 'triple'. Un triple est une unité consistant en un opérateur et un ou deux opérandes. Les opérations ou fonctions qui n'ont seulement qu'un seul operande laisse l'espace pour l'autre inutilisé. L'opérande d'un triple est souvent le résultat d'un triple précédent. L'exemple de triples suivants représente l'expression :

A + B * C

TRIPLE #	OPERATEUR	OPERANDE GAUCHE	OPERANDE DROIT
1	charge	B	
2	charge	C	
3	multiplie	triple 1	triple 2
4	charge	A	
5	additionne	triple 4	triple 3

Les triples 1 et 2 indiquent quelles variables B et C doivent être chargées. Le triple 3 instruit le générateur de code de générer une multiplication de B par C. Le triple 4 indique quelle valeur doit être chargée pour A. Le triple 5 spécifie que A doit être additionné au résultat produit par la multiplication dans le triple 3.

9.3 GENERATION DU CODE.

la génération de code travaille de deux manières différentes. Le code pour expressions est généré par un appel explicite au générateur de code à l'aide des triples. Le code pour instructions est généré durant l'analyse syntaxique. Par exemple, l'appel à la routine CLEAR est généré aussitôt que l'analyse syntaxique reconnaît l'instruction CLEAR. La plupart des instructions comme END, GR, TEXT,etc. sont représentées par un appel simple à une librairie ou à la routine APPLESOFT.

Les instructions qui impliquent des expressions font aussi normalement l'objet d'appel simple. L'analyseur de syntaxe reconnaît l'instruction, et l'analyse grammaticale,l'expression. Un appel au générateur de code produit l'instruction pour évaluer l'expression, et l'analyseur de syntaxe termine le code pour l'instruction avec appel langage machine à la routine qui utilise cette valeur.

L'interpréteur doit chercher la liste des variables et les numéros de ligne dans l'ordre pour trouver une variable ou un numéro de ligne. Le compilateur, lui, génère une adresse absolue pour les variables et les numéros de ligne. Au lieu de rechercher quelque chose à l'exécution, le compilateur collecte toutes les informations sur le programme à la compilation. Au lieu de chercher le numéro de ligne lorsqu'il rencontre un GOTO 80, le code compilé saute simplement à l'adresse déjà fournie par le compilateur.

Le compilateur ne cherchant pas ses informations pendant l'exécution du programme, doit donc les rechercher à la compilation.

9.4 TECHNIQUES SPECIALES.

Bien que la vitesse d'exécution soit importante, si la longueur du code compilé est supérieure au programme source, pour un programme déjà long, ceci peut devenir une cause d'ennuis. Dans cette optique, TASC utilise plusieurs techniques qui génèrent un code compilé.

9.4.1 Accessibilité des variables.

L'accessibilité des variables est une source majeure d'expansion du code. Les codes compilés et interprétés doivent tous deux charger et stocker des valeurs dans les variables. L'accessibilité des variables est requise presque constamment, aussi le compilateur utilise-t-il des routines de chargement et de stockage, ce qui permet de sauver un peu d'espace mémoire. La routine de transfert peut être inclue dans le code objet seulement une fois au lieu de l'employer à chaque fois que c'est nécessaire.

L'information nécessaire pour cette routine est l'adresse mémoire de la variable. Cette adresse doit être passée à chaque fois que la routine est appelée. L'adresse est normalement passée par deux registres du microprocesseur. Chaque fois qu'une variable est accédée, le même processus se répète : chargement des registres, appel de la routine, chargement des registres, appel de la routine...

L'utilisation des registres pour passer les adresses diminue de plus de la moitié la quantité de code requis pour accéder à chaque variable. On peut encore améliorer le processus en ayant une routine spéciale pour chaque variable. Cette routine prend la même place qu'une routine générale, mais chaque accès à cette variable peut simplement appeler cette routine spécialisée. Il n'est donc plus nécessaire de passer l'adresse à cette routine.

Cette technique prend plus d'espace si la variable n'est utilisée qu'une fois, mais sauve de l'espace si elle est utilisée plusieurs fois. Par exemple, une variable référencée vingt fois utilise 67 octets au lieu de 140 octets normalement.

Malheureusement, cette technique ne travaille bien que si les variables ne sont référencées qu'une fois. Puisqu'une variable peut être chargée, stockée, ajoutée, etc., il est réellement nécessaire d'avoir une routine spécialisée par variable et pour chaque opération. Plusieurs routines seraient donc requises par variable.

