IUT 'A' Paul SABATIER
Dépt Informatique

ASR => Architecture des ordinateurs : Ass2

Langage d'assemblage des microprocesseurs ARM

---

corrigé du TD4 : Les sous-programmes

J.P.CARRARA - R.FACCA - P.MAGNAUD

Exercice 1 :

Soit la procédure **`libererChaine`** définie dans le TP de Compléments de C sur les chaînes de caractères dynamiques et dont le prototype en C est le suivant :

void `libererChaine`(ChaineDyn * *ch) ;*

Cette procédure libère le bloc d'octets alloué à la chaîne ***ch*** en mémoire dynamique et réinitialise ***ch*** en chaîne vide. Son algorithme est le suivant :

procédure `libererChaine` (**mise-à-jour** ch <ChaineDyn>)

Glossaire

libérer <fonction> : utiliser la fonction **`free`** de la bibliothèque C

début

libérer(ch.ptrCar) ;
ch.nbCar ← 0 ;
ch.ptrCar ← **NULL** ;

fin

1°)- Définir en langage d'assemblage ARM, le fichier **`defTypes.s`** qui contiendra les déclarations de la constante symbolique **NULL** et du type enregistrement suivant :

```
typedef  char *  PointeurCar;

typedef  struct
{
  int  nbCar;                    /* nombre de caractères de la chaîne */
  PointeurCar   ptrCar;          /* pointeur sur le tableau de caractères contenant la suite de caractères
                                    ASCII constituant la chaîne */
} ChaineDyn;
```

2°)- Ecrire l'algorithme détaillé et le glossaire de ce S/P sachant qu'il est appelé par le programme **`testChaine`** en langage C.

3°)- Donner le codage de ce S/P en langage d'assemblage ARM, sous forme d'une unité de compilation complète.

1°)- Fichier **`defTypes.s`** (à inclure en tête des S/P)

```
@ définition de symboles

    .equiv   NULL, 0

@ définition de l'enregistrement du type <ChaineDyn>

    .equiv   nbCar, 0    @ entier naturel 32 bits, nombre de caractères de la chaîne
    .equiv   ptrCar, 4   @ pointeur 32 bits, pointe sur le tableau dynamique contenant
                         @ la suite des caractères ASCII constituant la chaîne
```

2°)- et 3°)-

Algorithme détaillé

glossaire

paramètres**:**

R0 <=> ch : passage par adresse

variables locales **:**

aucune

début

```
empiler(R4, LR)
R4  ← R0
R0 ← (R4 + #ptrCar)↑
appel free
R1 ←  #0
(R4 + #nbCar)↑  ← R1
R1 ←  #NULL
(R4 + #ptrCar)↑ ← R1
dépiler(R4, PC)
```

fin

Assembleur

```
@ S/P libererChaine

        .include    defTypes.s

@ parametres
@ r0 <=> ch : passage par adresse


        .global   libererChaine

        .text
        .align   2
libererChaine:
        stmfd    sp!, {r4, lr}
        mov      r4, r0
        ldr      r0, [r4, #ptrCar]
        bl       free
        mov      r1, #0
        str      r1, [r4, #nbCar]
        mov      r1, #NULL
        str      r1, [r4, #ptrCar]
        ldmfd    sp!, {r4, pc}
```

Exercice 2 :

Soit la procédure **`copierChaine`** définie dans le TP de Compléments de C sur les chaînes de caractères dynamiques et dont le prototype en C est le suivant :

void `copierChaine` (**ChaineDyn** * *ch1*, **const ChaineDyn** *ch2* , **jmp_buf** *ptRep*);

Cette procédure affecte la chaîne *ch2* à la chaîne *ch1*. Elle lève l'exception ″ERR_ALLOC″ en cas d'erreur d'allocation. Si une exception est levée la chaîne *ch1* ne sera pas modifiée.
Son algorithme est le suivant :

procédure **`copierChaine`** (**mise-a-jour** ch1 <ChaineDyn>, **entrée** ch2 < ChaineDyn >)

Glossaire

lgCh2 <Entier> : longueur de la chaîne ch2 ;
ptrAlloc <PointeurCar> : pointeur sur le bloc d'octets alloué dynamiquement;
copierBlocOctets <fonction> : utiliser la fonction **`memcpy`** de la bibliothèque C

début

```
lgCh2 ← ch2.nbCar;
si ( lgCh2 = 0 ) alors
        ptrAlloc ← NULL;
sinon
        ptrAlloc ← allouer( lgCh2 );
        si ( ptrAlloc = NULL ) alors
                lever ( ERR_ALLOC );
        finsi;
        copierBlocOctets( ptrAlloc, ch2.ptrCar, lgCh2 );
finsi;

si ( ch1.ptrCar!= NULL ) alors
        libérer (ch1.ptrCar);
finsi;
ch1.nbCar ←  lgCh2;
ch1.ptrCar ←  ptrAlloc;
```

