TP Programmation C/C++
Structures fondamentales d'algorithmie

© 2017 *tv* <tvaira@free.fr> v.1.1

Sommaire

Les objectifs de ce tp sont de comprendre et mettre en oeuvre des variables dans les structures fondamentales d'algorithmie : enchaînements, alternatives, itérations. Les exercices sont extraits du site www.france-ioi.org.

Les critères d'évaluation de ce TP sont :

- le minimum attendu : on doit pouvoir fabriquer un exécutable et le lancer !
- le programme doit énoncer ce qu'il fait et faire ce qu'il énonce !
- le respect des noms de fichiers
- le nommage des variables ainsi que leurs types, l'utilisation de constantes
- le code est fonctionnel
- la simplicité du code

Exercices : Un parcours semé d'embûches

En l'an 2132, une planète habitée a été découverte aux confins de la galaxie, et baptisée Algoréa. Dix ans plus tard, vous êtes envoyé(e) en mission autour de cette planète dans le but d'établir un tout premier contact avec ses habitants. Malheureusement, la comète Hal a percuté votre vaisseau alors que vous vous approchiez d'Algoréa. Vous avez été contraint(e) de vous éjecter en urgence dans une capsule de sauvetage. Vous atterrissez sur la planète avec pour seul bagage un **robot** de maintenance.

Isolé(e) sur ce globe lointain, vous allez donc rencontrer ses habitants, découvrir leurs coutumes et créer des liens afin d'entamer une relation pacifique en attendant que la prochaine mission vienne vous récupérer. Vous allez pour cela profiter des capacités de votre robot ! Celui-ci est accompagné d'un **manuel**, qui vous sera bien utile dans votre parcours. Ce robot est capable d'imprimer du texte, d'afficher des caractères sur un écran, de se déplacer et de bouger des objets.

Vous remarquez aussi que le manuel propose de **programmer** le robot à l'aide de langages, en écrivant du code sur un clavier. Une fois votre choix effectué, vous vous dirigez vers un site qui semble abriter la vie.

Grâce à votre robot, vous allez pouvoir :

- enchaîner des suites d'instructions : Dans le fourré
- répéter (itérer) des instructions : Vendanges, Course avec les enfants
- effectuer des actions sous conditions : Bornes kilométriques, Bagarre générale, Concours de tir à la corde, , Type d'arbres, L'espion est démasqué !
- lire des entrées et produire des résultats en sortie : Calculatrice, Graduation de thermomètres, Calcul des dénivelées, Moyenne des notes, Tarif du bateau, Roue de la fortune

✍ *Les langages C/C++ nécessitent un certain niveau de maîtrise. Il faut donc pratiquer intensément. Il n'est pas rare que les débutants commettent quelques erreurs évitables ... L'Annexe n°1 (page ??) recense les erreurs fréquentes des débutants.*

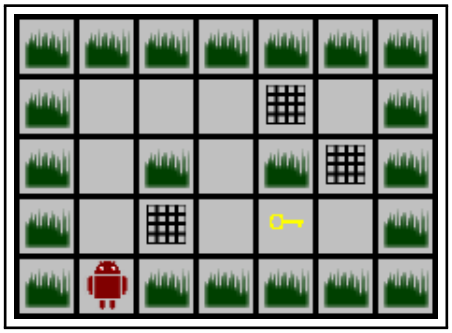
Dans le fourré

Sur l'étroit chemin qui mène à la montagne, vous parvenez à une énorme porte qui vous empêche de passer. Les villageois vous ont prévenu que la clé de cette porte demeure dans le fourré qui se trouve juste à côté. Celui-ci est organisé selon des cases avec des pièges et des buissons, dont on vous a fourni le plan.

Cependant, vous n'êtes pas sûr que le plan soit tout à fait exact et vous redoutez les pièges du fourré. Vous décidez donc d'envoyer votre robot chercher la clé pour vous.

☞ Ce que doit faire votre programme :

Votre programme doit diriger votre robot dans la grille suivante :



Le robot se trouve initialement à l'entrée du fourré et doit atteindre la case où se trouve la clé sans passer par les cases où se trouvent des buissons infranchissables ni celles qui contiennent un piège . Vous n'avez pas besoin de programmer le chemin retour.

Afin que vous puissiez manipuler le robot, nous avons créé un module appelé « robot ».