Pour pallier à ce problème, TASC utilise une méthode plus sophistiquée. Plutôt que de générer plusieurs routines par variable, TASC génère une seule routine. Cette routine a des points d'entrées différents pour chaque type d'opération sur les

variables. Ces points d'entrées chargent chacun un nombre différent pour indiquer l'opération nécessaire, puis ils utilisent tous le même code pour charger l'adresse de la variable. Toutefois, à cause de la sous-routine qui est maintenant utilisée pour exécuter différentes opérations, elle saute à une routine spéciale qui utilise le numéro de l'opération pour distribuer et exécuter l'opération correcte.

9.4.2 La librairie RUNTIME.

TASC utilise d'autres techniques pour réduire la taille du code objet. Plusieurs techniques utilisées déplacent des parties du code objet et les placent dans la librairie. Ceci augmente la taille de la librairie, mais réduit subtantiellement celle du code.

La librairie RUNTIME inclue dans TASC réduit également l'espace nécessaire sur disquette, puisqu'il suffit de ne la charger qu'une fois. Tous les programmes utilisent donc une librairie commune. Un désavantage subsiste, toute la librairie est utilisée pour chaque programme. Par exemple, pour un programme n'utilisant pas les chaînes de caractères, la routine, elle, les contient. Toutefois, les grands programmes utilisent la plus grande quantité de routines disponibles, alors que les petits ne les utilisent pas tous, mais ont plus de mémoire disponible.

10. MESSAGES D'ERREUR ET DEPANNAGE.

Ce chapître explique les messages d'erreur qui peuvent se produire à la compilation ou à l'exécution.

10.1 MESSAGES D'ERREUR A LA COMPILATION.

TASC utilise deux types de messages d'erreur : précautions et erreurs fatales. Les erreurs fatales indiquent les problèmes qui empêcheront la réussite de la compilation. Les précautions indiquent simplement les instructions qui seront ignorées par le compilateur.

Les précautions qui indiquent un code inexécutable sont habituellement causées par des instructions suivant un GOTO ou un RETURN sur la même ligne de programme. Les précautions n'empêchent pas qu'une file objet soit créée, mais aucun code ne sera généré pour l'instruction signalée. RESUME est ignoré si l'option n'est pas choisie.

Les erreurs fatales produisent un message et une prise d'erreur. Le pointeur d'erreur, !ERR!, apparait dans l'instruction incorrecte au point où l'erreur est reconnue. Les erreurs fatales font que la file objet incomplète est annulée. La compilation ne continue que si toutes les autres erreurs peuvent etre détectées, la génération du code ne continue pas. Voici la liste des erreurs fatales et leurs causes :

DECLARATION	INTEGER ou COMMON : déclarations hors de la séquence ou non en début du programme. USECOMMON et DEFCOMMON déclarés dans un simple programme. Variable déclarée COMMON plus d'une fois.
INCOMPLETE	Expression incomplète. Oubli de parenthèse droite dans une expression.
OPERAND	Opérande illégal dans une expression. Constante arithmétique trop grande.
REDEFINED	Fonction définie plus d'une fois. Dimensions d'un tableau spécifique différentes des dimensions spécifiées la première fois.
SUBSCRIPT	Oubli du premier souscrit. Dimension n'étant pas une constante entière. Dimension négative ou supérieure à 32767. Plus de 88 souscrits. Nombre de souscrits différent que dans le premier usage.

SYMBOLE TABLE FULL	Compilateur hors de l'espace de la table des symboles. Voir section 5.3 .
SYNTAX	Oubli ou ajout d'un caractère ou d'une donnée. Numéro de ligne supérieure à 65534.
TOO COMPLEX	Expression trop complexe. IF/THEN : trop de boucles.
TOO LONG	Entrée de ligne supérieure à 240 caractères. Code objet ou variables pour le programme compilé supérieurs à 48K.
TYPE MISMATCH	Présence d'une expression numérique là où une chaîne de caractères était supposée, ou vice versa. Expression chaîne de caractères dans IF/THEN.

Les numéros de lignes indéfinis ou fonctions produisent des erreurs fatales au commencement de PASS2. Le numéro de ligne donné dans le message d'erreur est la dernière référence de la fonction ou ligne. Toute autre référence doit être corrigée.

