INF1600

Travail pratique 4

Programmation en assembleur et C++

Tarek Ould-Bachir

Sami Sadfa

Wail Ayad

Département de Génie Informatique et Génie Logiciel
Polytechnique Montréal

1 Introduction et sommaire

1.1 Remise

Voici les détails concernant la remise de ce travail pratique :

- Méthode : sur Moodle, une seule remise par équipe.
- Format: une archive ZIP, sous la forme <matricule1>-<matricule2>.zip. L'archive doit contenir les fichiers `circle_perimeter.s,circle_area.s,triangle_perimeter.s,triangle_area.s, triangle_height.s.` **(Ne pas modifier le nom des fichiers .s).**
- Langue écrite : Français.
- Distribution : les deux membres de l'équipe recevront la même note.

Attention : L'équipe de deux que vous formez pour ce TP sera définitive jusqu'au TP5. Il ne sera pas possible de changer d'équipe au cours de la session.

1.2 Barème

Le travaux pratiques 1 à 5 sont notés sur 4 points, pour un total de 20/20 Le TP1 est noté selon le barème suivant. Il est possible d'obtenir une note de 4,25/4,00 en répondant correctement à la question bonus. La note des travaux pratiques ne peut cependant pas dépasser 20/20. Reproduisez ce tableau dans le document PDF que vous remettrez.

TP 4			**/4,00**
Exercice 1			/4,00
	circle_perimeter.s	/0,50	
	circle_area.s	/0,50	
	triangle_height.s	/0,50	
	triangle_perimeter.s	/1,00	
	triangle_area.s	/1,50	

2 Exercice 1 : Assembleur x86

Vous devrez implémenter en assembleur cinq fonctions écrites en C++.

- `CCircle::PerimeterAsm()`
- `CCircle::AreaAsm()`
- `CTriangle::PerimeterAsm()`
- `CTriangle::AreaAsm()`
- `CTriangle::HeightAsm()`

Les méthodes correspondantes en C++ sont :

- `CCircle::PerimeterCpp()`
- `CCircle::AreaCpp()`
- `CTriangle::PerimeterCpp()`
- `CTriangle::AreaCpp()`
- `CTriangle::HeightCpp()`

Voici la description des fichiers :

- `tp4.c` : Programme de test qui utilise les fonctions de références et celles en assembleur.
- `shape.h` : la définition de la classe `CShape`.
- `circle.h` : la définition de la classe `CCircle`.
- `circle.cpp` : l'implémentation C++ de la classe `CCircle`
- `circle.vtable` : La structure de la `virtual table` de la classe `CCircle`
- `triangle.h` : : la définition de la classe `CTriangle`.
- `triangle.cpp` : l'implémentation C++ de la classe `CTriangle`
- `triangle.vtable` : la structure de la virtual table de la classe `CTriangle`

Vous devez complétez les fichier *.s et les remettre dans une archive zip. Votre code doit passer chaque test dans tp4.c.

Utilisez la commande pour compiler votre exécutable

```
make
```

Si la compilation lance une erreur *« `ne peut trouver -lstdc++` »*, il faut installer les librairies permettant de compiler du C++ en 32bits avec les deux commandes suivantes

```
sudo dnf install libstdc++-static libstdc++-static.i686
```

et/ou

```
sudo dnf install libstdc++-devel libstdc++-devel.i686
```

Pour l'exécuter, vous devez passer 4 arguments. Le premier est le rayon du cercle, les trois suivants sont la longueur des côtés du triangle. Par exemple :

```
./tp4 1 2 3 4
```

le rayon est 1 et les côtés son 2, 3, et 4.

Les longueurs des côtés ne peuvent pas avoir n'importe quelle valeur ; les valeurs doivent respecter l'inégalité triangulaire. Si les valeurs ne la respectent pas, le programme sort une erreur.

2.1 État de la pile au début d'une méthode

Lorsqu'une méthode **`M`** d'une classe **`C`** est appelé, 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)	addresse de l'objet de type C
4(%ebp)	ancienne valeur de %eip
(%ebp)	ancienne valeur de %ebp

Pour cet exercice, vous devrez vous servir des instructions en `floating point` vue au TP2 pour calculer l'aire, le périmètre et la hauteur. Voici un tableau décrivant les instructions utiles pour ce TP.

Instruction	**Description**
flds **addr**	Ajoute au-dessus de la pile l'entier à l'adresse mémoire **addr**. (st[1] prend la valeur de st[0] et st[0] devient la nouvelle valeur chargée de la mémoire.
fldpi	Ajoute au-dessus de la pile la valeur de π (3,1415…)
fstp **addr**	Retire l'élément st[0] pour le mettre en mémoire principale à l'adresse **addr.** st[1] devient st[0]
faddp	st[0] est additionné à st[1] et le résultat remplace st[0]. st[1] est libéré.
fsubp	st[0] est soustrait de st[1] et le résultat remplace st[0]. st[1] est libéré.
fsubrp	st[1] est soustrait de st[0] et le résultat remplace st[0]. st[1] est libéré.
fmulp	st[0] est multiplié avec st[1] et le résultat remplace st[0]. st[1] est libéré.
fdivp	st[1] est divisé par st[0] et le résultat remplace st[0]. st[1] est libéré.
fdivrp	st[0] est divisé par st[1] et le résultat remplace st[0]. st[1] est libéré.
fsqrt	st[0] = sqrt(st[0])

IMPORTANT : Veuillez écrire un court et bref commentaire devant chaque ligne d'assembleur m'expliquant votre raisonnement. Remettez les fichiers .s sans modifier leur nom.

2.2 Débogage

Vous pouvez déboguer votre programme avec gdb. Si vous avez une erreur de segmentation, gdb vous indiquera à quelle ligne se produit celle-ci et vous pourrez alors observer le contexte (valeurs des registres, des variables, de la mémoire, de la pile) et déterminer plus facilement la cause de l'erreur.

Une vidéo référence avec les commandes de gdb se trouve ici :

https://www.youtube.com/watch?v=wIuZajISL-E