DAEMONLAB

Carte au tresor

Ne cherchez pas le trésor !
Cherchez comment faire le programme qui cherche le trésor.

- DaemonLab -

L'objectif de la ruée Carte au trésor est d'apprendre à chercher dans un espace simple le plus court chemin.

INDEX

01 – Avant-propos

Votre travail doit être rendu via les dossiers **~/rue/tresor/** dans votre espace personnel.

Si vous faites erreur et que le dossier que vous utilisez pour votre rendu est différent, vous ne serez pas évalué faute d'avoir pu trouver votre travail.

Ce travail est a effectuer en binôme. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée de la même manière que la correction : de manière **automatique**. Prenez garde si vous pensez pouvoir passer au travers.

Votre rendu doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** fichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale (programme ou bibliothèque)

La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.

Votre programme doit respecter les Tables de la Norme dans leur intégralité. Vous êtes invité à les observer depuis **l'Infosphère**. Elles sont disponibles comme ressource de cette activité.

02 – Fonctions autorisées

La bibliothèque logicielle venant avec le C est vaste et disponible. La LibLapin, que vous utilisez dans vos projets multimédia, est également vaste… Cependant nous avons fait le choix de vous interdire leurs utilisation intégrales, afin de vous amener progressivement à reprogrammer vous même ses fonctionnalités les plus utiles.

L'utilisation d'une fonction interdite est assimilée à de la triche. La triche provoque l'arrêt de l'évaluation et la perte des médailles.

Vous n'avez le droit d'utiliser aucune fonction issue de la LibC ou de la LibLapin à l'exception de celles que nous vous autoriserons explicitement.

Pouvoir utiliser une fonction ne signifie pas nécessairement que celle-ci soit utile à votre cas.

Pour cette activité, issu de la LibC, vous n'avez le droit qu'à la liste suivante :

- open
- close
- read
- srand
- cos
- atan2
- write
- alloca
- atexit
- rand
- sin
- sqrt

Remarquez bien l'absence de **malloc** et de **free**. En effet ils sont **interdits** ! Issu de la LibLapin, vous avez le droit aux fonctions suivantes :

- bunny_start
- bunny_stop
- **bunny_malloc**
- **bunny_free**
- bunny_new_pixelarray
- bunny_delete_clipable
- bunny_blit
- bunny_display
- bunny_set_*_function
- bunny_set_*_response
- bunny_loop
- bunny_create_effect
- bunny_compute_effect
- bunny_sound_play
- bunny_sound_stop
- bunny_delete_sound
- bunny_open_configuration
- bunny_delete_configuration
- bunny_configuration_getf
- bunny_configuration_setf
- bunny_configuration_*
- bunny_set_memory_check
- bunny_release
- bunny_usleep

La suite sur la page d'après.

Pour utiliser **bunny_malloc**, vous pouvez soit programme directement avec, soit en utilisant le modèle de projet qui vous a été transmis, mettre 1 dans la variable de Makefile **BMALLOC**, qui transformera **malloc** en **bunny_malloc**.

Vous appellerez **bunny_set_memory_check** au début de votre fonction **main** de sorte à provoquer une vérification de vos allocations à la fermeture du programme.

bunny_malloc, par défaut, limitera votre consommation de RAM à 20Mo.

Pour information :
Une image en 1920*1080 fait environ 8Mo.

Une musique en 44kHz de 1 minute en stéréo fait environ 10Mo.

Vous devrez donc disposer d'une discipline de fer avec vos allocations... et probablement trouver des compromis.

L'utilisation de **bunny_malloc** parfois **cachera** des erreurs dans votre programme, et parfois en **révélera** : son principe d'allocation était différent de **malloc**, il sera parfois plus « fort » ou plus « faible ». Ne vous mentez pas à vous en disant « l'utilisation de **bunny_malloc** fait planter mon programme », ce n'est pas **bunny_malloc**, c'est *vous*.

N'hésitez pas à l'activer, à le désactiver (Et lorsqu'il est désactivé, à utiliser valgrind). Et n'oubliez pas que désormais, votre demande de RAM a de véritables chances d'échouer. Car exploiter aussi peu de RAM, cela va très vite…

Dans votre rendu, il ne devra pas y avoir la moindre trace de malloc.

03 – Méthode de construction

Il peut vous être demandé d'écrire des programmes ou des fonctions.

