TP2: Arguments en ligne de commande et E/S – Correction

Module Sys1

Objectifs pédagogiques:

- Lecture / écriture de fichier en C (E2, E3, E5)
- Paramètres en ligne de commande (E1, E2, E3)
- Syntaxe de base du C (E1, E2, E3, E4, E5, E6)
- Syntaxe du switch (E2)
- Curiosité et culture informatique (E2, E3, E5)
- Représentation des nombres en C (E4, E6)

Les exercices et questions indiquées comme optionnelles ne sont là que pour votre culture et pour amuser les élèves qui le souhaitent. Ne les faites pas si ce n'est pas un jeu pour vous. Ils ne seront pas corrigés en classe, et ne donneront lieu à aucune évaluation.
Des questions issues d'exercices non-optionnels peuvent être posées en examen.

À préparer avant la séance : Exercice 1.

Réponse

Note pour l'animateur: La difficulté de ce TP est d'éviter de prononcer le mot "pointeur" autant que possible, car cette notion ne sera expliqué que dans le cours suivant. L'objectif de ce TP est de faire naître des interrogations qu'on assouvira en cours ensuite, mais il faut éviter de noyer les élèves d'explications peu claires données en petits groupes. Il faut savoir utiliser un petit argument d'autorité de type "fais ça tu comprendras plus tard pourquoi", mais en douceur.

Fin réponse

★ Exercice 1: Échauffement (à préparer avant).

▷ **Question 1:** Écrivez un programme C acceptant un nombre variable d'arguments sur la ligne de commande et affiche pour chacun d'eux le nombre de caractères correspondant. Ainsi, si ce programme est compilé sous le nom `arglen`, l'exécution de la commande
`arglen 0 bonjour 2.56 adieu` produira la sortie :

```
l'argument no 0 contient 6 caractere(s)
l'argument no 1 contient 1 caractere(s)
l'argument no 2 contient 7 caractere(s)
l'argument no 3 contient 4 caractere(s)
l'argument no 4 contient 5 caractere(s)
```

Indication: On peut retrouver la longueur d'une chaîne de caractères avec la fonction `strlen()`, utilisable après avoir inclu le fichier d'entête `<string.h>`.

Réponse

C'est l'occasion de réexpliquer argc/argv en s'appuyant sur la partie correspondante du poly.

arglen.c

```c
#include <stdio.h> // printf
#include <string.h> // strlen

int main(int argc, char** argv) {
   for (int i=0;i<argc;i++) {
      printf("l'argument no %d contient %d caractere(s)\n", i, strlen(argv[i]));
   }
   return 0;
}
```

Fin réponse

★ Exercice 2: Leet Speak

▷ **Question 1:** Écrivez un programme affichant le contenu d'un fichier *source* à l'écran. Utilisez pour cela les blocs de code fournis dans la refcard du C, et affichez chaque caractère avec la fonction `printf` et la chaîne de formatage `"%c"`.

Réponse

Les fonctions `fscanf` et `fprintf` n'ont pour ainsi dire pas été vues en classe, et il faudra sans doute que l'animateur réponde aux questions qui se posent ici.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {

   char filename[512];
   printf("Quel est le nom du fichier? ");
   scanf("%s", filename);

   FILE *fd = fopen(filename, "r");
   if (fd == NULL) {
      printf("Le fichier %s ne semble pas exister, car je n'arrive pas à l'ouvrir\n", filename);
      exit(1); // Fin brutale du programme
   }

   // On utilise le code de la refcard du C, sans chercher à trop comprendre
   char* line = NULL; size_t lgr = 0; ssize_t read;
   while ((read = getline(&line, &lgr, fd)) != -1) {
      // Dans la boucle, on affiche chaque caractère de la ligne lue par getline
      for (int i=0; i<read; i++) {
         printf("%c", line[i]);
      }
   }
   free(line);

   return 0;
}
```

Le code de la refcard marche bien, mais il y a quand même beaucoup de magie, surtout quand on ne sait pas ce que `free()` fait. En attendant la semaine 4, on peut vouloir utiliser `fscanf()` à la place de ce bloc de code. Il faut cependant "se souvenir" que cette fonction retourne EOF quand la fin du fichier est atteinte. Il faut donc consulter la valeur de retour de cette fonction sans utiliser `feof()`. Je ne sais pas pourquoi cette interface est si compliquée, au final. Heureusement que les pages de manuel sont assez explicites.

