Piscine - C - Tek1

Sujet Jour 11

Responsable Astek astek_resp@epitech.eu

Table des matières

Consignes

- Le sujet peut changer jusqu'à une heure avant le rendu.
- Vos exercices doivent être à la norme.
- Travaillez en local !
  C'est-à-dire que pour chaque exercice vous devez le compiler sur votre compte linux puis, une fois qu'il fonctionne, le copier sur votre compte AFS.
  Ceci dans le simple but de ne pas surcharger les serveurs car vous êtes nombreux.

Indices `Faites-vous un script shell pour copier vos fichiers sur l'AFS`

- Nous compilons avec votre lib et vos includes.
- Rendu :
  `/afs/epitech.net/users/group/login/rendu/piscine/Jour_11`

`Attention aux droits de vos fichiers et de vos répertoires`

Exercice 1 - Makefile

- Écrire le `Makefile` qui compile votre `libmy`.
- Le `Makefile` doit copier la `library` dans :
  `/afs/epitech.net/users/group/login/rendu/lib`
- Le `Makefile` devra copier le `my.h` dans :
  `/afs/epitech.net/users/group/login/rendu/include`
- Le `Makefile` devra implémenter la règle `clean`.
- Votre `Makefile` se trouvera, ainsi que tous les `.c` et `.h` nécessaires, dans :
  `/afs/epitech.net/users/group/login/rendu/lib/my/`

`A partir de cet exercice vous devez TOUJOURS avoir votre lib dans rendu/lib/my/ et vos includes dans rendu/include`

Exercice 2 - do-op

- Écrire un programme qui s'appelle `do-op`.
- Le programme devra être lancé avec trois arguments :
  `do-op valeur1 operateur valeur2`
- Exemple :

```
$>./do-op 42 "+" 21
63
$>
```

- Le caractère `operateur` correspondra à la fonction appropriée dans un tableau de pointeurs sur fonction.
- Ce répertoire comportera un `Makefile` avec une règle `all` et une règle `clean`.
- Dans le cas d'une expression fausse `./do-op foo devide bar` le programme affiche `0`.
- Si le nombre d'arguments n'est pas correct `do-op` n'affiche rien.
- Exemple de tests de la moulinette :

```
$> make clean
$> make
$> ./do-op
$> ./do-op 1 + 1
2
$> ./do-op 42amis - ---20toto12
62
$> ./do-op 1 p 1
0
$> ./do-op 1 +toto 1
2
$> ./do-op 1 + toto3
1
$>
$> ./do-op toto3 + 4
4
$> ./do-op foo plus bar
0
$> ./do-op 25 / 0
Stop : division by zero
$> ./do-op 25 % 0
Stop : modulo by zero
```

- Tous les fichiers relatifs à votre programme seront dans :
`/afs/epitech.net/users/group/login/rendu/piscine/Jour_11/do-op/`

`Attention à la division ou le modulo par 0`

Exercice 3 - my_sort_wordtab

- Écrire la fonction `my_sort_wordtab` qui trie par ordre `ascii` les mots obtenus grâce à `my_str_to_wordtab`
- Le tri s'effectuera en échangeant les pointeurs du tableau.
- Elle devra être prototypée de la façon suivante :

```
1   int my_sort_wordtab(char **tab);
```

- La moulinette utilisera votre `lib` (donc votre `my_putchar`, etc...)
- La fonction devra toujours retourner `0`
- Rendu :

`/afs/epitech.net/users/group/login/rendu/piscine/Jour_11/ex_03/my_sort_wordtab.c`

Exercice 4 - my_advanced_sort_wordtab

- Écrire la fonction `my_advanced_sort_wordtab` qui trie en fonction du retour de la fonction passee en paramter les mots obtenus grâce à `my_str_to_wordtab`
- Le tri s'effectuera en échangeant les pointeurs du tableau.
- Elle devra être prototypée de la façon suivante :

```
1   int my_advanced_sort_wordtab(char **tab, int(*cmp)(char *, char *));
```

- La moulinette utilisera votre `lib` (donc votre `my_putchar`, etc...)
- La fonction devra toujours retourner `0`
- Rendu :

`Jour_11/ex_04/my_advanced_sort_wordtab.c`

Indices un appel à my_advanced_sort_wordtab() avec pour second parametre my_strcmp donnera le même resultat que my_sort_wordtab()