UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR | COLLÈGE STEE SCIENCES ET TECHNOLOGIES POUR L'ÉNERGIE ET L'ENVIRONNEMENT

Techniques de programmation - TP nº 1 - Bases du langage C

Samson Pierre <samson.pierre@univ-pau.fr>

03/10/2021

Ce TP a pour objectif de vous familiariser avec les bases du langage C. Pour chaque exercice, sauf indication contraire, aucune fonction n'est autorisée et les mêmes options de compilation que celles du cours doivent être utilisées.

Exercice nº 1

Créez le programme `hello-world.c` vu en cours. Compilez ce code puis lancez votre programme `hello-world.out`.

Voici les fonctions autorisées pour cet exercice :

```
int printf(const char *format, ...);
```

Exercice nº 2

Reprenez le code de l'exercice précédent.

Compilez ce code avec l'option `-E` afin de stopper la compilation après l'étape de précompilation. Observez le contenu du fichier généré dans un éditeur de texte. Remarquez que seules les lignes commençant par un `#` ont été transformées. Grâce à la commande `file` vérifiez le type du fichier généré.

Compilez ce code avec l'option `-S` afin de stopper la compilation après l'étape de compilation. Observez le contenu du fichier généré dans un éditeur de texte. Remarquez que le langage utilisé dans ce fichier est l'assembleur. Grâce à la commande `file` vérifiez le type du fichier généré.

Compilez ce code avec l'option `-c` afin de stopper la compilation après l'étape d'assemblage. Remarquez que le contenu du fichier généré est à présent difficilement consultable dans un éditeur de texte. C'est normal puisque c'est un fichier binaire destiné à la machine. Grâce à la commande `file` vérifiez le type du fichier généré. Remarquez qu'il est impossible de lancer ce programme car l'édition des liens n'est pas faite. Grâce à la commande `ldd` vérifiez qu'il est impossible de voir à quelles bibliothèques le programme est lié. C'est normal puisque l'édition des liens n'est pas faite.

Compilez ce code sans ces trois options afin de laisser la compilation atteindre la dernière étape d'édition des liens. Grâce à la commande `file` vérifiez le type du fichier généré. Grâce à la commande `ldd` vérifiez qu'il est possible de voir à quelles bibliothèques le programme est lié.

Voici les fonctions autorisées pour cet exercice :

```
int printf(const char *format, ...);
```

Exercice nº 3

Créez le programme `types.c` qui affiche pour chaque type, sa taille en octets et sa plage de valeurs. Utilisez dans votre code l'opérateur `sizeof` pour obtenir la taille en octets des types. Utilisez dans votre code les macros vues en cours pour obtenir la plage de valeurs des types.

Voici le résultat attendu :

```
$ ./types.out
char: 1 byte(s), -128 to 127
signed char: 1 byte(s), -128 to 127
unsigned char: 1 byte(s), 0 to 255
short: 2 byte(s), -32768 to 32767
unsigned short: 2 byte(s), 0 to 65535
int: 4 byte(s), -2147483648 to 2147483647
unsigned int: 4 byte(s), 0 to 4294967295
long: 8 byte(s), -9223372036854775808 to 9223372036854775807
unsigned long: 8 byte(s), 0 to 18446744073709551615
float: 4 byte(s), 1.175494e-38 to 3.402823e+38
double: 8 byte(s), 2.225074e-308 to 1.797693e+308
long double: 16 byte(s), 3.362103e-4932 to 1.189731e+4932
$
```

Voici les fonctions autorisées pour cet exercice :

```
int printf(const char *format, ...);
```

Exercice n° 4

Créez le programme `overflow.c` dans lequel vous affectez à une variable de type `short` un entier en dehors de la plage de valeurs du type `short`. Essayez de compiler votre programme. Remarquez que le compilateur affiche un message d'erreur.

Exercice n° 5

Créez le programme `mark.c` qui lit au clavier la note d'un étudiant. Tant que cette valeur n'est pas un entier compris entre 0 et 20, le programme doit redemander la saisie. Utilisez l'instruction d'itération `while` pour que le programme redemande la saisie. Utilisez les instructions de sélection `if` et `else` imbriquées pour que le programme affiche la mention `"failing"` pour une note entre 0 et 10, `"satisfactory"` pour une note entre 10 et 12, `"good"` pour une note entre 12 et 16 et `"excellent"` pour une note entre 16 et 20.

Voici le résultat attendu :

```
$ ./mark.out
mark: -1
mark: a
mark: 21
mark: 11
satisfactory
$
```

Voici les fonctions autorisées pour cet exercice :

```
int getchar(void);
int printf(const char *format, ...);
int scanf(const char *format, ...);
```

Exercice n° 6

Créez un programme similaire à celui de l'exercice précédent. Utilisez les instructions d'itération `do` et `while` à la place du `while`. Utilisez seulement l'instruction de sélection `if` (donc pas de `else`) non imbriquée et compliquez les conditions avec l'opérateur `&&` (ET logique) lorsque c'est nécessaire.

