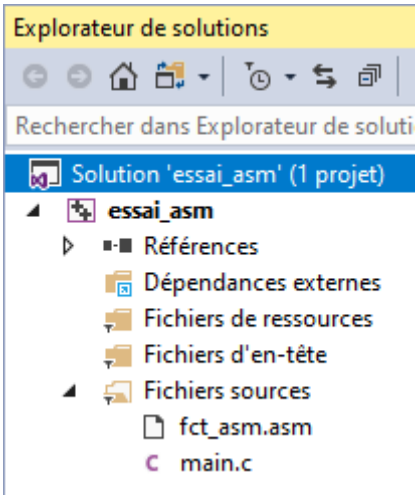
Liste d'exercices 5 : les fonctions en assembleur avec passage des paramètres par valeur (1 séance)

Notions à considérer pour pouvoir faire les exercices

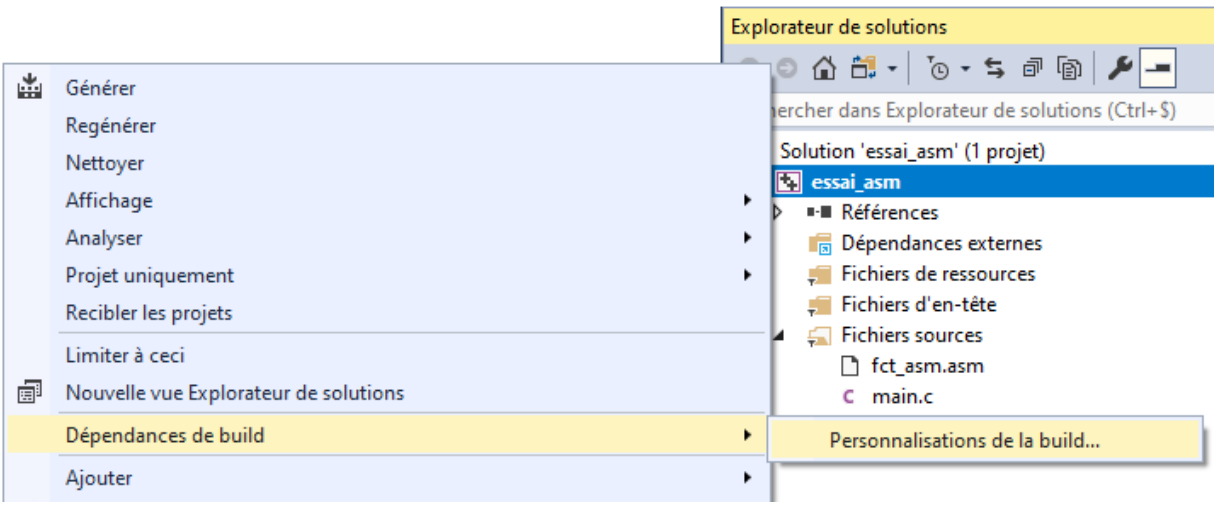
Création d'un projet multi-fichiers :

1. Ajouter 2 fichiers source dans le projet. Dans la capture d'écran suivante, un fichier est nommé fct_asm.asm et l'autre est nommé main.c.

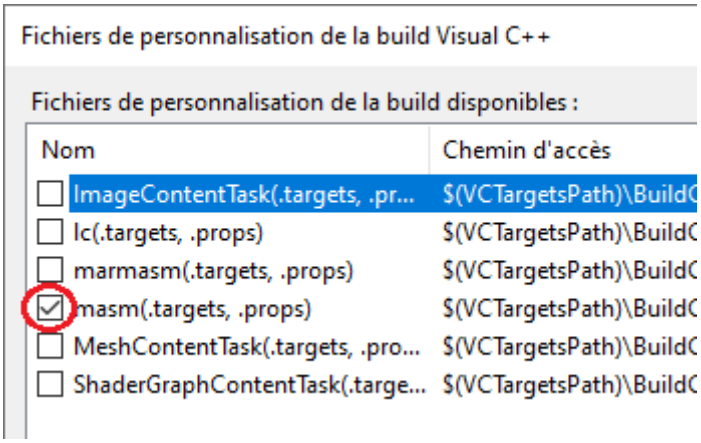


   Le fichier dont l'extension est *asm* va contenir les fonctions écrites en assembleur. Le fichier dont l'extension est *c* va contenir la fonction main() depuis laquelle les fonctions écrites en assembleur seront invoquées. **Ces 2 fichiers doivent être nommés différemment**. Par exemple, si on nomme les 2 fichiers *source*.asm et *source*.c, des erreurs vont survenir lors de la génération du fichier exécutable.