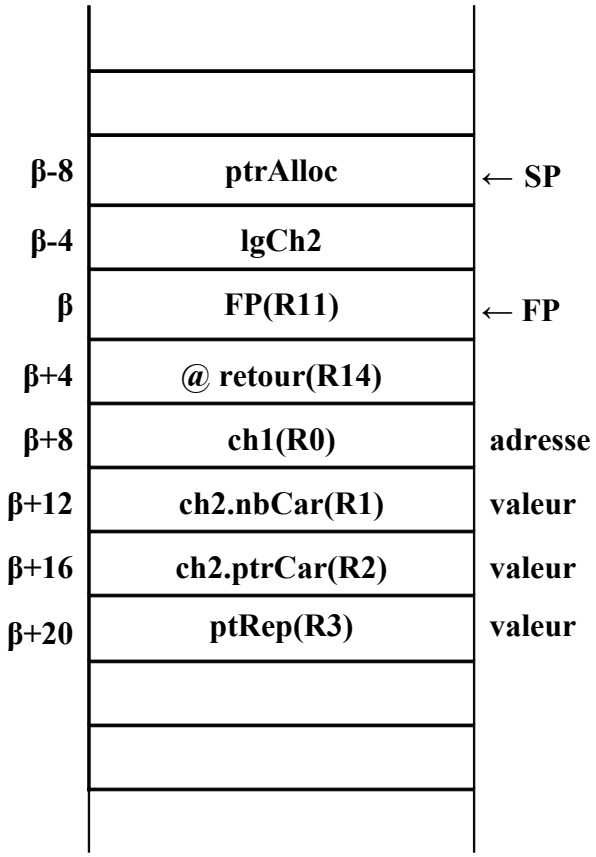
fin

1°)- Ecrire l'algorithme détaillé et le glossaire de ce S/P sachant qu'il est appelé par le programme **`testChaine`** en langage C.

2°)- Donner le codage de ce S/P en langage d'assemblage ARM, sous forme d'une unité de compilation complète.

Organisation de la pile système en début du S/P

on utilisera le modèle proposé au chap. 3 § 4.4 du cours.



glossaire de l'algorithme détaillé

paramètres formels :

ch1 : position relative du paramètre effectif ch1 = 8 , passage par **adresse** ;
ch2 : position relative du paramètre effectif ch2 = 8 + 4 , passage par **valeur** ;
ptRep: position relative du paramètre effectif ptRep = 8 + 12 , passage par **valeur** ;

variables locales :

lgCh2 : position relative de la variable locale lgCh2 = - 4
ptrAlloc : position relative de la variable locale ptrAlloc = - 8

Algorithme détaillé

début
    empiler (R0-R3)
    empiler (FP,LR)
    FP←SP
    SP←SP sub 2*4
    R0←(FP+#ch2+nbCar)↑
    (FP+#lgCh2)↑←R0
    si R0=#0 **alors**
        R1←NULL
        (FP+#ptrAlloc)↑←R1
    sinon
        appel malloc
        (FP+#ptrAlloc)↑←R0
        si R0 = #NULL **alors**
            R0←(FP+#ptRep)↑
            R1←.L_ERR_ALLOC
            appel longjmp
        finsi;
        R0←(FP+#ptrAlloc)↑
        R1←(FP+#ch2+ptrCar)↑
        R2←(FP+#lgCh2)↑
        appel memcpy
    finsi;
    si (R0←(FP+#ch1)↑, R0←(R0+#ptrCar)↑, R0 ≠ #NULL)
    alors
        appel free
    finsi;
    R0←(FP+#ch1)↑
    R1←(FP+#lgCh2)↑
    (R0+#nbCar)↑←R1
    R1←(FP+#ptrAlloc)↑
    (R0+#ptrCar)↑←R1
    SP←FP
    dépiler(FP,LR)
    SP←SP add 4*4
    PC←LR
fin

Assembleur

```
@S/P  copierChaine
     .include    deftype.s
     .global     copierChaine

     .text
     .align      2
.L-ERR_ALLOC:
      .asciz "Echec malloc"
     .align      2
copierChaine:
   stmfd   sp!,{ro-r3}
   stmfd   sp!,{fp,lr}
   mov     fp,sp
   sub     sp,sp,#2*4
   ldr     r0,[fp,#ch2+nbCar]
   str     r0,[fp,#lgCh2]
.L_si1:
   cmp     r0,#0
   bne     .L_sinon1
   mov     r1,#NULL
   str     r1,[fp,#ptrAlloc]
   b       .L_finsi1
.L_sinon1:
   bl      malloc
   str     r0,[fp,#ptrAlloc]
.L_si2:
   cmp     r0,#NULL
   bne     .L_finsi2
   ldr     r0,[fp,#ptRep]
   ldr     r1,.L_ERR_ALLOC
   bl      longjmp
.L_finsi2:
   ldr     r0,[fp,#ptrAlloc]
   ldr     r1,[fp,#ch2+ptrCar]
   ldr     r2,[fp,#lgCh2]
   bl      memcpy
.L_finsi1:
.L_si3:
   ldr     r0,[fp,#ch1]
   ldr     r0,[r0,#ptrCar]
   cmp     r0,#NULL
   bne     .L_finsi3
   bl      free
.L_finsi3:
   ldr     r0,[fp,#ch1]
   ldr     r1,[fp,#lgCh2]
   str     r1,[r0,#nbCar]
   ldr     r1,[fp,#ptrAlloc]
   str     r1,[r0,#ptrCar]
   mov     sp,fp
   ldmfd   sp!,{fp,lr}
   add     sp,sp,#4*4
   mov     pc,lr
```