Le compilateur suppose une file source qui a été analysée et a eu les espaces déplacés par l'interpréteur, le compilateur ne permet pas des extra-espaces. Les espaces additionnels causeront des erreurs de syntaxe durant la compilation. Si une file contient des espaces additionnels, les déplacer en LISTant le programme dans une file texte, puis EXECuter la file texte en revenant en mémoire.

10;2 MESSAGES D'ERREUR PENDANT L'EXECUTION.

A part l'erreur '?TYPE MISMATCH', les messages d'erreur produits par un programme compilé sont identiques à ceux produits par l'interpréteur. L'erreur 'TYPE MISMATCH' ne devrait pas sortir à l'exécution, puisqu'il est recherché durant la compilation. Cette erreur peut être utilisée pour indiquer un erreur de bloc COMMON. Voir section 8.

Puisque le code compilé est un programme en langage machine, il ne peut afficher de numéro de ligne dans ses messages d'erreur. Par contre, il produit une adresse du code objet. Cette adresse peut être comparée avec la ligne correspondante du programme source APPLESOFT en utilisant le numéro de ligne à l'adresse de référence du code objet offert à la fin de PASS2. S'il y a des erreurs dans le programme, il devra être dépanné avec l'interpréteur.

Certaines erreurs ayant cours à l'exécution n'incluent pas une adresse du code objet. Ces erreurs sont affichées avec l'adresse

0. L'utilisation de l'option ONERR/Dépannage rend possible pour le compilateur de toujours inclure une adresse du code objet, mais l'inclusion de cette option rend la file objet plus lente et plus longue. Pour cette raison, le développement du programme et son dépannage doit être faits le plus possible avec l'interpréteur.

Les erreurs se produisant pendant l'exécution d'un programme compilé stoppent le programme sans réinitialiser correctement l'interpréteur. NEW après une erreur permet de s'assurer que les pointeurs et les adresses utilisés par l'interpréteur sont replacés correctement.

10.3 SOURCES DE PROBLEMES COMMUNS.

Cette section explique les problèmes qui peuvent survenir avec un programme compilé.

10.3.1 APPLESOFT.

TASC ne peut fonctionner qu'avec l'interpréteur APPLESOFT, qu'il soit en ROM, ou en carte langage ou en carte RAM, mais pas avec celui qui se chargeait comme un programme INTEGER.

10.3.2 Graphiques Haute-Résolution.

TASC supporte pleinement tous les graphiques haute-résolution sans modification. APPLESOFT utilise deux zones : HGR, qui réside de 8192 à 16383 ($2000-$3FFF) et HGR2, qui réside de 16384 à 24575 ($4000-$5FFF).

L'exécution d'une commande HGR ou HGR2 écrit des zéros dans la zone mémoire appropriée et détruirait donc toute partie de programme compilé, librairie ou variables qui s'étendrait dans cette zone. Le programme compilé ne pourrait donc pas être exécuté. Ce problème est habituellement aisé à reconnaître et à corriger, car il a lieu avec les programmes utilisant les graphiques haute-résolution et se produit aussitôt après une commande HGR ou HGR2.

Ces problèmes peuvent être évités en examinant les statistiques fournies à la fin de la compilation. Les adresses incluses indiquent où se trouvent le programme, la librairie et les variables. Si le programme utilise la haute-résolution, vérifier que les adresses n'entrent pas dans le domaine HGR utilisé. Si c'est le cas, le programme doit être recompilé en utilisant l'emplacement mémoire comme étudié au chapitre 5.

Les tables de forme utilisées par la haute-résolution sont souvent la cause de problèmes plus subtils. Elles doivent être chargées (BLOAD) ou POKE dans la mémoire comme un programme en langage machine, et créent souvent les mêmes problèmes de conflit mémoire. La table de forme peut résider au même emplacement que le code compilé.

En fait, les tables de forme pour programmes interprétés tendent à être stockées exactement là ou le programme compilé se trouve. Le programme interprété se situe normalement au-dessous des pages haute-résolution, aussi la table se situe-t-elle juste au-dessus de la page haute-résolution. L'emplacement de cette table doit être déterminée en regardant l'adresse spécifiée dans les POKE qui la placent en mémoire. S'il y a conflit, il doit etre simple de réallouer l'adresse de la table. Sinon, le programme peut aussi être compilé à une autre adresse. Voir chapitre 5.

