PROJET C : ENONCE

Vous trouverez dans cet énoncé deux parties différentes à traiter et une partie Bonus :

Vous trouverez tout le basecode et les corrections des exercices de cette semaine à l'adresse suivante :

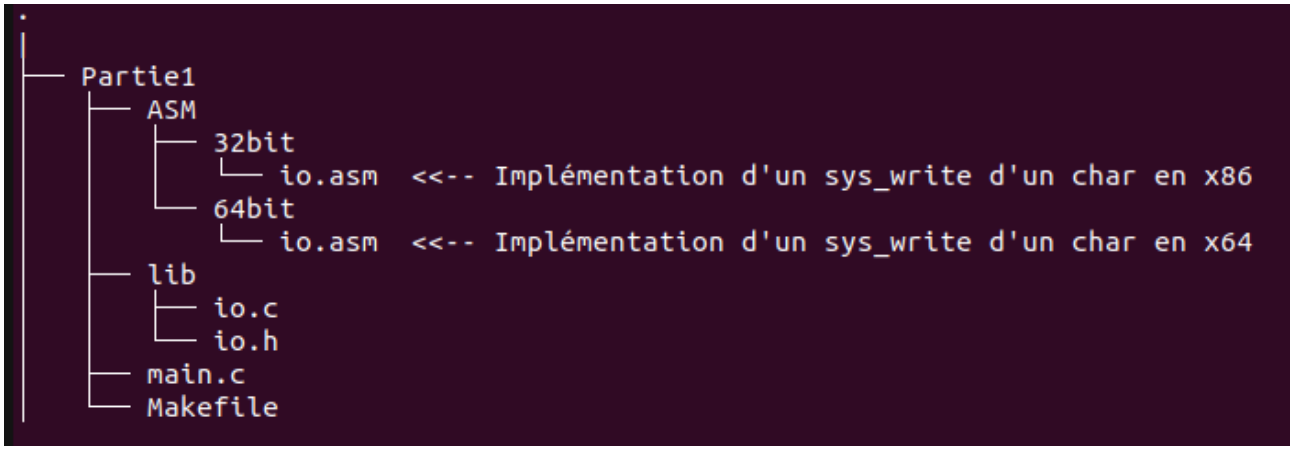
https://github.com/charlesDec/ProjetC

Partie1: Printf

Cet Exercice doit être traité dans Projet/Partie1/

Au terme de cette semaine, nous avons le basecode nécessaire pour afficher un caractère avec un appel système depuis le C. Le but de cet exercice est de développer en C à partir de ce code une fonction plus sophistiquée présentant des fonctionnalités équivalentes à **printf()** de la bibliothèque **stdio.h**. Vous disposez d'un basecode assez complet pour cet exercice ; il vous suffit pour la majorité des questions de compléter la fonction correspondante dans le fichier **Projet/Partie1/lib/io.c**

L'architecture de la Partie1 est la suivante :



Des morceaux de code de test des différentes fonctions à implémenter sont présentes dans le **main.c.** Pour les utiliser il vous sufit de compiler avec les bonnes options de compilation (présentes dans le **Makefile**).

Exemple : Pour compiler le projet pour tester le **TODO1** vous pouvez utiliser **make TODO1** ou **make TODO1_32** pour la version **32bit**.

TODO1 :

Compléter la fonction **printChar(char)** dans **Projet/Partie1/lib/io.c ,** qui imprime un **char** passé en paramètre.

Output attendu :

```
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ make TODO1
nasm -f elf64 ASM/64bit/io.asm -o io.o
gcc -D TODO1 -g main.c lib/io.c io.o -o a.out
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ ./a.out
C
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$
```

TODO2 :

Compléter une fonction **printString(char *)** dans **Projet/Partie1/lib/io.c**, qui imprime une chaîne de caractères passée en paramètre. Rappel : les chaînes de caractères sont terminées par char(0) ;

Output attendu :

```
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ make TODO2
nasm -f elf64 ASM/64bit/io.asm -o io.o
gcc -D TODO2 -g main.c lib/io.c io.o -o a.out
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ ./a.out
Hello Word!
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$
```

TODO3 :

Compléter une fonction **printDigit(int)** dans **Projet/Partie1/lib/io.c**, qui imprime le caractère correspondant au digit passé en paramètre. Attention : le caractère 5 par exemple n'a pas le code ASCII 5. Si votre entier n'est pas compris entre 0 et 9 ce n'est pas un digit, vous n'afficherez simplement pas les entiers qui ne sont pas des digits.

