Chapitre 10
Programmes et sous-programmes : documentation

10.1 Exemple de documentation du programme Facture

La figure 10.2 donne un exemple de sous-programme, tant du point de vue de la programmation structurée en assembleur que du point de vue de la documentation interne et externe. Le guide d'utilisation de Facture qui suit, permet de comprendre les objectifs du sous-programme.

Guide d'utilisation de **Facture**

BUT

Facture est la version simplifiée d'un programme de production calculant les frais maximum pour des services médicaux fournis à un patient pour une simple visite. On passe à Facture les différents traitements subis (de 1 à 5), ainsi que le nombre total de quarts d'heure de service. Facture retourne les frais maximaux en cents. Ces frais sont calculés en déterminant les frais pour chaque type de service et en conservant le maximum ainsi calculé. Facture permet également un code de retour indiquant soit un retour normal, soit un retour avec erreur si un type non permis a été passé ou si un nombre de quarts d'heure erroné a été transmis.

Calcul des frais :

Type	**Coût**
1	15$ jusqu'à 3 heures, 45$ au-dessus de 3 heures
2	3 heures = 36$, 4 heures = 48$, 5 heures = 60$, 6 heures et plus = 72$. La durée est arrondie à la prochaine heure.
3	3.75$ par quart d'heure.
4	12.00$
5	1/4 heure = 12.00$, 1/2 heure = 22.50$, 3/4 heure = 34.50$, 1 heure et plus = 45.00$
6	12.00$

Dans la figure 10.2 on notera l'utilisation de pseudocode structuré dans les commentaires, ainsi que des étiquettes numérotées pour les diverses structures telles que vues au chapitre 8. Ceci alourdit passablement l'exemple et dans la réalité on n'utilise qu'une seule de ces méthodes.

Le code de la figure 10.1 donne un exemple d'utilisation de Facture pour tests par exemple.

```
; Appel du sous-programme Facture
;
;     Philippe Gabrini   Octobre 1993 (revu en août 2005)
;
LF:     .EQUATE 0x0A            ;Line Feed
                                ;int main() {
AppFact: LDA    typ1,i          ;  Liste des cinq mots des types de service
         STA    liste1,d        ;    premier paramètre
         LDA    temps,i         ;  Durée
         STA    liste2,d        ;    deuxième paramètre
         LDA    result,i        ;  Résultat
         STA    liste3,d        ;    troisième paramètre
                                ;
         LDA    liste1,i        ;    liste de trois pointeurs
```

```
         STA    -4,s            ;    sauter espace pour code résultat
         SUBSP  4,i             ;    empilage
         CALL   Facture         ;  Facture(liste1);
                                ;
Affiche: DECO   0,s             ;  cout >> code;
         ADDSP  2,i             ;  Désempile code
         CHARO  LF,i            ;  cout >> endl;
         DECO   result,d        ;  cout >> résultat
         CHARO  LF,i            ;       >> endl
         STRO   mesg,d          ;       >> "Fin du traitement";
         STOP                   ;}
;
typ1:    .WORD   3
typ2:    .WORD   4
typ3:    .WORD   5
typ4:    .WORD   2
typ5:    .WORD   0
temps:   .WORD   10
result:  .WORD   0
liste1:  .BLOCK  2    ;Liste des trois pointeurs
liste2:  .BLOCK  2
liste3:  .BLOCK  2
mesg:    .ASCII  "Fin du traitement\n\x00"
;
```