Le graphique basse-résolution normal ne doit pas poser de problèmes. La partie mémoire utilisée est la même que pour la page texte et cette zone est hors de portée des programmes interprétés et compilés.

10.3.3 Programmes en langage machine.

Les programmes en langage machine sont utilisés avec APPLESOFT pour des applications spécifiques. Puisque la compilation change totalement la représentation interne du programme, quelques programmes en langage machine risquent de ne plus pouvoir travailler avec le programme compilé.

Un tel programme ne fonctionnera pas car il dépend des caractéristiques du programme interprété, la routine ne langage machine devra donc être réécrite. Toutefois, certaines routines ne présentent que des conflits d'emplacement mémoire.

Certains programmes en langage machine sont chargés en même temps que le programme APPLESOFT et sont difficiles à détecter comme par exemple le programme PENNY ARCADE. Ces programmes sont généralement appelés par des CALL à partir de l'APPLESOFT. Les adresses spécifiées dans les CALL ou les POKE donnent une bonne indication de l'emplacement où ce programme cause le problème. La plupart des routines sont écrites pour résider en Page 3, qui est inutilisée par l'interpréteur et le compilateur, mais dont certaines parties sont utilisées par le DOS. La Page 3 occupe les adresses 768-1023 ($300-$3FF).

Les routines localisées en Page trois ne causent pas de problèmes habituellement. Puisqu'elle n'est pas utilisée par le code compilé, les programmes en langage machine peuvent s'y trouver. Mais les CALL dans les ONNER GOTO utilisent une routine

spéciale APPLESOFT qui est utilisée pour éliminer les problèmes causés par ONNER GOTO. Voir chapître 7 pour plus d'informations.

10.3.4 Programmes s'auto-modifiant.

Certains programmes interprétés se modifient eux-mêmes tel que le programme 'PHONE LIST' de démonstration. Dans ce programme, les données sont sauvées à l'intérieur du programme lui-même et font donc partie intégrante du programme. D'autres programmes suppriment des lignes afin d'être exécutés plus rapidement ensuite. Ces programmes ne fonctionnent pas correctement quand ils sont compilés. La seule façon de les compiler est de les réécrire normalement.

APPENDIX A

DEPLACEMENT DE FILES BINAIRES AVEC L'UTILITAIRE ADR.

A.1 FILES BINAIRES.

Cette section décrit comment charger et sauver des files en langage machine.

Le DOS inclue trois types de files : APPLESOFT, TEXTE et BINAIRE. Le type de file est indiqué par une lettre : A,T ou B. Les files en langage machine sont stockées en BINAIRE.

A.2 SAUVETAGE ET CHARGEMENT DES FILES BINAIRES.

La commande BSAVE sauve une partie de la mémoire sur disquette dans un format binaire, c'est la façon dont sont traités les programmes compilés. Avec cette commande, l'adresse de départ et la longueur du programme doivent être fournies :

BSAVE <nom de file>, A <adresse>, L <longueur>

A est l'adresse de départ et L la longueur du programme. Sans le signe $ placé après ces deux lettres, la valeur sera donnée en décimal, avec le signe $ placé après, la valeur sera donnée en hexadécimal. Les paramètres : numéro de connecteur, numéro du drive, numéro de volume peuvent être précisés ou non.

Les commandes BLOAD et BRUN sont similaires à la commande BSAVE. Elles chargent normalement le contenu de la file disquette dans la même zone mémoire d'où elle avait été précédemment sauvée (par BSAVE). Par exemple, le chargement d'un programme se situant en 4000 sera rechargé en 4000. Toutefois, il est possible de le recharger à un autre emplacement en le précisant

BLOAD <nom de file>, A 6000

Ce qui provoquera le chargement en 6000 au lieu de 4000. Normalement, il n'y a pas lieu de modifier cette adresse de chargement, mise à part la librairie RUNTIME dont l'adresse par défaut est 2051 ($803).

La commande BRUN lancera le programme dès qu'il se trouvera en mémoire.

A.3 L'UTILITAIRE ADR.

Cet utilitaire permet de connaître l'adresse de départ et la longueur du programme compilé. Le DOS tient l'adresse de départ et la longueur de la file la plus récemment BLOAD ou BRUN dans un emplacement spécial. L'utilitaire ADR regarde simplement le contenu de cet emplacement et affiche ces paramètres. ADR est une file texte, elle s'exécute en tapant 'EXEC ADR'. ADR n'affecte aucun programme ou file en mémoire, elle exécute simplement une instruction PRINT pour fournir l'information nécessaire. ADR ne fonctionne qu'à partir de l'APPLESOFT.