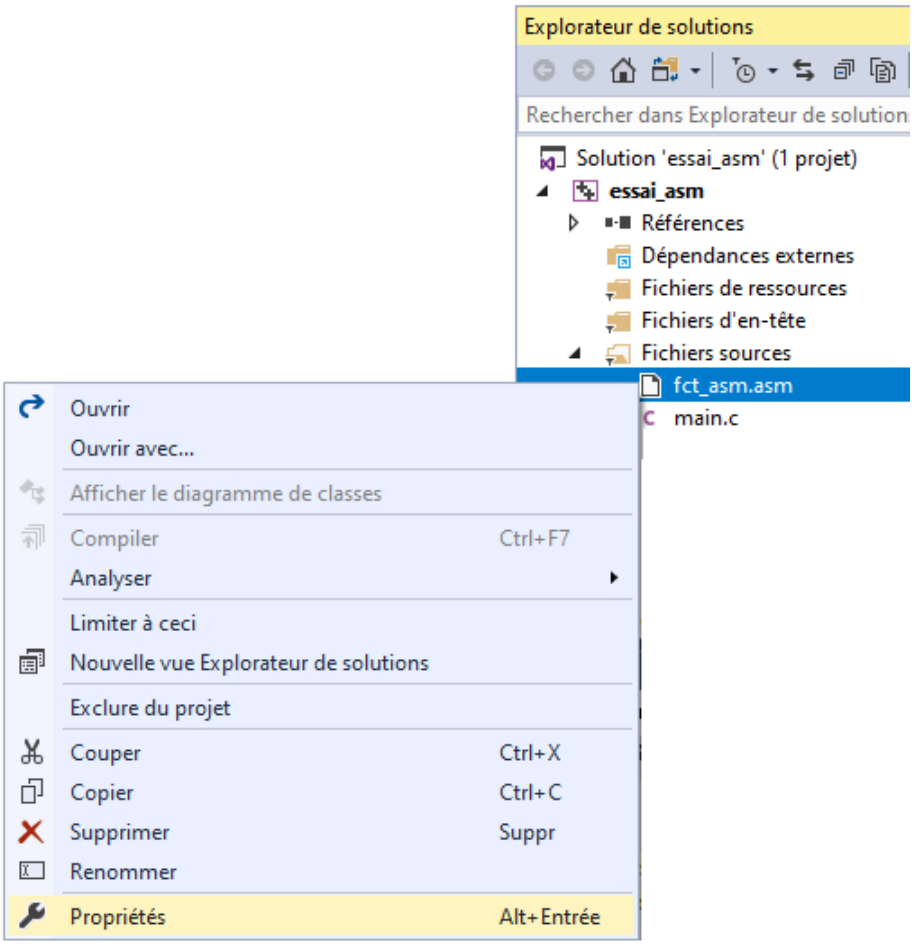
2. Dans la fenêtre Explorateur de solutions, cliquez avec le bouton droit sur le nom du projet et sélectionnez *Dépendances de build*, puis *Personnalisations de la build*.