Figure 10.1 Utilisation du programme Facture

```
; Calcul de la somme due par un  patient de la clinique
;
; Ce programme reçoit un tableau de cinq entiers indiquant les
; types de service fournis au patient (entiers de 1 à 5). S'il
; y en a moins que 5, le tableau est complété par des zéros.
; Si un type est invalide, on revient au programme appelant avec
; un code d'erreur négatif au sommet de la pile.  Le programme
; reçoit aussi la durée de traitement en quarts d'heure et
; produit comme résultat le coût dû par le patient.
;
;  Appel:  réserver espace pour code résultat
;          empiler adresse liste 3 pointeurs
;          CALL  Facture
;             avec Liste de 3 mots
;                Mot 1 ---> 5 mots: types de service
;                Mot 2 ---> 1 mot: quarts d'heure de service
;                Mot 3 ---> 1 mot: résultat du traitement
;
;      Philippe Gabrini    juin 1993 (revu en août 2005)
;************************************************************
type:    .EQUATE 0         ; type de traitement temporaire
compte:  .EQUATE 2         ; compteur temporaire
cout:    .EQUATE 4         ; cout temporaire
res:     .EQUATE 6         ; adresse résultat
duree:   .EQUATE 8         ; valeur durée
table:   .EQUATE 10        ; adresse table des types
regX:    .EQUATE 12        ; sauvegarde X
regA:    .EQUATE 14        ; sauvegarde A
retour:  .EQUATE 16        ; adresse de retour
liste:   .EQUATE 18        ; adresse de la liste
code:    .EQUATE 20        ; indice du résultat
;
Facture: SUBSP    16,i       ;void Facture(liste) {
         STA      regA,s     ;  sauvegarde A
         STX      regX,s     ;  sauvegarde X
         LDA      0,i        ;  code retour normal
         STA      code,s     ;
         LDX      0,i        ;  index = 0;
         LDA      liste,sxf  ;  adresse table des types de service
         STA      table,s    ;
         ADDX     2,i        ;  index = 1;
         LDA      liste,sxf  ;  adresse durée
         STA      duree,s    ;
```

```
         ADDX     2,i        ;  index = 2;
         LDA      liste,sxf  ;  adresse résultat
         STA      res,s      ;
         LDA      duree,sf   ;  valeur durée
         STA      duree,s    ;
         LDA      0,i        ;  coût nul
         STA      cout,s     ;
         STA      compte,s   ;
Boucle:  NOP0               ;  while(true) {  //5 fois
         LDX      compte,s   ;    index = 0;
         LDA      table,sxf  ;    service(index)
         CPA      0,i        ;
         BREQ     FinBouc    ;  if(type == 0) break;
         BRLT     Others     ;    if(type < 0) non valide
         CPA      6,i        ;
         BRGT     Others     ;    if(type >6) non valide
         SUBA     1,i        ;    numéro cas-1
         ASLA                ;    * 2 (taille des éléments de la table)
         STA      type,s     ;
         LDX      type,s     ;    => adresse dans la table
         BR       TabCas,x   ;    switch(type) {
; table des cas placée dans le code
TabCas:  .ADDRSS  Cas1
         .ADDRSS  Cas2
         .ADDRSS  Cas3
         .ADDRSS  Cas4
         .ADDRSS  Cas5
         .ADDRSS  Cas6
Cas1:    LDA      duree,s    ;    case 1:
IF01:    CPA      12,i       ;      if(Durée <= 12)
         BRGT     ELSE01     ;
THEN01:  LDA      1500,i     ;        Frais courants = 15.00;
         BR       ENDIF01    ;      else
ELSE01:  LDA      4500,i     ;        Frais courants = 45.00;
ENDIF01: BR       FinCas     ;
Cas2:    LDA      duree,s    ;    case 2: if(Durée >= 9)
IF02:    CPA      9,i        ;
         BRLT     ENDIF02    ;
THEN02:  ADDA     3,i        ;              Frais courants = (Durée + 3)
         ASRA                ;                                / 4
         ASRA                ;
         ASLA                ;                               *12$
         ASLA                ;
         STA      type,s     ;               (multiplie par 4
         ADDA     type,s     ;               et ajoute 3 fois!)
         ADDA     type,s     ;
IF03:    CPA      7200,i     ;             if(Frais courants > 72)
         BRLE     ENDIF03    ;
THEN03:  LDA      7200,i     ;               Frais courants = 72;
ENDIF03: BR       FinCas     ;
ENDIF02: BR       FinCas     ;
Cas3:    LDA      duree,s    ;    case 3 : Frais courants = Durée
         STA      -4,s       ;