ADR affiche les paramètres en décimal. Les nombres seront à utiliser avec les paramètres 'A' et 'L' pour BSAVE le programme sur disquette :

```
BLOAD <nom de file>
EXEC ADR
BSAVE <nom de file>, A<adresse>, L<longueur>
```

L'adresse (A) et la longueur (L) sont donc trouvées par EXEC ADR. La même procédure peut être employée pour déplacer d'autres files binaires sur le TASC - RUNTIME, PASS0, PASS1 et PASS2.

Le programme CREATE ADR est également fourni et permet de créer d'autres files texte ADR sur d'autres disquettes. La procédure pour transférer la file ADR sur une autre disquette est simple :

1. Charger le programme CREATE ADR à partir de la disquette TASC.

2. Enlever la disquette TASC.

3. Insérer la disquette devant recevoir la copie de ADR.

4. Taper RUN.

Le programme CREATE ADR ouvre une file texte appelée ADR sur la disquette, écrit la commande PRINT, ferme la file et stoppe.

APPENDIX C

CREATION D'UN SYSTEME AUTOMATIQUE.

Il est très simple de lancer automatiquement un programme dès que la disquette est bootée. Le programme APPLESOFT donné dans cette section BLOAD la librairie RUNTIME et BRUN la file désirée. En tapant ce programme comme un programme HELLO sur une disquette, le programme compilé sera exécuté dès le chargement de la disquette.

Dans le programme suivant, le programme à charger se nomme PROGRAMME.OBJ. Pour l'utiliser, changer simplement ce nom.

```
10 PRINT CHR$(4) + "BLOAD RUNTIME" + CHR$(13) + CHR$(4) +
     "BRUN PROGRAMME.OBJ"
```

CHR$(4) correspond au <CTRL-D>, CHR$(13) à RETURN. L'instruction PRINT exécute les deux commandes DISK - un BLOAD et un BRUN. Le RETURN est nécessaire pour séparer les deux commandes.

Le programme APPLESOFT est détruit par le chargement de RUNTIME, donc BLOAD et BRUN doivent se trouver dans la même instruction PRINT. C'est aussi pourquoi la concaténation (+) est utilisée pour réunir les chaînes.

APPENDIX D

NOTES SUR APPLESOFT.

Cet appendix donne des informations sur quelques instructions APPLESOFT qui ne sont pas données dans le manuel APPLESOFT.

1. TAB et SPC dans les instructions PRINT.
2. Réentrée de paramètres après une erreur dans INPUT.
3. Dessin sur écran avec DRAW et XDRAW.
4. Les instructions ONERR GOTO.
5. Les instructions FOR/NEXT.

D.1 TAB et SPC.

Utilisés en dernier item dans une instruction PRINT, TAB et SPC, agissent comme s'ils étaient suivis d'un point-virgule et suppriment l'impression du RETURN.

Par exemple, les lignes de programme suivantes sont équivalentes :

```
PRINT "MEME" TAB(10) : PRINT "LIGNE"
PRINT "MEME" TAB(10);: PRINT "LIGNE"
PRINT "MEME" TAB(10) "LIGNE"
```

Ces trois formes afficheront :

```
MEME          LIGNE
```

D.2 L'INSTRUCTION INPUT.

Lorsque le message '?REENTER' apparait, l'instruction INPUT complète est réexécutée.

D.3 DRAW et XDRAW.

Utilisés dans des formes haute-résolution, si les coordonnées X

et Y dépassent les valeurs requises, un message '?ILLEGAL QUANTITY ERROR' est affiché. Si la forme dépasse un bord d'écran, la partie coupée réapparait sur l'autre bord de l'écran.

D.4 ONERR GOTO.

ONERR GOTO a un petit problème qui fait que l'interpréteur ignore les instructions suivant un ONERR GOTO sur la même ligne. L'instruction PRINT "ET APRES" en ligne 10 ne sera jamais exécutée :

```
10 PRINT "AVANT" : ONERR GOTO 30: PRINT "ET APRES"
20 STOP
30 PRINT "ERREUR"
```

Le compilateur tient l'ONERR GOTO de la même manière que l'interpréteur.

D.5 FOR/NEXT.

Toutes les conditions utilisées pour les boucles sont maintenues par TASC.