Développement pour mobiles
Programmation bas niveau avec Smali

M1 informatique

1. Écrivez un programme qui affiche `Hello World!` à l'écran.
2. Écrivez un programme qui calcule et affiche la valeur de l'expression $(7 + 1) \times (8 - 3)$.
3. Récupérez le fichier `Hello.apk` sur Moodle. En utilisant `Apktool` ou `APK Studio`, modifiez ce programme afin qu'il affiche le produit plutôt que la somme des deux entiers choisis au hasard ; modifiez également les couleurs d'affichage des deux `TextView`. Fabriquez un nouveau fichier `Hello.apk` à partir de ce programme modifié.
4. Écrivez un programme qui affiche 20 fois `Hello World!` au moyen d'une boucle.
5. Écrivez un programme qui affiche tous les multiples de 7 compris entre 1 et 500.
6. Écrivez un programme qui calcule la valeur de 10! au moyen d'une boucle et l'affiche.
7. Écrivez un programme qui calcule la valeur de 10! au moyen d'une fonction récursive et l'affiche.
8. Écrivez un programme qui :
   — construit un tableau dont la taille est $n+1$, où n est le nombre d'arguments fournis sur la ligne de commandes,
   — remplit ce tableau de la façon suivante : pour tout entier i compris entre 0 et n, la case d'indice i contient $i!$,
   — affiche le contenu de ce tableau à l'écran.
   Le programme devra comporter une méthode calculant $n!$ pour tout entier n fourni en paramètre et une méthode affichant le contenu du tableau fourni en paramètre.
9. Écrivez un programme qui convertit au format *heures*: *minutes*: *secondes* une durée en secondes (une valeur entière) fournie sur la ligne de commandes. Par exemple :

   ```
   $ adb shell dalvikvm -cp /data/local/Convertir.zip Convertir 38313
   38313s = 10h38m33s
   ```

10. Écrivez un programme qui calcule et affiche à l'écran le volume d'une sphère dont le rayon est fourni sur la ligne de commandes. Rappel : le volume V d'une sphère de rayon r est donné par la formule $V = \frac{4}{3}\pi r^3$.
11. Écrivez un programme qui permet de coder un message en clair et de décoder un message crypté en utilisant le chiffrement de César. La clé de chiffrement (le décalage) ainsi que le message à coder ou décoder doivent être fournis sur la ligne de commandes. L'option `c` (resp. `d`) sur la ligne de commandes indique que l'on souhaite crypter (resp. décrypter).

    Par exemple, pour crypter le message `programmation bas niveau` avec un décalage de 3 on écrira :

    ```
    $ adb shell dalvikvm -cp /data/local/Cesar.zip Cesar c 3 "programmation bas niveau"
    ```

    et pour décoder le message `surjudppdwlrq edv qlyhdx` avec un décalage de 3 on écrira :

    ```
    $ adb shell dalvikvm -cp /data/local/Cesar.zip Cesar d 3 "surjudppdwlrq edv qlyhdx"
    ```

Indications

Toutes les instructions disponibles sont présentées dans le tableau à l'adresse

https://source.android.com/devices/tech/dalvik/dalvik-bytecode

Quelques-unes sont rappelées ci-dessous.

— Quelques instructions de base :

```
# un commentaire
:ici   # une étiquette
const-string v0, "toto" # écrit l'adresse de la chaîne "toto" dans v0
const v1, 3 # écrit la constante 3 dans v1
const v2, 7 # écrit la constante 7 dans v2
move v1, v2 # recopie le contenu de v2 dans v1
add-int v3, v1, v2 # écrit le résultat de v1+v2 dans v3
sub-int v3, v1, v2 # écrit le résultat de v1-v2 dans v3
mul-int v3, v1, v2 # écrit le résultat de v1*v2 dans v3
rem-int v3, v1, v2 # écrit le résultat de v1%v2 dans v3
goto :ici # aller à l'étiquette :ici
if-eqz v0, :ici # si v0 = 0 alors aller à :ici sinon instruction suivante
if-lez v0, :ici # si v0 <= 0 alors aller à :ici sinon instruction suivante
if-ltz v0, :ici # si v0 < 0 alors aller à :ici sinon instruction suivante
if-eq v0, v1, :ici # si v0 = v1 alors aller à :ici sinon instruction suivante
if-le v0, v1, :ici # si v0 <= v1 alors aller à :ici sinon instruction suivante
if-lt v0, v1, :ici # si v0 < v1 alors aller à :ici sinon instruction suivante
```

— Affichage de la chaîne "toto" :

```
sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
const-string v1, "toto"
invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(Ljava/lang/String;)V
```

— Exécution d'une méthode statique et récupération de son résultat :

```
invoke-static {v0}, LFacto;->facto-ite(I)I
move-result v1
invoke-static {v2,v3}, LFacto;->facto-tab(II)[I
move-result-object v4
```

— Terminer l'exécution d'une méthode :

```
return-void
return v0 # si résultat de type primitif (par exemple int)
return-object v0 # si résultat de type objet ou tableau
```

— Lors d'un appel à une méthode, les valeurs des paramètres sont placées dans les derniers registres de la méthode. Exemple :

```
.method public static facto-ite(I)I
    .registers 5
    # 5 registres = v0,v1,v2,v3,v4
    # p0 = v4 = dernier registre = valeur de l'unique paramètre de facto-ite
    ...
.end method
```

— Récupération des arguments sur la ligne de commandes :

```
.method public static main([Ljava/lang/String;)V
  .registers 3

  # L'adresse du tableau contenant les arguments de la ligne de commandes
  # se trouve dans le dernier registre = v2 = p0 ici

  # Nombre d'arguments sur la ligne de commandes = taille du tableau
  # Pour mettre la taille du tableau dans le registre v0 :
  array-length v0, v2

  # Mettre le premier argument de la ligne de commandes dans le registre v1 :
  const v0, 0
  aget-object v1, v2, v0  # v1 = v2[v0] = v2[0]
  ...
  return-void
.end method
```

— Conversion d'une chaîne de caractères en une valeur entière : utiliser la méthode

```
static int parseInt(String)
```

de la classe `java/lang/Integer`. Si la conversion n'est pas possible, elle génère une exception.

— Opérations sur un tableau d'entiers :

```
const v0, 10
new-array v1, v0, [I   # création d'un tableau d'entiers de 10 cases
const v2, 0
aput v0, v1, v2  # v1[v2] = v0 c'est à dire v1[0] = 10
aget v3, v1, v2  # v3 = v1[v2] c'est à dire v3 = v1[0] = 10
```