         LDA      375,i      ;
         STA      -6,s       ;
         ADDSP    -6,i       ;
         CALL     Multipli   ;                              * 3.75$
         LDA      0,s        ;
         ADDSP    2,i        ;           Désempile résultat multiplication
         BR       FinCas     ;
Cas4:    LDA      1200,i     ;    case 4 : Frais courants = 12.00 ;
         BR       FinCas     ;
Cas5:    LDA      duree,s    ;    case 5 : Type5();
         CALL     Type5      ;
         BR       FinCas     ;
Cas6:    LDA      1200,i     ;    case 6 : Frais courants = 12.00 ;
BR       FinCas     ;
Others:  LDA      -1,i       ;    default : code d'erreur
         STA      code,s     ;
         BR       FinBouc    ;              break ;
                             ;    }// switch
```

```
FinCas:  CPA      cout,s     ;    if(Frais courants > Frais maximum)
         BRLE     ENDIF04    ;
THEN04:  STA      cout,s     ;      Frais maximum = Frais courants;
ENDIF04: LDX      compte,s   ;    index++;
         ADDX     2,i        ;    prochain type de service
         STX      compte,s   ;
         CPX      10,i       ;  if(index -> 5) break;
         BRLT     Boucle     ;  }// while
FinBouc: LDA      cout,s     ;  Retourne Frais maximum
         LDX      0,i        ;
         STA      res,sxf    ;  résultat
         LDA      retour,s   ;  adresse retour
         STA      liste,s    ;  déplacée
         LDA      regA,s     ;  restaure A
         LDX      regX,s     ;  restaure X
         ADDSP    18,i       ;  nettoyer pile
         RET0                ;  return ;
;***************************************************************
;  Sous-programme interne pour le traitement du type 5
;  Le registre A comprend la durée.
Type5:   CPA      4,i        ;  if(durée > 4)
         BRLT     ELSIF06    ;
THEN05:  LDA      4500,i     ;    Frais courants = 45.00;
         BR       ENDIF05    ;
ELSIF06: CPA      3,i        ;  else if(Durée == 3)
         BRLT     ELSIF07    ;
THEN06:  LDA      3450,i     ;    Frais courants = 34.50;
         BR       ENDIF05    ;
ELSIF07: CPA      2,i        ;  else if(Durée == 2)
         BRLT     ELSIF08    ;
THEN07:  LDA      2250,i     ;    Frais courants = 22.50;
         BR       ENDIF05    ;
ELSIF08: CPA      1,i        ;  else if(Durée == 1)
         BRLT     ENDIF05    ;
THEN08:  LDA      1200,i     ;    Frais courants = 12.00;
ENDIF05: RET0                ;  return;
;***************************************************************
;  Sous-programme interne pour la multiplication de deux entiers
;  Appel:  réserver espace résultat (mot)
;          empiler premier entier
;          empiler second entier
;          CALL     Multipli
;  Retour: résultat au sommet de la pile
;
sauveX:  .EQUATE  0
sauveA:  .EQUATE  2
adRetour:.EQUATE  4
op2:     .EQUATE  6         ; opérande 2
op1:     .EQUATE  8         ; opérande 1
resu:    .EQUATE  10        ; résultat
;
Multipli: SUBSP   4,i        ; espace local sauvegarde
          STA     sauveA,s   ; sauvegarde A
          STX     sauveX,s   ; sauvegarde X
          LDA     0,i        ;
          STA     resu,s     ; res = 0;
          LDX     op1,s      ; compte = op1;
          CPX     0,i        ; if(op1 == 0)
          BREQ    Fini       ;   résultat nul
          LDA     op2,s      ; additioner op2
          STA     resu,s     ;
          CPA     0,i        ; if(op2 == 0)
          BREQ    Fini       ;     résultat nul
          CPX     op2,s      ; if(op1 < op2){
          BRLE    Calcul     ;
Echange:  LDX     op2,s      ;   compte = op2;
          LDA     op1,s      ;   additioner op1
          STA     resu,s     ;
Calcul:   ADDA    resu,s     ; for(int i = 1; i <= Compte-1; i++){
          BRV     Deborde    ;
          SUBX    1,i        ;   A = A + res;
```

```
          CPX     1,i         ;
          BRNE    Calcul      ; }
          STA     resu,s      ;
Fini:     LDA     adRetour,s  ; adresse retour
          STA     op1,s       ; déplacée
          LDA     sauveA,s    ; restaure A
          LDX     sauveX,s    ; restaure X
          ADDSP   8,i         ; nettoyer pile
          RET0                ; return
Deborde:  LDA     65535,i     ;
          STA     resu,s      ; res = 65535;
          BR      Fini        ;
         .END
```