Pour disposer de ces instructions en C, vous devez inclure la ligne suivante en haut de votre programme :

```
#include "robot.h"
```

Pour déplacer le robot dans le fourré, nous proposons les quatre instructions suivantes :

– Aller en haut ;
– Aller en bas ;
– Aller à gauche ;
– Aller à droite.

Chacune demandant au robot de se déplacer d'une case dans une direction sur la grille.

En C, vous devrez les écrire comme suit :

```
allerHaut();
allerBas();
allerGauche();
allerDroite();
```

✍ *Notez bien que le robot ne tourne pas : il se déplace de case en case sur la grille, vers le haut, le bas, la gauche ou la droite.*

☞ Exemple :

Pour vous aider, voici en guise d'exemple un programme qui envoie le robot dans un buisson en trois déplacements.

```
#include <stdio.h>
#include "robot.h"

int main()
{
   allerHaut();
   allerHaut();
   allerGauche();
}
```

main.c

```
$ make

$ ./fourre
  1234567
1 BBBBBBB
2 B   P B
3 B B BPB
4 BRP C B
5 B BBBBB

  1234567
1 BBBBBBB
2 B   P B
3 BRB BPB
```



```
4 B P C B
5 B BBBBB

Le robot est entré dans un buisson !
```

✍ *Pour fabriquer le programme, vous devez d'abord taper la commande* ***make****.*

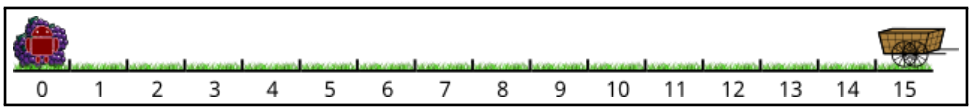
Question 1. Écrire le programme dans le fichier `main.c`. Tester et valider.

Vendanges

Par cette belle journée d'automne, vous accompagnez les villageois qui partent faire les vendanges. Vous remarquez tout de suite un homme faisant de manière répétitive des allers-retours entre les cueilleurs et la charrette qui doit être remplie à ras bord avant la fin de la journée. Cet homme tombe de fatigue et vous lui proposez de terminer le chargement. Bien évidemment, vous allez utiliser votre robot pour effectuer cette tâche à votre place.

☞ Ce que doit faire votre programme :

Le champ est représenté ci-dessous :



Le robot est initialement tout à gauche, là où se trouve le grand tas de raisins. Il devra, **20 fois** :

- ramasser des raisins pour remplir la hotte de ramassage ;
- se rendre à la charrette ;
- déposer le contenu de la hotte ;
- revenir au point de départ.

Vous disposez des quatre opérations suivantes :

- `int lirePosition()` : Affiche et retourne la position du robot
- `int allerGauche()` : Aller à gauche et retourne la position du robot
- `int allerDroite()` : Aller à droite et retourne la position du robot
- `int rammasser()` : Ramasse du raisin sur le tas et retourne 1 si du raisin a été rammasé sinon 0
- `int deposer()` : Dépose du raisin sur la charette et retourne 1 si du raisin a été déposé sinon 0

A chaque fois que du raisin a été déposé quelque part, la fonction `deposer()` affiche l'état de la situation :

```
 T  1  2  3  4  5  6  7  8  9  10  11  12  13  14  C
19  .  .  .  .  .  .  .  .  .   .   .   .   .   .  01
```

Vous disposez aussi de la fonction `afficherJeu()` qui affichera à tout moment l'état de la situation :

```
 T  1  2  3  4  5  6  7  8  9  10  11  12  13  14  C
20  .  .  .  .  .  .  .  .  .   .   .   .   .   .  00
```

☞ Exemples :

```c
#include <stdio.h>
#include "robot.h"
```

```c
int main()
{
    lirePosition();
    afficherJeu();

    return 0;
}
```

main.c

Pour fabriquer et tester le programme :

```
$ make
Ou
$ make cheat

$ ./vendanges
position : 0 (T)
 T  1  2  3  4  5  6  7  8  9  10  11  12  13  14  C
20  .  .  .  .  .  .  .  .  .   .   .   .   .   .  00
```

