

# INF1600 — TP4

## Programmation en assembleur et C++

Giovanni Beltrame `giovanni.beltrame@polymtl.ca`  
Carlo Pincioli `carlo.pincioli@polymtl.ca`

Polytechnique Montréal  
Année académique 2015/2016

# Remise

Voici les détails concernant la remise de ce travail pratique :

- **Méthode** : sur Moodle (une seule remise par groupe).
- **Échéance** : avant 12h00; le 28 mars 2017 pour la section B1, le 4 avril 2017 pour la section B2;
- **Format** : un seul fichier zip, dont le nom sera `<matricule1>-<matricule2>.zip`. Exemple : `0123456-9876543.zip`. L'archive doit contenir les fichiers `circle_perimeter.s`, `circle_area.s`, `triangle_perimeter.s`, `triangle_area.s` et `triangle_height.s`.
- **Langue écrite** : français.
- **Distribution** : les deux membres de l'équipe recevront la même note.

# Barème

Contenu	Points du cours
<code>circle_perimeter.s</code>	1
<code>circle_area.s</code>	1
<code>triangle_perimeter.s</code>	2
<code>triangle_area.s</code>	3
<code>triangle_height.s</code>	1
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

# Travail demandé

Vous êtes en charge de coder en assembleur cinq méthodes :

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

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

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

## 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.c` : programme de test qui utilise les fonctions de référence 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 compléter les fichiers `*.s` et les remettre dans une archive zip. Votre code doit passer chaque tests dans `tp4.c`.

## Compilation et testing

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

```
$ make
```

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, dans la commande :

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

le rayon est 1 et les côtés sont 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 le respectent pas, le programme sort une erreur.

## 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 <code>x</code>
8(%ebp)	adresse de l'objet de type <code>C</code>
4(%ebp)	ancienne valeur de <code>%eip</code>
(%ebp)	ancienne valeur de <code>%ebp</code>

## 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
<code>fld x</code>	Ajoute au dessus de la pile la valeur à l'adresse mémoire <code>x</code> ; <code>st[1]</code> prend la valeur de <code>st[0]</code> et <code>st[0]</code> devient cette nouvelle valeur chargée de la mémoire.
<code>fldpi x</code>	Ajoute au dessus de la pile la valeur de $\pi$ (3,1415...).
<code>fstp x</code>	Retire l'élément <code>st[0]</code> pour le mettre en mémoire principale à l'adresse <code>x</code> . <code>st[1]</code> devient <code>st[0]</code> .
<code>faddp</code>	<code>st[0]</code> est additionné à <code>st[1]</code> et le resultat remplace ces deux éléments.
<code>fsubp</code>	<code>st[1]</code> est soustrait de <code>st[0]</code> et le resultat remplace ces deux éléments.
<code>fsubrp</code>	<code>st[0]</code> est soustrait de <code>st[1]</code> et le resultat remplace ces deux éléments.
<code>fmlp</code>	<code>st[0]</code> est multiplié avec <code>st[1]</code> et le resultat remplace ces deux éléments.
<code>fdivp</code>	<code>st[0]</code> est divisé par <code>st[1]</code> et le resultat remplace ces deux éléments.
<code>fdivrp</code>	<code>st[1]</code> est divisé par <code>st[0]</code> et le resultat remplace ces deux éléments.
<code>fsqrt</code>	la valeur dans <code>st[0]</code> est remplacée par sa racine carrée

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