Figure 10.2 Programme Facture

10.2 Exercices

10.2.1 Établir un ensemble de données permettant de vérifier le programme Facture.

10.2.2 Utilisez le programme de conduite ("driver") de la figure 9.1 permettant d'appeler le programme Facture pour en vérifier le fonctionnement.

10.2.3 Écrire un sous-programme permettant d'imprimer le contenu de mots mémoire situés entre deux adresses données (vidange partielle).

10.3 Sous-programme récursif : permutations

Une permutation d'un ensemble de caractère comprend des ensembles ordonnés de tous ces caractères, chacun dans un ordre particulier. Dans certaines applications, on veut engendrer toutes les permutations possibles d'un ensemble de données. Ceci peut être fait automatiquement au moyen d'un algorithme bien connu, illustré par le code C++ suivant:

```
void Permuter(string liste, int k){
  if(k == MAX) {
    i = 0;
    while(i < MAX &&
          liste[i]!= NULL) {
      cout << liste[i];
      i++;
    } //while
  } // if
  else
    for(int i=k-1; i < MAX; i++){
      Echanger(liste [k], liste [i]);
      Permuter(liste, k+1);
    }//for
}
```

Le sous-programme `Permuter` ne fait que réaliser cet algorithme. Nous avons fixé à six la taille maximum de la chaîne à permuter. Pour lancer le traitement, il suffit d'exécuter l'appel :