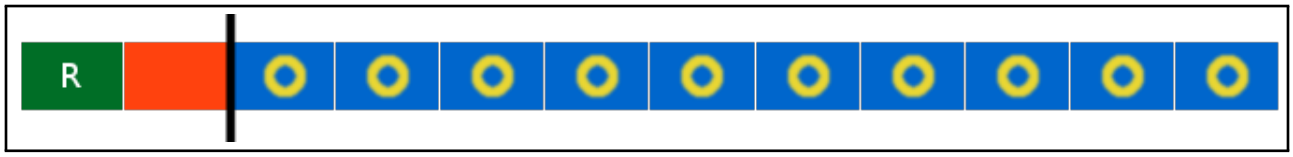
Question 2. Écrire le programme dans le fichier `main.c`. Tester et valider.

Course avec les enfants

C'est l'heure du cours de sport et les enfants font une petite course à laquelle vous décidez d'inscrire votre robot. Le principe est simple : il faut aller chercher le premier anneau, revenir le déposer derrière la ligne, puis aller chercher le second anneau et le ramener, et ainsi de suite ... jusqu'à ce que le dixième anneau ait été ramené derrière la ligne, le but du jeu étant d'aller le plus vite possible (sans tricher bien sûr !).

☞ Ce que doit faire votre programme :

Votre robot doit partir de la case de gauche (en orange, marquée D), aller chercher les anneaux (les ronds sur fond bleu) dans l'ordre (de gauche à droite) et les ramener un par un à gauche de la case de départ (en vert, marquée R).




```
-1 00 01 02 03 04 05 06 07 08 09 10
00 D  1  1  1  1  1  1  1  1  1  1
```

Vous disposez des quatre opérations suivantes :

- `int lirePosition()` : Affiche et retourne la position du robot
- `int allerGauche()` : Aller à gauche et retourne la position du robot
- `int allerDroite()` : Aller à droite et retourne la position du robot
- `int rammasserAnneau()` : Ramasse un anneau et retourne 1 si un anneau a été rammasé sinon 0
- `int deposerAnneau()` : Dépose un anneau et retourne 1 si un anneau a été déposé sinon 0

A chaque fois qu'un anneau a été déposé quelque part, la fonction `deposerAnneau()` affiche l'état de la situation :

```
01 D  0  1  1  1  1  1  1  1  1  1
```

Vous disposez aussi de la fonction `afficherJeu()` qui affichera à tout moment l'état de la situation.

☞ Exemples :

```c
#include <stdio.h>
#include "robot.h"

int main()
{
    lirePosition();
    afficherJeu();

    return 0;
}
```

main.c

Pour fabriquer et tester le programme :

```
$ make
Ou
$ make cheat

$ ./anneaux
position : 0 (D)
R  D 01 02 03 04 05 06 07 08 09 10
00 D  1  1  1  1  1  1  1  1  1  1
```

Question 3. Écrire le programme dans le fichier `main.c`. Tester et valider.

Bornes kilométriques

Le long de la route sur laquelle vous marchez se trouvent des bornes. Sur chaque borne est écrit le nombre de kilomètres séparant la position actuelle du bout de la route. Si la borne numéro zéro se trouve derrière vous, alors les numéros augmentent au fur et à mesure que vous avancez. Au contraire, si la borne zéro se trouve devant vous, alors les numéros diminuent au fur et à mesure que vous avancez.

Vous avez noté le numéro de la borne de laquelle vous êtes parti(e) ce matin, ainsi que le numéro de la borne à laquelle vous êtes arrivé(e) ce soir. Vous souhaitez calculer la distance que vous avez parcourue dans la journée.

☞ Ce que doit faire votre programme :

Votre programme doit lire deux entiers, correspondant à deux numéros de bornes kilométriques, et il doit afficher la distance séparant ces deux bornes. Notez que le résultat doit être un nombre positif ou nul.

☞ Exemples :

```
$ gcc bornes-kilometriques.c -o bornes-kilometriques

Ou :

$ g++ bornes-kilometriques.c -o bornes-kilometriques
```



```
$ ./bornes-kilometriques
152
189
Distance : 37

$ ./bornes-kilometriques
814
786
Distance : 28
```

Question 4. Écrire le programme `bornes-kilometriques.c`.

Question 5. Tester les 2 exemples donnés ci-dessus. Quel est le test qui manque pour valider le programme ?