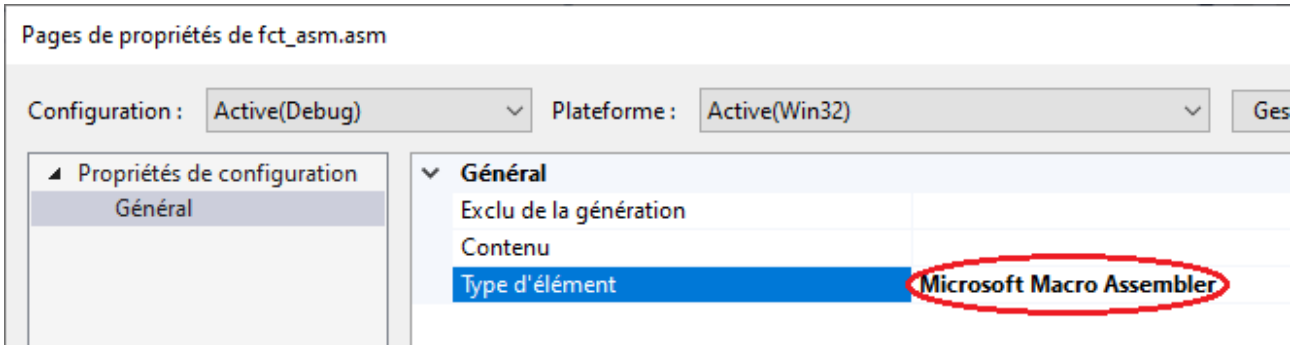
3. Cochez dans la fenêtre Fichiers de personnalisation de la build Visual C++ l'option *masm (.targets, .props)*, puis cliquez sur Ok.



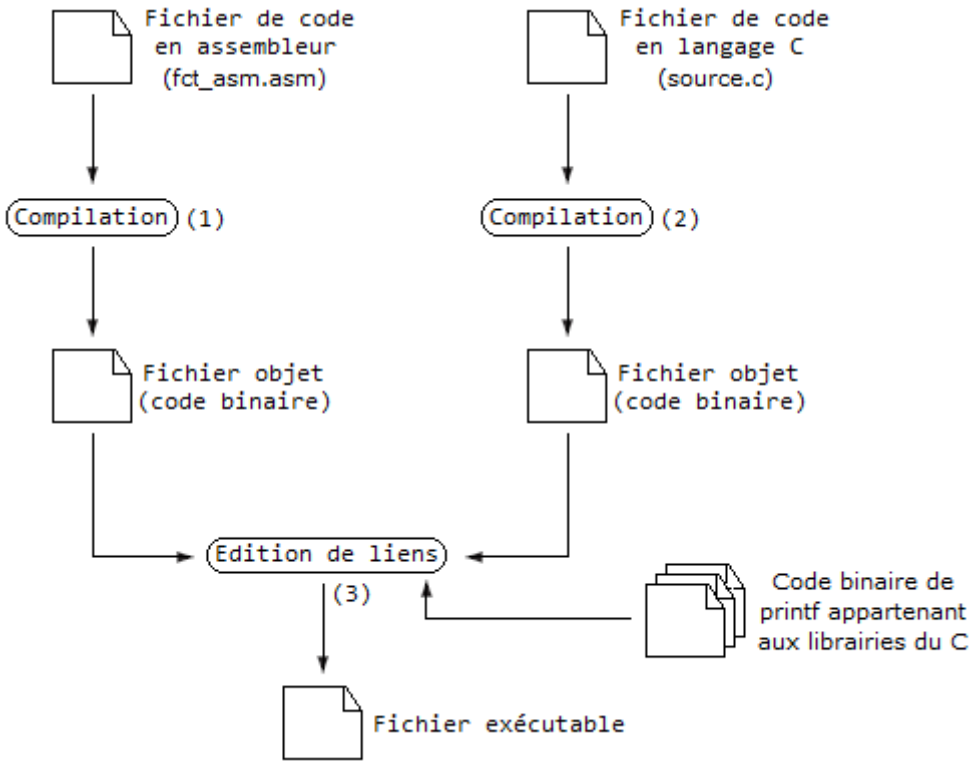
4. Dans la fenêtre Explorateur de solutions, cliquez avec le bouton droit sur le fichier fct_asm.asm et sélectionnez *Propriétés :*



5. Choisissez dans la fenêtre Pages de propriétés de fct_asm.asm, l'option *Microsoft Macro Assembler* comme type d'élément, puis cliquez sur Ok.



Quand on génère le fichier exécutable à partir d'un projet constitué de 2 fichiers de code source, voici ce qui se passe :



(1) Le compilateur, à partir du code en assembleur figurant dans le fichier fct_asm.asm, génère un "fichier objet" contenant le code en langage machine équivalent.

(2) Le compilateur, à partir du code en langage C figurant dans le fichier source.c, génère un second "fichier objet" contenant le code en langage machine équivalent.