```c
FILE* IN = fopen("the fichier", "r");
char c;
while (fscanf(IN, "%c", &c) != EOF)) {
  printf("%c", c);
}
```

Fin réponse

▷ **Question 2:** Modifiez votre programme pour qu'il prenne son argument (`source`) depuis la ligne de commande grâce à `argv`. Si aucun argument n'est fourni en ligne de commande, il faut encore demander interactivement à l'utilisateur le nom du fichier concerné.

Réponse

L'erreur classique ici est de tenter de copier un tableau dans un autre comme on copie une variable dans une autre: `filename = argv[1]` C'est interdit en C, pour des raisons qu'on explicitera à la semaine 3.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // strcpy

int main(int argc, char** argv) {
   char filename[512];

   if (argc<2) {
      printf("Quel est le nom du fichier? ");
      scanf("%s", filename);
   } else {
      // Copier un tableau dans un autre nécessite d'utiliser la fonction strcpy ainsi.
      // filename = argv[1] est interdit en C
      strcpy(filename, argv[1]);
   }

   // Le reste du fichier est inchangé
   FILE* fd= fopen(filename, "r");
```

Si on se lance à changer filename en un `char*`, il faut utiliser `malloc()` pour réserver la place mémoire nécessaire. Mais cette fonction ne sera vue qu'à la semaine 4... C'est triste, le C sans tous les outils. Mais il faut bien commencer quelque part.

On peut quand même réussir à faire fonctionner le code sans strcpy. Il faut dupliquer le code d'ouverture du fichier dans un gros `if (argc<2)`, avec `FILE* fd;` au dessus du if pour qu'il soit visible de toute la fonction et non seulement des branches du if. Dupliquer du code est assez vilain, mais on n'a pas le choix pour l'instant.

Fin réponse

▷ **Question 3:** Modifiez votre programme pour qu'il change les minuscules en majuscules lors de l'affichage, sans changer les autres caractères. Un `man ascii` pourra être utile pour connaître la valeur numérique des lettres majuscules, mais on peut aussi utiliser le décalage `'a' - 'A'`.

Réponse

```
25    while ((read = getline(&line, &lgr, fd)) != -1) {
26       for (int i=0; i<read; i++) {
27          char c = line[i];
28          if ('a' <= c && c <= 'z') // Version facile sans connaître ascii
29            c = c + 'A' - 'a';
30 //       if ( 97 <= c && c <= 122) // Si vous voulez utiliser les codes ascii
31 //         c -= 32;                // mais à quoi bon s'embêter ?
32          printf("%c", c);
33       }
34    }
```

Fin réponse

▷ **Question 4:** Modifiez votre programme pour qu'il change maintenant les majuscules en minuscules, puis qu'il convertisse les 't' en '7', les 's' en '5', les 'i' en '1' et les autres changements indiqués sur la page wikipédia ≪Leet Speak≫.

Réponse

```
25    while ((read = getline(&line, &lgr, fd)) != -1) {
26       for (int i=0; i<read; i++) {
27         char c = line[i];
28         if ('A' <= c && c <= 'Z') // Version facile sans connaître ascii
29           c = c + 'a' - 'A';
30         switch (c) {
31         case 'a':
32           c = '4';
33           break;
34         case 't':
35           c = '7';
36           break;
37         case 'o':
38           c = '0';
39           break;
40         case 'e':
41           c = '3';
42           break;
43         case 's':
44           c = '5';
45           break;
46         case 'i':
47           c = '1';
48           break;
49         }
50         printf("%c", c);
51       }
52    }
```

Il faut résister à l'envie d'écrire chaque cas sur une seule ligne, car la régularité entre les programmes est préférable, et il n'est pas toujours possible d'écrire tout le cas sur la même ligne. On gardera donc l'habitude de mettre une seule instruction par ligne.

