INF1600 — TP4
Programmation en assembleur et C++

Giovanni Beltrame giovanni.beltrame@polymtl.ca

Polytechnique Montréal
Hiver 2020

Remise :

Voici les détails concernant la remise de ce travail pratique :

- **Méthode :** sur Moodle (une seule remise par groupe).
- **Format :** un seul fichier zip, dont le nom sera **<Nom1>-<Nom2>.zip**.
  Exemple : **Belgana-Be1trame.zip**.
- **Contenue :** le dossier archive dois contenir les fichiers suivants :
  - Rectangle_Perimeter.s
  - Rectangle_Area.s
  - Rectangle_Diagonal.s
  - Cylindre_Perimeter.s
  - Cylindre_Area.s
  - Cylindre_Volume.s
  - Decryption_fct.c

Barème :

Contenu	**Points du cours**
Rectangle_Perimetre.s	0.5
Rectangle_Area.s	0.5
Rectangle_Diagonal.s	0.5
Cylindre_Perimetre.s	0.5
Cylindre_BaseArea.s	0.5
Cylindre_LateralArea.s	0.75
Cylindre_Area.s	0.75
Cylindre_Volume.s	0.75
Tp4_Ex2.c	1.00
Illisibilité du code (peu de commentaires, mauvaise structure...)	jusqu'à -1
Format de remise erroné (irrespect des noms de fichiers demandés, fichiers superflus, etc.)	jusqu'à -1
Retard	-0,025 par heure

1. Exercice 1 :

Travail demandé :

Vous êtes en charge de coder en assembleur six méthodes :

- Rectangle::PerimeterAsm() ;
- Rectangle::AreaAsm() ;
- Rectangle::DiagonalAsm()
- Cylindre::PerimeterAsm() ;
- Cylindre::BaseAreaAsm() ;
- Cylindre::LateralAreaAsm() ;
- Cylindre::AreaAsm() ;
- Cylindre::VolumeAsm()

La réalisation de ces méthodes doit suivre les exemples donnés dans les méthodes correspondantes en C++ :

- Rectangle::PerimeterCpp() ;
- Rectangle::AreaCpp() ;
- Rectangle::DiagonalCpp()
- Cylindre::PerimeterCpp() ;
- Cylindre::BaseAreaCpp() ;
- Cylindre::LateralAreaCpp() ;
- Cylindre::AreaCpp() ;
- Cylindre::VolumeCpp()

Fichiers fournis :

Les fichiers nécessaires à la réalisation du TP sont dans l'archive inf1600_tp4.zip, disponible sur Moodle. Voici la description des fichiers :

Makefile : le makefile utilisé pour compiler et nettoyer le projet ;
Tp4_Ex1.c : programme de test qui utilise les fonctions de référence et celles en assembleur de l'exercice 1;
shape.h : la définition de la classe CShape ;
Rectangle.h : la définition de la classe Rectangle ;
Rectangle.cpp : l'implémentation C++ de la classe Rectangle ;
Rectangle.vtable : la structure de la Virtual table de la classe Rectangle ;
Cylindre.h : la définition de la classe Cylindre ;
Cylindre.cpp : l'implémentation C++ de la classe Cylindre ;
Cylindre.vtable : la structure de la Virtual table de la classe Cylindre.

Vous devez compléter les fichiers *.s et les remettre dans un archive zip. Votre code doit passer chaque tests dans `Tp4_Ex1.cpp`.

Compilation et test :

Pour compiler le programme de test (tp4), il est suffisant de taper :

```
make
```

Pour l'exécuter,

```
./Tp4_Ex1
```

État de la pile au début d'une méthode :

Lorsqu'une méthode M d'une classe C est appelée, le premier argument qui se trouve dans la pile (8(%ebp)) est toujours l'adresse de l'objet de la classe C.

Par exemple, avec cette définition :

```
class  C  {
      public:
         void M(int x);
      private:
         int i;
};
```

quand la méthode M est appelée, la pile est :

12(%ebp)	valeur de x
8(%ebp)	adresse de l'objet de type C
4(%ebp)	ancienne valeur de %eip
(%ebp)	ancienne valeur de %ebp

Opérations avec valeurs float :

Pour cet exercice, on se sert de la partie FPU (unité à virgule flottante) du processeur Intel. Il s'agit d'une pile dédiée au calcul flottant (différente de la pile d'appel), de grandeur 8 (elle peut contenir jusqu'à 8 entrées de type float), mais il est rarement nécessaire de dépasser 2 ou 3 entrées. Les quelques instructions agissent toujours sur le premier et le deuxième éléments de la pile (st[0] et st[1]).

Voici ces instructions :

Instruction	**Rôle**
fld x	Ajoute au-dessus de la pile la valeur à l'adresse mémoire x ; st[1] prend la valeur de st[0] et st[0] devient cette nouvelle valeur chargée de la mémoire.
fldpi x	Ajoute au-dessus de la pile la valeur de π (3,1415. . .).
fstp x	Retire l'élément st[0] pour le mettre en mémoire principale à l'adresse x. st[1] devient st[0].
faddp	st[0] est additionné à st[1] et le résultat remplace ces deux éléments.
fsubp	st[1] est soustrait de st[0] et le résultat remplace ces deux éléments.
fsubrp	st[0] est soustrait de st[1] et le résultat remplace ces deux éléments.
fmulp	st[0] est multiplié avec st[1] et le résultat remplace ces deux éléments.
fdivp	st[0] est divisé par st[1] et le résultat remplace ces deux éléments.
fdivrp	st[1] est divisé par st[0] et le résultat remplace ces deux éléments.
fsqrt	la valeur dans st[0] est remplacée par sa racine carrée

Pour retourner un float en sortant d'une fonction, laissez la valeur sur st[0].

2. Exercice 2 :

Nous désirons passer de big endian à little endian (et vice versa) à l'aide d'une fonction C.
Une façon de ce faire est la suivante :

```
unsigned int change_endianness(unsigned int x){
   unsigned int y;
   y = ( x & 0xff000000 ) >> 24 | (x & 0x00ff0000) >> 8 |
       ( x & 0x0000ff00 ) <<  8 | (x & 0x000000ff) << 24;
}
```

Malheureusement, le compilateur gcc génère trop d'instructions pour que notre programme ait de bonnes performances. Vous pouvez voir le code généré en tapant la commande :

gcc -m32 -S Tp4_Ex2.c

Un fichier `Tp4_Ex2.s` sera généré, il contient le code assembleur correspondant au programme. Votre travail est d'implémenter une routine `change_endianness(unsigned int x)` plus efficacement que le compilateur (en utilisant moins d'instructions) via quelques lignes d'assembleur en ligne dans le code `C`.

Note : vous n'êtes pas supposés réécrire le code précédent en assembleur mais trouver une façon plus efficace de faire la même chose en assembleur. Investiguez l'utilisation de l'instruction `bswap`.

Le fichier `Tp4_Ex2.c` qui contient la fonction à modifier vous est fourni. Vous pouvez compiler le programme avec la ligne de commande suivante :

gcc -Wall -m32 -gdwarf - o Tp4_Ex2 Tp4_Ex2.c

Vous pouvez exécuter votre programme ainsi :

./Tp4_Ex2