Output attendu :

```
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ make TODO3
nasm -f elf64 ASM/64bit/io.asm -o io.o
gcc -D TODO3 -g main.c lib/io.c io.o -o a.out
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ ./a.out
0123456789
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$
```

TODO 4 :

Compléter une fonction **printInteger(int)** dans **Projet/Partie1/lib/io.c,** qui imprime une chaîne de caractères correspondant à l'entier passé en paramètre. Exemple : printInteger(12) doit afficher la chaîne de caractères ‘‘12’’

Output attendu :

```
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ make TODO4
nasm -f elf64 ASM/64bit/io.asm -o io.o
gcc -D TODO4 -g main.c lib/io.c io.o -o a.out
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ ./a.out
123456
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$
```

TODO 5 :

Le fonction **printf()** original est une fonction variadique, c'est à dire qu'elle possède un nombre d'arguments supérieur ou égal à 1. Renseignez vous sur les fonctions variadiques (par exemple : https://en.cppreference.com/w/c/variadic). Vous allez malheureusement être obligé d'utiliser au moins une bibliothèque, **stdarg.h**

Tentez maintenant d'implémenter une fonction **newPrintf(char *, ...)** dans **Projet/Partie1/lib/io.c**, qui prend une chaîne de caractères en premier paramètre représentant la structure à afficher, puis le nombre d'arguments attendus pour remplir le template. On se concentre dans cet exercice uniquement sur les types suivants :

integer :%d

char : %c

char * :%s

Output attendu :

```
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ make TODO5
nasm -f elf64 ASM/64bit/io.asm -o io.o
gcc -D TODO5 -g main.c lib/io.c io.o -o a.out
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$ ./a.out
L'entier vaut : 111, le chractère est : A, et le string est : hello
marvin@marvinPC:~/Bureau/ProjetC_CORREC/Projet/Partie1$
```

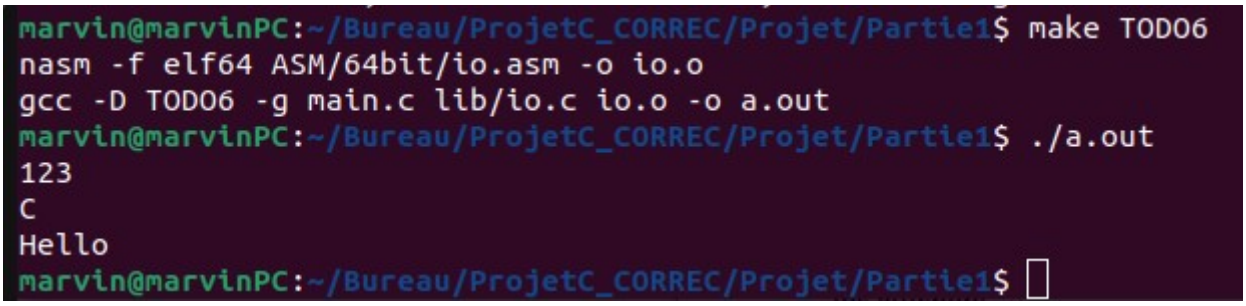
QUESTION :

(répondez dans votre tête) Comprenez vous maintenant l'implémentation de la fonction printf() ?

TODO 6 :

Interessez vous à la selection **_Generic** (par exemple https://en.cppreference.com/w/c/language/generic). Créez ensuite une fonction **printG()** dans **Projet/Partie1/lib/io.h** qui accepte tous les types de données présentées avant dans le document et les affiche.

Output attendu :



POUR ALLER PLUS LOIN :

Dans la section pour aller plus loin, tentez au maximum de rajouter des commentaires dans les différents documents que vous modifirez pour indiquer à quel point appartiennent les portions de code.

Exemple :

Si je traite le point **FURTHER 7** je peux l'indiquer dans du code C en l'écrivant entre deux **/*FURTHER 7*/** par exemple.

Si vous n'avez pas réussi un point mais que vous souhaitez communiquer sur le travail de recherche effectué n'hésitez pas à laisser un fichier **Projet/Partie1/ReadMe**

FURTHER 7 :

Rajoutez maintenant les types manquants du langage C à votre fonction **newPrintf(char *, ...)**.

FURTHER 8 :

DUR : Auriez-vous pu réaliser une fonction variadique sans utiliser de librairie ?

FURTHER 9 :

DUR : Tentez de porter votre code vers un autre OS ou une autre archi de façon à pouvoir choisir au moment de la compilation quelle architecture et quel OS vous visez.

Partie 2 :

Cet Exercice doit être traité dans Projet/Partie2/

Cette deuxième partie est plus simple et vous fait travailler le C à plus haut niveau dans un premier temps.

TODO :

En utilisant les bibliothèques de votre choix, implémentez une petite calculatrice.

Votre programme sera composé de deux parties :

→ Un interpréteur de commandes qui lit les commandes utilisateurs. Elles devront être de la forme :

```
expression1 operator expression2
```

Où **expression1**/**2** seront des valeurs entières, flottantes, ou la chaîne de caractères **res** (le resultat précédent). Et **operator** pourra être **+,-,*,/, %**
Le retour à la ligne force l'interprétation de la ligne. Pensez à afficher des messages d'erreurs si la ligne entrée n'est pas correcte ou si l'opération n'est pas permise (exemple : division par 0).

Le mot clé **flush** videra le contenu à l'écran et **quit** permettra de quitter le programme.

→ Un affichage de la forme suivante :

```
id>expression operator expression

res : resultat

id+1> expression operateur expresion

res : resultat
```

Afin de coller au terminal par défaut, votre affichage devra comporter 80 colonnes et 25 lignes. Si vous arrivez à la fin de cette zone le contenu le plus ancien devra s'éclipser par le haut (si c'est le comportement par défaut de votre terminal, ce ne sera peut être pas le cas quand vous chercherez à porter le code).