(3) L'éditeur de liens prend les instructions en langage machine figurant dans les 2 fichiers objet, ainsi que les instructions en langage machine de fonctions des librairies du C éventuellement invoquées (printf, scanf, etc.), puis il génère un fichier exécutable contenant l'ensemble de ces instructions en langage machine.

Structure d'un fichier de code en assembleur :

```
.MODEL FLAT, C
.DATA

   ; variables globales ...

.CODE

Mafonction PROC

   ; instructions de la fonction ...

Mafonction ENDP

END
```

- Un commentaire dans un fichier de code en assembleur débute avec le symbole ;.

- Les variables globales sont déclarées dans la section .DATA. Cette section n'est pas nécessaire quand aucune variable globale n'est déclarée.

  La déclaration d'une variable globale a la forme : nom DIRECTIVE valeur.

Directives	*Tailles*
DB	1 octet
DW	2 octets
DD	4 octets
DQ	8 octets

- Une fonction figure dans la section .CODE. Elle commence par son nom suivi de PROC et se termine par son nom suivi de ENDP. On peut bien entendu placer plusieurs fonctions dans la section .CODE.

- END est placé tout à la fin pour terminer un fichier de code en assembleur.

Le registre ESP :

Nous allons faire 2 usages distincts du registre ESP :

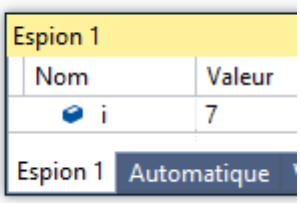
- Le registre ESP est le registre qui permet les accès à la pile selon le principe LIFO (last in, first out) lors de l'utilisateur des instructions PUSH et POP, car il mémorise l'adresse de l'élément situé au sommet de la pile. Pour réserver des emplacements dans la pile, on diminue l'adresse se trouvant dans ESP, et, pour libérer des emplacements dans la pile, on augmente l'adresse se trouvant dans ESP.

- Le registre ESP peut aussi servir en tant que pointeur vers des emplacements dans la pile qui nous intéressent. Avec ESP, on va faire des accès indirects dans la pile sans tenir compte du principe LIFO.

Exemple : la fonction SommeInt() retourne la somme de 2 entiers passés en paramètre.

Code en langage C invoquant la fonction SommeInt() :

```
1   extern int SommeInt(int i1, int i2);
2
3   void main()
4   {
5       int i;
6
7       i = SommeInt(3, 4);
8   }
```



Fonction SommeInt() en assembleur :

```
1   .MODEL FLAT, C
2   .CODE
3
4   ; Fonction qui calcule la somme de 2 nombres du type int
5   ; Prototype en C : int SommeInt(int i1, int i2)
6   ; Params : i1 et i2 sont les 2 nombres à additionner
7   ; Retour : la somme
8
9   SommeInt PROC
10      i2  EQU <DWORD PTR [ESP + 8]>
11      i1  EQU <DWORD PTR [ESP + 4]>
12
13      mov  eax, i1
14      add  eax, i2
15
16      ret
17  SommeInt ENDP
18
19  END
```

Analyse de l'exécution du programme :

1. Dans la fonction main(), les instructions qui correspondent à l'appel de fonction i = SommeInt(3, 4) générées par le compilateur C sont les suivantes :

   ```
   push dword ptr 4
   push dword ptr 3
   call dword ptr SommeInt
   add  esp, 8
   mov  i, eax
   ```

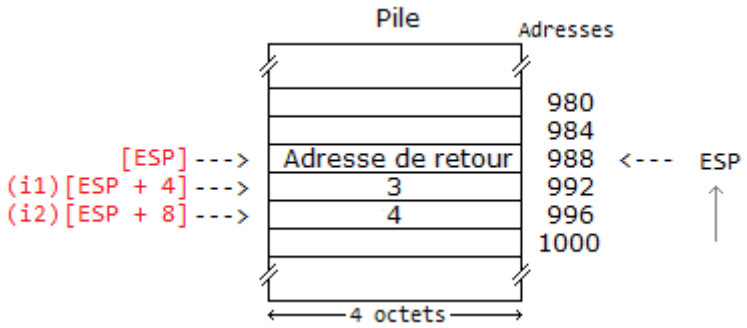
2. Dans la fonction en assembleur SommeInt(), avec la directive EQU, on assigne un nom à chacun des 2 emplacements dans la pile stockant les valeurs passées en paramètre :