Voici les fonctions autorisées pour cet exercice :

```
int getchar(void);
int printf(const char *format, ...);
int scanf(const char *format, ...);
```

Exercice n° 7

Créez un programme `counter.c` qui lit au clavier un entier positif ou nul `max`. Tant que cette valeur n'est pas un entier positif ou nul, le programme doit redemander la saisie. Utilisez l'instruction d'itération `for` pour afficher les entiers de 0 à `max`.

Voici le résultat attendu :

```
$ ./counter.out
max: -1
max: a
max: 3
0
1
2
3
$
```

Voici les fonctions autorisées pour cet exercice :

```
int getchar(void);
int printf(const char *format, ...);
int scanf(const char *format, ...);
```

Exercice n° 8

Créez un programme `banknotes.c` qui, dans une boucle infinie, lit au clavier la valeur d'un billet puis compte et affiche le nombre de billets de chaque valeur. Utilisez l'instruction de sélection `switch` et l'instruction étiquetée `case` pour compter le nombre de billets de chaque valeur. Les billets acceptés sont ceux de 5, 10, 20, 50, 100, 200 et 500 euros (ainsi que -1 pour afficher le message `"exit"` dans le flux de sortie standard et quitter le programme avec l'instruction de saut `return`). Utilisez l'instruction étiquetée `default` dans le `switch` pour afficher un message d'erreur dans le flux d'erreur standard si la valeur n'est pas celle d'un billet accepté.

Voici le résultat attendu :

```
$ ./banknotes.out
banknote: 2
invalid banknote value
0 bankote(s) of 5 euros
0 bankote(s) of 10 euros
0 bankote(s) of 20 euros
0 bankote(s) of 50 euros
0 bankote(s) of 100 euros
0 bankote(s) of 200 euros
0 bankote(s) of 500 euros
banknote: 5
1 bankote(s) of 5 euros
0 bankote(s) of 10 euros
0 bankote(s) of 20 euros
0 bankote(s) of 50 euros
0 bankote(s) of 100 euros
0 bankote(s) of 200 euros
0 bankote(s) of 500 euros
banknote: 10
1 bankote(s) of 5 euros
1 bankote(s) of 10 euros
0 bankote(s) of 20 euros
0 bankote(s) of 50 euros
0 bankote(s) of 100 euros
0 bankote(s) of 200 euros
0 bankote(s) of 500 euros
banknote: -1
exit
$
```

Voici les fonctions autorisées pour cet exercice :

```
int fprintf(FILE *stream, const char *format, ...);
int getchar(void);
int printf(const char *format, ...);
int scanf(const char *format, ...);
```

Astuces

Pour les exercices de ce TP, vous devez utiliser des spécificateurs de conversion (`%d`, ...) pour le paramètre `format` de la fonction `printf`. Ces spécificateurs de conversion seront présentés dans le CM n° 4. Vous pouvez trouver la liste des spécificateurs de conversion dans le manuel de la fonction `printf` :

```
$ man 3 printf
$
```

À partir de l'exercice n° 5, vous devez utiliser les fonctions `getchar` et `scanf`. Ces fonctions seront présentées dans le CM n° 4. La fonction `scanf` lit dans le flux d'entrée standard (c'est-à-dire une saisie au clavier). Elle affecte les valeurs lues aux éléments passés en paramètres. Elle a un nombre de paramètres variable. Elle est déclarée dans le fichier d'en-tête `stdio.h`. Elle a pour paramètres `format` (le format des éléments à lire) et `...` (les valeurs à convertir). Si succès, elle a pour valeur de retour le nombre d'éléments qui ont été affectés à des valeurs. Si échec, elle a pour valeur de retour `EOF`. Le problème est que `scanf` ne consomme pas forcément tous les caractères du flux d'entrée standard. Par conséquent, les appels suivants à `scanf` pourraient ne pas être bloquants. La raison est que des caractères restent dans le flux d'entrée standard. La solution est de vider le flux d'entrée standard grâce à la fonction `getchar`. Exemple :

```
scanf("%d", &my_int);
while(getchar() != '\n');
```

Pour l'exercice n° 8, lorsque vous souhaitez écrire dans le flux d'erreur standard, vous devez utiliser la fonction `fprintf`. Cette fonction sera présentée dans le CM n° 4. Lorsque vous souhaitez appeler cette fonction pour écrire dans le flux d'erreur standard, vous devez utiliser le pointeur `stderr` pour le paramètre `stream`.