Au niveau du rendu je vous laisse libre d'organiser le répertoire **Projet/Partie2/** comme bon vous semble. Vous devrez cependant avoir au moins un **Projet/Partie2/Makefile** tel que :
make : compile votre code en un fichier de sortie nommé **Projet/Partie2/a.out**
make install : installe toutes les dépendances nécessaires au fonctionnement de votre code
make clean : supprime tous les fichiers non sources

Si vous souhaitez m'expliquer les fonctionnalités, votre avancée ou les problèmes rencontrés, n'hésitez pas à laisser un fichier **Projet/Partie2/ReadMe**

POUR ALLER PLUS LOIN :

- Ajouter des opérations supplémentaires et un nouveau formalisme pour l'interprétation des commandes (exemple ajout de parenthèses).

- Rajouter un nombre limité de variables pouvant stocker une valeur

- Dur : Ne plus utiliser de librairie. En réutilisant le code déjà présent dans la partie précédente pour effectuer un sys_write, trouvez comment effectuer un sys_read vous permettant de récupérer les entrées clavier. **Attention** : le comportement par défaut du terminal peut être un obstacle à une utilisation plus intuitive d'un sys_read dans certains cas. Interessez vous à la sortie du mode canonical du terminal pour résoudre ce problème.

- Attention c'est dur : Comme pour le bonus de la partie précédente tentez votre chance en portant votre code vers une autre cible (l'archi présentée dans la Partie Bonus pourrait être un bon cas pratique).

Partie Bonus :

Cet Exercice doit être traité dans Projet/PartieBonus

Le but de cet exercice est de réaliser un programme dans le BootSector (le premier secteur chargé en mémoire) d'une machine. Afin de simplifier les opérations, nous utiliserons une machine i386 virtualisée par QEMU. Vous êtes libre de développer l'application de votre choix (Jeu, RickRollAscii, tout ce qui vous passe par la tête...). Vous trouverez dans le Basecode le code minimum nécessaire pour compiler et éxécuter la VM. C'est un bon exercice pour travailler l'ASM.

Vous n'avez pas dans cette configuration de kernel pour vous permettre de faire des syscall, et sans OS il est compliqué de parler au matériel. Heureusement pour vous (en real mode) le BIOS va vous aider dans votre tâche. La plupart des opérations matérielles peuvent se faire par l'intermédiaire des interruptions du BIOS (https://en.wikipedia.org/wiki/BIOS_interrupt_call).

Modalités de Rendu

Votre rendu s'effectura sur Moodle : https://learning.esiea.fr/course/view.php?id=519

Vous pouvez constituer des groupes de deux ou rendre seul. Dans l'hypothèse où vous seriez un nombre impair, vous pouvez constituer un seul groupe de 3. Les groupes devront être indiqués dans le document suivant avant vendredi 14 octobre 2022 à 23h59 :

Vous remplirer aussi le fichier **Projet/groupe.csv**

Vous devez me rendre une archive conenant le répertoire **Projet/** au format **.zip**.

Dates de Rendu

Vendredi 14 octobre 2022 avant 23h59 : conception des équipes sur le lien :https://etesiea-my.sharepoint.com/:x:/g/personal/charles_decellieres_ext_esiea_fr/Eb8MCTfoNohGg3hxEDKOeyQBkKpgaUnYiTvXHmUH8Y-yMA?e=kwHuyp

Vendredi 16 décembre 2022 avant 23h59 : Rendu de l'archive sur moodle

Contact

Si vous avez des questions, vous pouvez me contacter par Discord à **Charles Decellieres #4324**, ou par mon mail ESIEA charles.decellieres@ext.esiea.fr (moins réactif).

ANNEXE COMMANDES UTILES

Compilation

Option

pour pouvoir débugger l'exécutble :

```
gcc -g
```

pour rajouter un FLAG a la compilation

```
gcc -D FLAG
```

En x64 :

fichier C → elf linké (executable) :

```
gcc main.c -o a.out
```

fichier C → assembleur (.s pas .asm) :

```
gcc -S main.c -o main.s
```

fichier C→ elf non linké (objet) :

```
gcc -c main.c -o main.o
```

fichier ASM → elf non linké (objet) :

```
nasm -f elf64 main.asm -o main.o
```

fichier ASM → elf linké (executable) :

```
nasm -f elf64 main.asm -o main.o
ld -m elf_x86_64 main.o -o a.out
```

En x32 :

fichier C → elf linké (executable) :

```
gcc -m32 main.c -o a.out
```

fichier C → assembleur (.s pas .asm) :

```
gcc -m32 -S main.c -o main.s
```

fichier C→ elf non linké (objet) :

```
gcc -m32 -c main.c -o main.o
```

fichier ASM → elf non linké (objet) :

```
nasm -f elf main.asm -o main.o
```

fichier ASM → elf linké (executable) :

```
nasm -f elf main.asm -o main.o
ld -m elf_i386 main.o -o a.out
```