   ```
   i2  EQU  <DWORD PTR [ESP + 8]>
   i1  EQU  <DWORD PTR [ESP + 4]>
   ```

   La directive EQU fonctionne comme #define en C. Les identificateurs i1 et i2 figurant dans le code en assembleur seront remplacés, lors de la compilation, par ce à quoi ils correspondent. Plutôt que d'utiliser DWORD PTR [ESP + 4] ou DWORD PTR [ESP + 8] comme opérande dans certaines des instructions de la fonction SommeInt(), on peut alors

utiliser les identificateurs i1 ou i2. Les chevrons < et > sont des délimiteurs.

Voici l'état de la pile lorsque commence l'exécution de la fonction SommeInt() (on suppose que ESP stocke initialement l'adresse 1000) :



- La valeur 4 passée en paramètre se trouve dans l'emplacement de 4 octets dont l'adresse est 996. Cet emplacement est la variable appelée i2 dans la fonction SommeInt().

- La valeur 3 passée en paramètre se trouve dans l'emplacement de 4 octets dont l'adresse est 992. Cet emplacement est la variable appelée i1 dans la fonction SommeInt().

- L'adresse de retour à la fonction main() se trouve dans l'emplacement de 4 octets dont l'adresse est 988. Elle a été placée dans la pile lors de l'exécution de l'instruction CALL DWORD PTR SommeInt. Plus spécifiquement, c'est l'adresse dans la fonction main() de l'instruction ADD ESP, 8 qui supprime les paramètres de la pile.

- ESP pointe vers l'emplacement dont l'adresse est 988 qui est l'emplacement au sommet de la pile. Si, par exemple, ESP est utilisé comme pointeur, avec l'instruction MOV EAX, DWORD PTR [ESP], on copie dans EAX la valeur située au sommet de la pile.

3. MOV EAX, i1 copie dans EAX la valeur de la variable i1, c'est-à-dire la valeur 3. Cette instruction est, en réalité, MOV EAX, DWORD PTR [ESP + 4].

4. ADD EAX, i2 ajoute à EAX la valeur de la variable i2. Cette instruction est, en réalité, ADD EAX, DWORD PTR [ESP + 8].

5. La valeur retournée par la fonction SommeInt() se trouve dans EAX. Une fonction retourne un entier avec le registre EAX.

6. Le retour à la fonction main() est réalisé avec RET. Cette instruction dépile l'adresse de retour à main() et elle place cette adresse dans EIP qui est le registre pointant vers la prochaine instruction à charger en mémoire. Ceci entraîne un retour à la fonction main().

7. Dans la fonction main(), l'instruction ADD ESP, 8 libère de la pile les 8 octets occupés par les paramètres.

8. Avec MOV i, EAX, la valeur retournée par SommeInt() est copiée dans la variable i.

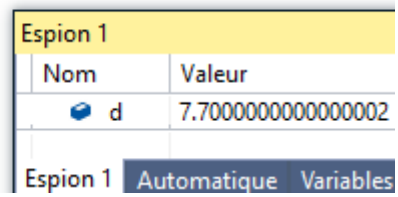
Exemple : la fonction SommeDouble() retourne la somme de 2 doubles passés en paramètre.

Code en langage C invoquant la fonction SommeDouble() :

```c
extern double SommeDouble(double d1, double d2);

void main()
{
    double d;

    d = SommeDouble(3.5, 4.2);
}
```



Fonction SommeDouble() en assembleur :

```
.MODEL FLAT, C
.CODE

; Fonction qui calcule la somme de 2 nombres du type double
; Prototype en C : double SommeDouble(double d1, double d2)
; Params : d1 et d2 sont les 2 nombres à additionner
; Retour : la somme

SommeDouble PROC
    d2     EQU <QWORD PTR [ESP + 20]>
    d1     EQU <QWORD PTR [ESP + 12]>
    varLoc EQU <QWORD PTR [ESP + 0]>

    sub   esp, 8

    movsd xmm0, d1
    addsd xmm0, d2
    movsd varLoc, xmm0

    fld   varLoc

    add   esp, 8

    ret
SommeDouble ENDP

END
```