Bagarre générale

À peine arrivé dans le village, voilà qu'une bagarre générale est sur le point d'éclater ! Tout en vous mettant à l'abri, vous tâchez de savoir ce qui se passe. On vous explique que le village est principalement composé de deux grandes familles rivales qui ne se supportent pas. Tout sujet étant une source de discorde possible, ils avaient décidé que les superficies de leurs champs respectifs ne devaient pas être trop différentes afin de ne pas attiser la jalousie de la famille opposée. Mais voilà que le patriarche des Arignon suspecte qu'un des champs des Evaran est trop grand ! Vous décidez de les aider ; mais la tâche ne sera pas facile, chacun gardant jalousement secrète la superficie réelle de ses champs.

☞ Ce que doit faire votre programme :

Votre programme devra lire deux entiers, la superficie d'un champ des Arignon et la superficie d'un champ des Evaran. Si l'un des champs est plus grand d'au moins 10 m^2 (strictement) que l'autre champ, alors il faudra afficher le texte « La famille X a un champ trop grand », « X » devant bien sûr être remplacé par « Arignon » ou « Evaran » selon le cas.

☞ Exemples :

```
$ ./bagarre-generale
42
54
La famille Evaran a un champ trop grand

$ ./bagarre-generale
10
20
```

✍ *Dans le second exemple, il n'y a rien à afficher.*

Question 6. Écrire le programme `bagarre-generale.c`.

Question 7. Tester les 2 exemples donnés ci-dessus. Quel est le test qui manque pour valider le programme ?

Calculatrice

Vous êtes dans la boutique du plus grand marchand de la ville, à la recherche d'un certain nombre d'ingrédients. Malheureusement pour vous, c'est la période du grand inventaire et cela peut durer très longtemps ! Vous décidez de les aider en programmant une calculatrice.

☞ Ce que doit faire votre programme :

Votre programme devra lire deux nombres entiers et une lettre représentant l'opération à effectuer ('s' pour la somme et 'p' pour le produit) et affichera le résultat de l'opération.

☞ Exemples :

```
$ ./calculatrice
5 4 s
9

$ ./calculatrice
2 3 p
6
```

Question 8. Écrire le programme `calculatrice.c`. Tester et valider.

Graduation de thermomètres

Les habitants du village utilisent beaucoup de thermomètres différents : certains pour l'été, d'autres pour l'hiver, d'autres pour la température de l'eau, etc. Ce qui change d'un thermomètre à l'autre, ce sont les valeurs de la température minimale et de la température maximale lisibles sur la graduation. Les habitants aimeraient pouvoir imprimer facilement la graduation des températures possibles pour chaque thermomètre.

☞ Ce que doit faire votre programme :

Étant données deux températures entières `tempMin` et `tempMax`, votre programme doit afficher toutes les températures comprises entre les deux bornes incluses.

☞ *Attention, la saisie des deux bornes peut être inversée. Vous devez vous assurer que* ***`tempMin`*** *et* ***`tempMax`*** *soient valides avant d'afficher les graduations.*

☞ Exemples :

```
$ ./graduation-thermometres
9
14
9
10
11
12
13
14

$ ./graduation-thermometres
14
9
9
10
11
```



```
12
13
14
```

Question 9. Écrire le programme `graduation-thermometres.c`.

Calcul des dénivelées

Aujourd'hui c'est l'étape de montagne et vous allez devoir franchir plusieurs cols. Vous allez passer votre temps à monter, descendre, remonter, redescendre, etc... Vous décidez de noter les différentes variations d'altitudes, afin de pouvoir calculer à la fin de la journée quelle est la dénivelée totale que vous avez montée ainsi que la dénivelée totale que vous avez descendue (les deux valeurs peuvent être différentes car vous ne retournez pas à votre point de départ).

☞ Ce que doit faire votre programme :

Votre programme lira d'abord un entier représentant le nombre de montées et descentes que vous avez réalisées. Pour chaque montée ou descente, il faut ensuite lire un entier représentant la variation d'altitude, cet entier étant strictement positif dans le cas d'une montée et strictement négatif dans le cas d'une descente (il n'y a rien à compter pour les tronçons qui sont bien à plat). Votre programme devra afficher l'altitude totale montée puis l'altitude totale descendue (ces deux nombres sont positifs).