`Permuter(initial,1);` avec :

```
initial: .ASCII  "ABCDEF\x00"
```

Le sous-programme suit. La seule chose digne de noter est l'adressage pile indexé, qui est utilisé pour accéder aux caractères de la chaîne située sur la pile.

```
;  Génération de permutations.
;    Philippe Gabrini septembre 2005
;
NEWLINE: .EQUATE 0x0A       ; saut de ligne
MAX:     .EQUATE 6          ; taille maximum de la chaîne
Start:   STRO    messa,d    ;  cout << "début permutations"
         CHARO   NEWLINE,i  ;       << endl;
         LDA     initial,d  ; empiler chaîne (pourrait être fait par boucle)
         STA     -6,s       ;
         LDA     suite,d    ;
         STA     -4,s       ;
         LDA     fin,d      ;
         STA     -2,s       ;
         LDA     1,i        ;
         STA     -8,s       ;
         SUBSP   8,i        ;
         CALL    Permuter   ;  Permuter(initial,1);
         CHARO   NEWLINE,i  ;
         STRO    fini,d     ;  cout >> "fin du programme"
         CHARO   NEWLINE,i  ;       >> endl;
         STOP
initial: .ASCII  "AB"       ;  chaîne ABCDEF
suite:   .ASCII  "CD"
fin:     .ASCII  "EF\x00"
messa:   .ASCII  "début permutations\x00"
fini:    .ASCII  "fin du programme\x00"
; Le sous-programme Permuter génère les permutations des caractères
; d'une chaîne de caractères d'au plus 6 caractères.  Appel:
;        LDA    fin chaîne(2 caractères)   ;
;        STA    -2,s                       ;
;        LDA    milieu chaîne(2 caractères);
;        STA    -4,s                       ;
;        LDA    début chaîne(2 caractères) ;
;        STA    -6,s                       ;
;        LDA    longueur,i                 ;
;        STA    -8,s                       ;
;        SUBSP  8,i                        ;
;        CALL   Permuter                   ;  Permuter(chaîne, longueur)
; Résultat: les permutations sont affichées
Pindex:  .EQUATE 0          ; index
Pcar1:   .EQUATE 2          ; 1 caractère
Pcar2:   .EQUATE 3          ; 1 caractère
PvieuxX: .EQUATE 4          ; sauvegarde X
PvieuxA: .EQUATE 6          ; sauvegarde A
PadRet:  .EQUATE 8          ; adresse de retour
Pk:      .EQUATE 10         ; nombre de caractères
Pliste:  .EQUATE 12         ; début chaîne
Pliste2: .EQUATE 14         ; milieu chaîne (pour copie facile)
Pliste3: .EQUATE 16         ; fin chaîne (pour copie facile)
;                           ;void Permuter(string Liste, int k){
Permuter:SUBSP   8,i        ;  espace local (comptez bien!)
         STA     PvieuxA,s  ;  sauve
         STX     PvieuxX,s  ;  sauve
         LDA     Pk,s       ;  if(k == MAX)
         CPA     MAX,i      ;
         BRLT    Boucle     ;
         LDA     0,i        ;
         LDX     0,i        ;    while(caractère != NULL
Repete:  LDBYTEA Pliste,sx  ;
         BREQ    Pretour    ;
         CPX     MAX,i      ;          && i < MAX) {
         BRGE    Pretour    ;
         CHARO   Pliste,sx  ;      cout << Liste[i];
         ADDX    1,i        ;      i++;
         BR      Repete     ;    } //while
```

```
Boucle:  LDX     Pk,s       ;  else
         SUBX    1,i        ;    for(int i=k-1; i < MAX; i++){
         STX     Pindex,s   ;
Recurse: CPX     MAX,i      ;      Échanger(Liste [k], Liste [i]);
         BRGE    Pretour    ;
         LDBYTEA Pliste,sx  ;
         STBYTEA Pcar1,s    ;      Pcar1 = Liste[i];
         LDX     Pk,s       ;
         SUBX    1,i        ;      basé zéro
         LDBYTEA Pliste,sx  ;
         STBYTEA Pcar2,s    ;      Pcar2 = Liste[k];
         LDBYTEA Pcar1,s    ;
         STBYTEA Pliste,sx  ;      Liste[k] = Pcar1;
         LDX     Pindex,s   ;
         LDBYTEA Pcar2,s    ;
         STBYTEA Pliste,sx  ;      Liste[i] = Pcar2;
         LDA     Pliste3,s  ;      préparation appel récursif
         STA     -2,s       ;      fin chaîne
         LDA     Pliste2,s  ;
         STA     -4,s       ;      milieu chaîne
         LDA     Pliste,s   ;
         STA     -6,s       ;      début chaîne
         LDA     Pk,s       ;
         ADDA    1,i        ;      k + 1
         STA     -8,s       ;
         SUBSP   8,i        ;
         CALL    Permuter   ;      Permuter(Liste, k+1);
         ADDX    1,i        ;
         STX     Pindex,s   ;
         BR      Recurse    ;    }//for
Pretour: CHARO   NEWLINE,i  ;  //if
         LDA     PadRet,s   ;  déplace adresse retour
         STA     Pliste3,s  ;
         LDA     PvieuxA,s  ;  restaure
         LDX     PvieuxX,s  ;
         ADDSP   16,i       ;  nettoyer pile
         RET0               ;}// Permuter
         .END
```

L'exécution de ce sous-programme produit une sortie avec 720 permutations, commençant par :

```
ABCDEF
ABCDFE
ABCEDF
ABCEFD
ABCFDE
ABCFED
ABDCEF
ABDCFE
```

Continuant plus loin par :

```
DFABCE
DFABEC
DFACBE
DFACEB
DFAEBC
DFAECB
DFBACE
DFBAEC
```

Et se terminant par :

```
FECBAD
FECBDA
FECDAB
FECDBA
FEDABC
FEDACB
FEDBAC
FEDBCA
FEDCAB
FEDCBA
```

