


Mark as done


 [Description](#)

 [Submission](#)



 [Edit](#)

 [Submission view](#)

 [Previous submissions list](#)

 **Available from:** Monday, 18 November 2024, 12:00 AM

 **Due date:** Sunday, 8 December 2024, 11:59 PM

 **Required files:** run.sh, valgrind_report.sh, perf_compil.sh, strip.sh, ex_tableau_erreur.c, fibonacci.c ( [Download](#))

 **Maximum number of files:** 15

Type of work:  Individual work

Reduction by automatic evaluation: 1 **Free evaluations:** 10

Pour cette activité, vous aurez à écrire un ensemble de scripts bash destinés à servir de petits utilitaires lors d'une activité de développement. Dans aucun des exercices il ne vous est demandé d'écrire un **Makefile**, si vous devez utiliser **make** vous pouvez supposer que le **Makefile** est déjà écrit ou que les règles implicites suffisent.

1. Compilation et exécution

Dans cette partie, il vous est demandé d'écrire un script nommé **run.sh** qui, pour chaque argument **x** qui lui est donné sur la ligne de commande, devra :

- compiler **x** grâce à **make** si **x** possède une extension : dans ce cas **x** est de la forme **base.extension**, et **make** devra être utilisé pour produire l'exécutable de nom **base**. Si la compilation échoue, le script devra se terminer avec le code de retour **1** ;
- exécuter **x** (si **x** n'a pas d'extension) ou **base** (si **x** a une extension).

Pour cet exercice, vous ne devez produire aucun affichage sur la sortie standard autre que ceux correspondants au travail demandé.

2. Comparaison de performance



Dans cette partie, vous devez écrire un script nommé **perf_compil.sh** qui, étant donné un programme **x** dont le nom est fourni en premier argument, devra :

1. compiler **x** en utilisant **clang** :
 - directement, en respectant la variable d'environnement **CFLAGS**, si **x** contient l'extension **.c** ;
 - avec **make**, en supposant que les variables d'environnement **CC** et **CFLAGS** sont prises en compte, sinon.
2. exécuter l'exécutable résultant :
 - avec pour arguments, tous les arguments restants (ceux donnés au script après le premier) ;
 - en stockant son temps d'exécution.
3. recommencer les points 1 et 2 avec **gcc**
4. selon l'exécution la plus rapide, afficher l'un des messages suivants :
 - **clang est meilleur que gcc** ;
 - **gcc est meilleur que clang** ;
 - **clang et gcc sont similaires.**

Work state summary

Submissions: 37

Last submission:

 Submitted on Saturday, 30 November 2024, 1:51 AM ( [Download](#))

Reviewed on Saturday, 30 November 2024, 1:54 AM by Automatic grade

Grade: 64.58 / 100.00

[Details](#)

?

Ce script devra se terminer avec le code de retour 0 (si tout se passe bien) ou :

- 1 si on ne lui donne aucun argument ;
- 2 si l'une des compilations échoue.

Pour cet exercice, vous ne devez produire aucun affichage autre que ceux correspondants au travail demandé (et dans l'ordre demandé) sur la sortie standard.

3. Rapport valgrind

Il vous faut maintenant écrire un script nommé `valgrind_report.sh` qui exécute à l'aide de `valgrind` le programme dont le nom est donné en premier argument en lui fournissant comme arguments tous les arguments restants (après le premier). Ce script devra capturer les sorties de `valgrind` afin de les réafficher sur la sortie d'erreur standard sous la forme simplifiée suivante :

- les erreurs de lecture devront être indiquées sur une ligne sous la forme :
`Erreur en lecture dans <fonction> (<fichier>:<ligne>)`
- les erreurs d'écriture devront être indiquées sur une ligne sous la forme :
`Erreur en ecriture dans <fonction> (<fichier>:<ligne>)`
- les fuites mémoire devront être comptabilisées en faisant la somme des fuites définitives, indirectes, possibles, ainsi que des zones encore atteignables et supprimées. Le total devra être affiché sous la forme :
`Total des fuites : <nombre> octets`

A titre d'exemple, l'utilisation de ce script sur `ex_tableau_erreur` donne la sortie d'erreur :

```
Erreur en ecriture dans main (ex_tableau_erreur.c:17)
Total des fuites : 480 octets
```

La localisation utilisée pour chaque erreur correspond à la source de l'erreur (première localisation indiquée par `valgrind`), pas à la localisation de l'allocation correspondante.

Pour cet exercice, vous pouvez supposer que l'exécutable fourni est compilé avec `-g` et vous ne devez produire aucun affichage autre que ceux correspondants au travail demandé sur la sortie d'erreur.

4. Nettoyage

Pour terminer, écrivez un script, `strip.sh`, qui, pour chaque argument `x` qui lui est donné :

- si `x` est un répertoire, applique le traitement effectué par le script à tout les fichiers (non cachés) contenus dans `x` ;
- sinon :
 - si `x` est un fichier au format ELF (la commande `file` peut vous aider), lui applique la commande `strip` ;
 - sinon, le supprime.

Required files

`run.sh`

`valgrind_report.sh`

`perf_compil.sh`

`strip.sh`

`ex_tableau_erreur.c`

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[]) {
5      int **p, ok=1;
6      p = malloc(sizeof(int *)*10);
7      if (p == NULL)
8          exit(1);
9      for (int i=0; i<10 && ok; i++) {
10         p[i] = malloc(sizeof(int)*10);
11         ok = (p[i] != NULL);
12     }
13     if (ok) {
14         printf("Square matrix of size %d allocated\n", 10);
15         for (int i=0; i<10; i++)
16             for (int j=0; j<10; j++)
17                 p[i][j] = 0;
18     }
19
20
21
22
23
24
25
26
27
28
29

```

fibonacci.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  unsigned long fibonacci_rec(unsigned long n) {
5      if (n < 2)
6          return n;
7      else
8          return fibonacci_rec(n-1) + fibonacci_rec(n-2);
9  }
10
11 unsigned long fibonacci_lin(unsigned long n) {
12     long fib_prec = 0;
13     long fib_curr = 1;
14     for (int i=2; i<=n; i++) {
15         long fib_next = fib_prec + fib_curr;
16         fib_prec = fib_curr;
17         fib_curr = fib_next;
18     }
19     return fib_curr;
20 }
21
22 int main(int argc, char *argv[]) {
23     unsigned int n = 0;
24     if (argc > 1) {
25         n = (unsigned int) atoi(argv[1]);
26     }
27     #ifdef __GNUC__
28     #ifdef __clang__
29         printf("Fibonacci(%u): %lu\n", n, fibonacci_lin(n));
30     #else
31         printf("Fibonacci(%u): %lu\n", n, fibonacci_rec(n));
32     #endif
33     #else
34         fprintf(stderr, "Compilateur non géré\n");
35     #endif
36 }
37

```

[Webservice](#)
[VPL 4.2.4](#)

?