Dans le cas des programmes, il vous sera toujours demandé de fournir un **dossier** pour l'exercice le requérant. Un **Makefile** vous sera également demandé. Le **nom du programme** de sortie vous sera précisé à chaque fois. Un Makefile incorrect, un mauvais nom de programme, et votre correction n'aura pas lieu…

Dans le cadre des fonctions, il vous ai demandé de fournir le fichier dans votre **dossier de bibliothèque personnelle**, de sorte à ce que vous puissiez utiliser toutes les fonctions que vous avez déjà réalisé jusqu'ici. Pour rappel, le dossier de votre bibliothèque doit être placé à la racine de votre espace personnel et s'appeler **libstd/.**

N'oubliez pas d'entretenir avec soin votre dossier **libstd/** de sorte à ce qu'il soit toujours propre, respecte la norme et soit en état de compiler… sans quoi elle fera obstacle à la correction.

Votre compilation devra toujours comporter les options **-W**, **-Wall** et **-Werror**.

Dans le cadre de la programmation multimédia, le système de correction établira toujours la variable d'environnement **BMALLOC** à 1. Si vous utilisez le modèle de projet, cela provoquera l'utilisation de **bunny_malloc** dans votre bibliothèque personnelle comme dans votre projet rendu.

04 – Petite carte simple

Votre travail consistera à écrire un programme dont l'objectif est de dessiner le plus court chemin permettant d'aller d'un point à un autre.

Votre programme prendra en paramètre un fichier contenant une carte et devra afficher cette carte avec le chemin supplémentaire.

Le fichier peut-être une image ou un fichier texte. La définition du format image vous serea donnée dans la section « Grande Carte ». Si il s'agit d'un fichier texte, voici un exemple de contenu :

```
12,6
############
#..........#
#.#B##.#.#.#
#.####.#.#.#
#........A.#
############
```

Le fichier commence par définir la largeur de chacune de ses lignes avant de placer une virgule puis de définir la hauteur de ses lignes. Si il y a une erreur, qu'il s'agisse de syntaxe ou d'une taille qui est mauvaise, **vous devez le signaler sur la sortie d'erreur.** Si vous parvenez à réparer cette erreur, vous pouvez continuer, sinon quitter est acceptable.

Le symbole **#** indique un mur : un espace infranchissable. Le symbole **.** Indique une case ou il est possible d'aller. Le caractère **A** est le point de départ et le point **B** l'arrivée.

Vous devez dessiner le chemin en utilisant le caractère **X**. Voici ce qui devrait être dessiné par votre programme, par exemple.

```
############
#..XXXX....#
#.#B##X#.#.#
#.####X#.#.#
#.....XXXA.#
############
```

05 – Petite carte sans bornes

Vous allez maintenant étendre votre recherche à un type de carte spécifique. En voici un exemple :

```
12,6
###.####.###
#A.......B##
..###.###...
####...#####
###......###
###.####.###
```

Remarquez ici l'absence de mur sur les contours. Vous devez être en mesure de gérer ces trous comme étant des passerelles menant à l'autre bord. Ainsi la solution de cette carte est la suivante :

```
###.####.###
#A.......B##
XX###.###XXX
####...#####
###......###
###.####.###
```

06 – Petite carte à poids

Jusqu'ici, chaque pas avait la même valeur. Le symbole **.** jouant le rôle d'un emplacement librement parcourable. Dorénavant, les espace parcourables pourront également être des chiffres. Le chiffre 1 étant l'équivalent du symbole **.** .

Ces chiffres valent chacun leur représentation. Le chiffre 1 signifie qu'il faut « une action » pour se déplacer sur la case. Le chiffre 2 signifie qu'il en faut deux. Le chiffre 0 indique que le déplacement est **gratuit**. Considérons à nouveau la carte précédente, mais avec des **poids**, cette fois.

```
12,6
###0####0###
#A0033330B##
99###.###999
####...#####
###000000###
###0####0###
```

Voici donc quel serait la solution de cette carte a présent. Notez l'absence de chiffres dans cette sortie, permettant de simplifier la lecture.

```
12,6
###X####X###
#AXX....XB##
..###.###...
####...#####
###XXXXXX###
###X####X###
```

A la place de renoncer à afficher les chiffres, vous pouvez préférer l'utilisation des couleurs du terminal afin de mettre en relief le tracé, les murs ou les chemins. Renseignez-vous sur internet comment réaliser ces couleurs.

07 – Grande carte

Votre programme va maintenant rencontre une difficulté d'un nouveau type : vous allez charger une grande carte. Pour des raisons de praticité, cette carte sera au format image. Vous pouvez utiliser bunny_load_pixelarray pour charger cette image.

Voici la définition des couleurs : noir indique un mur. Blanc est l'équivalent de 0. Les nuances de gris entre 10 et 90 représentent les chiffres de 1 à 9. Le point rouge est le point de départ, le point bleu le point d'arrivé.

Votre sortie en cas d'image devra être graphique : affichez la carte et tracer en vert le chemin que vous aurez trouvé. N'hésitez pas à faire de grosse traces plutôt qu'une fine ligne pour qu'on le voit bien.