Analyse de l'exécution du programme :

1. Dans la fonction main(), les instructions qui correspondent à l'appel de fonction d = SommeDouble(3.5, 4.2) générées par le compilateur C sont les suivantes :

   ```
   push  dword ptr dtmp2 + 4
   push  dword ptr dtmp2
   push  dword ptr dtmp1 + 4
   push  dword ptr dtmp1
   call  dword ptr SommeDouble
   add   esp, 16
   fstp  d
   ```

   On suppose que dtmp1 et dtmp2 sont 2 constantes créées ainsi lors de la compilation pour

pouvoir passer en paramètre les 2 valeurs du type double :

```
const double dtmp2 = 4.2, dtmp1 = 3.5;
```

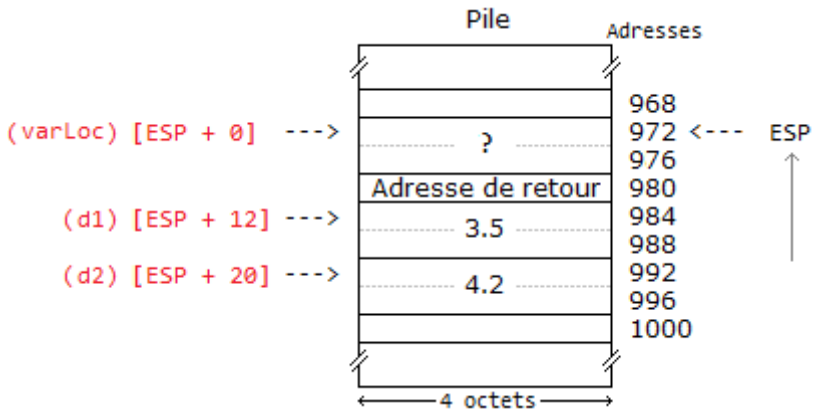
2. Dans la fonction en assembleur SommeDouble(), SUB ESP, 8 réserve 8 octets dans la pile. Ceci **crée dans la pile une variable locale de 8 octets** pour y stocker une valeur du type double.

3. Avec la directive EQU, on a assigné un nom à chacun des 2 emplacements dans la pile stockant les valeurs passées en paramètre ainsi qu'à la variable locale créée dans la fonction :

```
d2     EQU <QWORD PTR [ESP + 20]>
d1     EQU <QWORD PTR [ESP + 12]>
varLoc EQU <QWORD PTR [ESP + 0]>
```

Plutôt que d'utiliser QWORD PTR [ESP + 0], QWORD PTR [ESP + 4] ou QWORD PTR [ESP + 8] comme opérande dans certaines des instructions de la fonction SommeDouble(), on peut alors utiliser les identificateurs varLoc, d1 ou d2.

Voici l'état de la pile après l'exécution de l'instruction SUB ESP, 8 (on suppose que ESP stocke initialement l'adresse 1000) :



- La valeur 4.2 passée en paramètre se trouve dans l'emplacement de 8 octets dont l'adresse est 992. Cet emplacement est la variable appelée d2 dans la fonction SommeDouble().

- La valeur 3.5 passée en paramètre se trouve dans l'emplacement de 8 octets dont l'adresse est 984. Cet emplacement est la variable appelée d1 dans la fonction SommeDouble().

- L'adresse de retour à la fonction main() se trouve dans l'emplacement de 4 octets dont l'adresse est 980. Elle a été placée dans la pile lors de l'exécution de l'instruction CALL DWORD PTR SommeDouble. Plus spécifiquement, c'est l'adresse dans la fonction main() de l'instruction ADD ESP, 16 qui supprime les paramètres de la pile.

- La variable locale appelée varLoc dans la fonction SommeDouble() occupe l'emplacement de 8 octets dont l'adresse est 972. Comme cette variable est la dernière chose allouée dans la pile, elle constitue le sommet de la pile et ESP contient son

adresse.

4. MOVSD XMM0, d1 copie dans XMM0 la valeur de la variable d1, c'est-à-dire la valeur 3.5. Cette instruction est, en réalité, MOVSD XMM0, QWORD PTR [ESP + 12].