☞ Exemples :

```
$ ./calcul-denivelees
5
4
7
-6
-3
2
altitude totale montée : 13
altitude totale descendue : 9
```

Question 10. Écrire le programme `calcul-denivelees.c` en utilisant une boucle `Tant Que` (`while`). Tester et valider.

Concours de tir à la corde

Vous avez passé la nuit dans une auberge. Au petit matin, un championnat de tir à la corde y est organisé. Vous ne souhaitez pas participer, mais l'aubergiste insiste pour que vous soyez impliqué dans l'événement. Vous décidez alors de vous engager dans les paris qui se font sur les deux équipes qui concourent.

Le championnat oppose deux équipes, contenant chacune autant de joueurs. Pour donner de l'allure et pimenter les paris, au début du championnat, tous les joueurs sont présentés, avec leur poids, avant d'aller tenir leur côté de la corde. Il est d'abord présenté un membre de la première équipe, puis de la deuxième, puis de la première, puis de la deuxième etc. jusqu'à ce que tous les joueurs soient passés. Afin de vous faire un premier pronostic, vous calculez le poids total de chaque équipe, avec votre robot.

☞ Ce que doit faire votre programme :

Votre programme devra lire un premier entier : le nombre de membres `nbMembres` qui constituent une équipe. Ensuite, il devra lire les poids (en kilogrammes), au total `nbMembres` $\times$ `2`, sachant que le premier

poids est celui d'un joueur de la 1re équipe, le deuxième poids celui d'un joueur de la 2e équipe, le troisième la 1re équipe, le quatrième la 2e équipe, etc.

Après avoir calculé le poids total de chaque équipe, vous devrez afficher le texte « L'équipe X a un avantage » (en remplaçant X par la valeur 1 ou 2), en considérant qu'une équipe est avantagée si elle a un poids total supérieur à celui de l'autre ou « Aucune équipe a un avantage » si les deux totaux sont identiques.

Vous afficherez ensuite le texte « Poids total pour l'équipe 1 : » suivi du poids de l'équipe 1, puis « Poids total pour l'équipe 2 : » suivi du poids de l'équipe 2 (voir l'exemple ci-dessous).

☞ Exemples :

```
$ ./tir-corde
3
40
80
50
50
60
10
L'équipe 1 a un avantage
Poids total pour l'équipe 1 : 150
Poids total pour l'équipe 2 : 140
```

✍ *Chaque équipe est composée de trois joueurs. Ceux de la première pèsent 40, 50 et 60 kg, tandis que ceux de la seconde font 80, 50 et 10 kg. Cela fait 150 kg opposés à 140 kg.*

Question 11. Écrire le programme `tir-corde.c` en utilisant une boucle `Pour` (`for`).

Question 12. Tester et valider le programme. Combien de tests faut-il effectuer ?

Type d'arbres

Alors que vous traversez une forêt vous ne pouvez vous empêcher d'admirer la végétation autour de vous et notamment les nombreuses espèces d'arbres. Malgré votre intérêt, vous êtes très mauvais botaniste et avez beaucoup de mal à identifier les différents arbres. Une personne que vous croisez vous donne quelques indications et vous décidez d'écrire un programme qui vous donnera le nom de l'arbre en fonction de ses caractéristiques.

☞ Ce que doit faire votre programme :

Il existe 4 types d'arbres :

– le "Tinuviel" fait moins de 5 mètres de haut et ses feuilles sont composées de plus de 8 folioles
– le "Calaelen" fait plus de 10 mètres de haut et ses feuilles sont composées de plus 10 folioles
– le "Falarion" fait moins de 8 mètres de haut et ses feuilles sont composées de moins de 5 folioles
– le "Dorthonion" fait plus de 12 mètres de haut et ses feuilles sont composées de moins de 7 folioles

Votre programme lira deux entiers, la hauteur et le nombre de folioles de l'arbre, et affichera le nom de l'arbre correspondant.

✍ *Toutes les inégalités sont à prendre au sens large, c'est-à-dire que "moins" signifie "moins ou égal" ou et "plus" signifie "plus ou égal".*

☞ Exemples :

```
$ ./type-arbre
12
12
Calaelen

$ ./type-arbre
4
9
Tinuviel

$ ./type-arbre
9
4
Inconnu
```

Question 13. Écrire le programme `type-arbre.c`. Tester et valider.

Question 14. Faut-il vraiment tester tous les cas ?

L'espion est démasqué !

Grâce à un certain nombre d'informateurs plus ou moins fiables, le chef de la police a recueilli des indications qui devraient lui permettre enfin de démasquer cet espion qui lui échappe depuis des semaines. La population de la ville étant relativement importante, il vous demande votre aide afin d'automatiser un peu les choses. Vous devez estimer la probabilité qu'une personne soit un espion.

☞ Ce que doit faire votre programme :

Votre programme doit lire le signalement d'une personne sous la forme de cinq entiers : sa taille en centimètres, son âge en années, son poids en kilogrammes, un entier valant 1 si la personne possède un cheval et 0 sinon, et un entier valant 1 si la personne à les cheveux bruns et 0 sinon.

On veut déterminer pour cette personne à quel point elle correspond aux 5 critères suivants :

– il aurait au moins 34 ans ;
– il aurait une taille supérieure ou égale à 178 cm et inférieure ou égale à 182 cm ;
– il pèserait strictement moins de 70 kg ;
– il n'a pas de cheval ;
– il a les cheveux bruns.

✍ *Lorsque cela n'est pas précisé explicitement, les inégalités sont au sens large (c'est-à-dire que "moins" signifie "moins ou égal" ou et "plus" signifie "plus ou égal").*

Vous devez vérifier tous les critères. S'ils sont vérifiés tous les 5, vous devez afficher « Très probable ». Si seulement 3 ou 4 sont vérifiés, vous devez afficher « Probable ». Si aucun n'est vérifié, vous devez afficher « Impossible », et dans les autres cas, vous devez afficher « Peu probable ».

☞ Exemples :

```
$ ./espion-demasque
180
40
65
0
1
Très probable
```


✍ *Vous utiliserez des variables booléenes. Vous pouvez aussi faire preuve de simplicité dans le calcul des critères.*

Question 15. Écrire le programme `espion-demasque.c`. Tester et valider.

Moyenne des notes

Des écoliers d'une école voisine aiment bien calculer la moyenne qu'ils vont avoir sur leur bulletin de notes avant de le recevoir. Cependant, ils ont beaucoup de notes, et ils aimeraient donc pouvoir utiliser un petit programme pour calculer leur moyenne sans se fatiguer.

Le nombre de notes dépend non seulement des matières, mais aussi des possibles absences de chacun. Aussi, le programme doit être capable de calculer la moyenne d'un nombre arbitraire de notes. C'est pourquoi vous concevez un programme auquel il faut d'abord fournir le nombre de notes dont il faut faire la moyenne, puis fournir les notes elles-mêmes (une par une).

☞ Ce que doit faire votre programme :

Votre programme doit d'abord lire un premier entier, qui décrit le nombre de notes obtenues. Ensuite, il doit lire chacune de ces notes, qui sont également des nombres entiers. Enfin, il doit afficher la moyenne (avec 2 chiffres après la virgule), les notes minimum et maximum de toutes ces notes.

☞ Exemples :

```
$ ./moyenne-notes
3
10
11
13
Moyenne : 11.33
Minimum : 10
Maximum : 13
```

✍ *Il est fortement conseillé de lire l'Annexe n°2 (page **??**) qui évoque les problèmes concernant les nombres réels (`float` et `double`).*

Question 16. Écrire le programme `moyenne-notes.c`.

Tarif du bateau

Vous venez d'apprendre qu'un concert de rock va se dérouler ce soir dans la ville située de l'autre côté du lac. Vous avez tout juste le temps d'y arriver en prenant le bateau. Cependant, vous n'êtes pas le seul à vouloir y aller, et il y a une longue queue pour acheter les tickets pour la traversée.

Comme il n'y a qu'une seule personne qui s'occupe à la fois de vendre les tickets et d'aider les gens à monter sur le bateau, vous vous rendez vite compte que vous allez certainement arriver en retard. La vente des tickets est particulièrement lente car il faut demander aux gens leur âge pour savoir s'ils doivent payer plein tarif ou bien tarif réduit.

Vous proposez que votre robot s'occupe de la vente des billets et que la personne responsable du bateau s'occupe uniquement d'aider les gens à monter sur le bateau. Il va donc vous falloir programmer votre robot assez vite pour arriver à temps au concert.