5. ADDSD XMM0, d2 ajoute à XMM0 la valeur de la variable d2, c'est-à-dire la valeur 4.2. Cette instruction est, en réalité, ADDSD XMM0, QWORD PTR [ESP + 20].

6. MOVSD varLoc, XMM0 copie la somme figurant dans XMM0 dans la variable locale varLoc, c'est-à-dire la valeur 7.7. Cette instruction est, en réalité, MOVSD QWORD PTR [ESP + 0], XMM0.

7. FLD varLoc transmet la valeur de la variable locale varLoc à la FPU (Floating-Point Unit) qui est la valeur du type double que retourne la fonction SommeDouble(). Cette instruction est, en réalité, FLD QWORD PTR [ESP + 0]. En C, une fonction retourne un flottant via la FPU (Floating-Point Unit) du processeur.

8. ADD ESP, 8 supprime de la pile la variable locale varLoc.

9. Le retour à la fonction main() est réalisé avec RET. Cette instruction dépile l'adresse de retour à main() et elle place cette adresse dans EIP qui est le registre pointant vers la prochaine instruction à charger en mémoire. Ceci entraîne un retour à la fonction main().

10. Dans la fonction main(), ADD ESP, 16 libère de la pile les 16 octets occupés par les paramètres.

11. Avec FSTP d, la valeur retournée par SommeDouble() est extraite de la FPU et copiée dans la variable d.

2 remarques :

- Lorsque, dans un fichier contenant du code en langage C, on invoque certaines fonctions des librairies, par exemple, la fonction scanf, il peut arriver que survienne le problème suivant lors de la compilation : *Erreur C4996 'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.* Si cela survient, il faut ajouter au début du fichier de code en langage C la ligne suivante :

  ```
  #pragma warning(disable:4996)
  ```

- Si une valeur du type double est nécessaire dans une fonction écrite en assembleur, il suffit de déclarer une variable globale à laquelle cette valeur est assignée.

  Exemple : variable globale dont la valeur est le nombre e :

```
.MODEL FLAT, C
.DATA

   nbre_e DQ 2.718

.CODE

Mafonction PROC

   movsd xmm0, nbre_e
   ...

Mafonction ENDP

END
```

Exercices

1. Écrire une fonction en assembleur qui reçoit en paramètre une température en degrés Celsius et qui retourne la température en degrés Fahrenheit. La formule de conversion est F = C * 9 / 5 + 32.

   Prototype de la fonction : `int Celius2Fahrenheit(int celcius);`

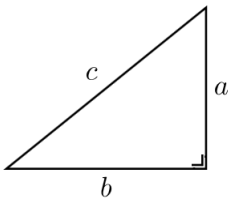
2. Écrire une fonction en assembleur qui reçoit en paramètre les valeurs pour R et I et qui retourne la valeur pour U. La loi d'Ohm est "la tension U (en volts) aux bornes d'une résistance R (en ohms) est proportionnelle à l'intensité du courant électrique I (en ampères) qui la traverse".

   Prototype de la fonction : `int LoiOhm(int resistance, int intensite);`

3. Écrire une fonction en assembleur qui reçoit en paramètre le rayon d'un cercle et qui retourne le périmètre de ce cercle. Formule du périmètre du cercle : P = 2 π r, où π correspond à 3.14 (nombre pi).

   Prototype de la fonction : `double PerimetreCercle(double rayon);`

4. Écrire une fonction en assembleur qui reçoit en paramètre la taille des côtés a, b et c d'un triangle et qui retourne un booléen indiquant si le triangle est rectangle ou non (valeur 1 si le triangle est rectangle ou valeur 0 si le triangle n'est pas rectangle). Le théorème de Pythagore est "dans un triangle rectangle, le carré de la longueur de l'hypoténuse est égal à la somme des carrés des longueurs des côtés de l'angle droit".

   Prototype de la fonction : `int TesterPythagore(int cotea, int coteb, int cotec);`

5. Écrire une fonction en assembleur qui reçoit en paramètre 2 valeurs ainsi que l'opération à réaliser sur ces 2 valeurs parmi : 1 pour +, 2 pour -, 3 pour * et 4 pour /, et qui retourne le résultat de l'opération.

   Prototype de la fonction : `double Calculer(double val1, double val2, int operation);`