☞ Ce que doit faire votre programme :

Votre programme doit lire l'âge d'une personne et le nombre de traversées indiqué sur sa carte et afficher soit « Tarif réduit », soit « Tarif plein », soit « Gratuit » suivant les cas. Vous avez le droit à une

réduction si vous avez entre 12 et 25 ans (pour les jeunes) ou si vous avez plus de 60 ans (pour lres seniors) mais, si vous n'avez pas de réduction et que vous faites plus de 50 traversées par an, alors vous avez le droit à une traversée gratuite.

☞ Exemples :

```
$ ./tarif-bateau
27
10
Tarif plein

$ ./tarif-bateau
22
15
Tarif réduit
```

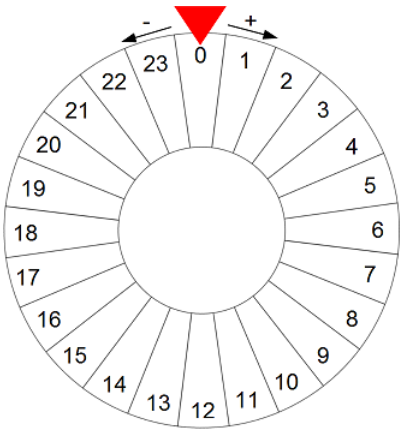
Question 17. Écrire le programme `tarif-bateau.c`.

Question 18. Tester et valider le programme. Indiquer les tests effectués.

La roue de la fortune

Lors de l'examen final à la fin de leurs études, la tradition veut que les élèves tirent un sujet au hasard. Tirer un numéro dans un chapeau n'étant pas très amusant, ils utilisent une roue comme celle-ci, qu'ils peuvent faire tourner dans un sens ou dans l'autre.

Les enseignants, par expérience, arrivent toujours à estimer de combien de "zones" la roue va tourner et peuvent aller chercher le sujet et revenir, pendant que la roue tourne encore. Il faut, pour cela, calculer rapidement sur quelle zone le curseur va s'arrêter, en fonction du nombre de zones dont la roue va tourner. A vous de jouer !



☞ Ce que doit faire votre programme :

Votre programme doit commencer par lire un entier `nbZones`. Sachant que la roue va tourner de `nbZones` zones, vous devez calculer (puis afficher) sur quelle zone le curseur va arriver.

Ainsi, si la route tourne de `+2` zones alors le curseur arrive sur la zone `2`, et si la roue tourne de `-2` zones, alors le curseur arrive sur la zone `22`.

☞ Exemples :

```
$ ./roue-fortune
25
1

$ ./roue-fortune
-50
22
```

✍ L'opérateur **modulo %** retourne le reste de la division euclidienne. Cet opérateur est très utilisé en informatique (nombre pair ou impair, multiple d'un nombre, intervalle, tampon circulaire ...). Le modulo sur 6 donnera toujours un nombre compris entre 0 et 5 par exemple :

```
0 % 6 retourne 0
1 % 6 retourne 1
2 % 6 retourne 2
6 % 6 retourne 0
7 % 6 retourne 1
8 % 6 retourne 2
...
```

Question 19. Écrire le programme `roue-fortune.c`.

Question 20. Tester et valider le programme. Indiquer les tests effectués.

Bilan

Vous avez découvert les notions suivantes :

– lire des entrées et produire des résultats en sortie
– enchaîner des suites d'instructions
– répéter (itérer) des instructions
– manipuler des variables et faire des opérations avec
– effectuer des actions sous conditions

Conclusion : parce qu'il est clair que vous débutez à peine votre carrière de programmeur, n'oubliez pas qu'écrire de bons programmes, c'est écrire des programmes corrects, simples et efficaces.

Rob Pike (ancien chercheur des Laboratoires Bell et maintenant ingénieur chez Google) :

> « *Règle n°4 : Les algorithmes élégants comportent plus d'erreurs que ceux qui sont plus simples, et ils sont plus difficiles à appliquer. Utilisez des algorithmes simples ainsi que des structures de données simples.* »

Cette règle n°4 est une des instances de la philosophie de conception KISS (*Keep it Simple, Stupid* dans le sens de « Ne complique pas les choses ») ou Principe KISS, dont la ligne directrice de conception préconise de rechercher la simplicité dans la conception et que toute complexité non nécessaire devrait être évitée.