Fin réponse

★ Exercice 3: Le radoteur

Le radoteur est un algorithme pour fabriquer des mots nouveaux à partir d'une liste de mots. Bien que très simple, il produit souvent des nouveaux mots qui *auraient pu* être dans la liste de départ. Lorsqu'on lui donne une liste de pays, il en fabrique de nouveaux : palombie syldavie bordurie kafiristan lizbékistan...

Cet algorithme a été imaginé par Claude Shannon puis exploré et baptisé par Roland Moreno (chap. 6 de *Théorie du bordel ambiant*, 1990).

▷ **Question 1:** Implémentez cet algorithme, dont le pseudo-code est le suivant:

1. Choisir une lettre au hasard dans la liste de mot et l'écrire dans le mot en construction.
2. Chercher la prochaine occurence de cette lettre dans la liste de mot.
3. Considérons la lettre juste après celle cherchée. On l'ajoute au mot en construction, et cela devient le nouveau motif à chercher en retournant à l'étape 2.

Par exemple, si l'on commence depuis le 'b' de "albanie", cela produit "bitie aze boswetourie".

albanie algérie allemagne angola antigua arabie saoudite argentine arménie australie autriche azerbaïdjan bahamas bahreïn bangladesh belgique bélize bénin bhoutan biélorussie birmanie bolivie bosnie botswana brésil bulgarie burkina burundi cambodge cameroun canada chili chine chypre colombie congo corée croatie cuba danemark djibouti égypte équateur érythrée espagne estonie états unis éthiopie fidji finlande france gabon gambie géorgie ghana grèce guatemala guinée guyana haïti hollande honduras hongrie inde indonésie iran iraq irlande islande israel italie jamaïque japon jordanie kazakstan kenya kirghizistan koweït laos lésotho lettonie liban libéria lybie liechtenstein lituanie luxembourg macédoine madagascar malaisie malawi maldives mali malte maroc maurice mauritanie ...

Réponse

```c
#include <stdio.h>
#include <stdlib.h> // getline
#include <string.h>

int main(int argc, char** argv) {
  char buffer[102400] = ""; // On garde 100k de mémoire pour stocker la liste

  char filename[512];
  if (argc < 2) {
    printf("Quel est le nom du fichier? ");
    scanf("%s", filename);
  } else {
    // Copier un tableau dans un autre nécessite d'utiliser la fonction strcpy
    // ainsi. filename = argv[1] est interdit en C
    strcpy(filename, argv[1]);
  }

  FILE *fd = fopen(filename, "r");
  if (fd == NULL) {
    printf("Le fichier %s ne semble pas exister, car je n'arrive pas à l'ouvrir\n",
        filename);
    exit(1);
  }

  char *line = NULL;
  size_t lgr = 0;
  ssize_t read;
  while ((read = getline(&line, &lgr, fd)) != -1) {
    // Cette fois, on utilise strcat pour recopier line à la fin du buffer.
    // C'est pour ça qu'on a initialisé le buffer à "", pour qu'il ait une fin.
    strcat(buffer, line);
  }
  free(line);

  int buflen = strlen(buffer); // Pour ne pas recalculer la longueur à chaque fois

  int pos = 2; // Initialise à la 3ieme lettre pour faire comme dans l'énoncé
  for (int i=0; i<10000; i++) { // On affiche 10000 lettres
    printf("%c", buffer[pos]);
    int next = pos + 1;
    while (buffer[pos] != buffer[next]) {
      next += 1;
      if (next > buflen) // On arrive au bout
        next = 0;
```

```
    }
    // On a trouvé l'occurrence suivante de la même lettre. Regardons la lettre après
    pos = next + 1;
    if (pos > buflen) // On arrive au bout
      pos = 0;
  }
  printf("\n");
  return 0;
  }
```

Fin réponse

▷ **Question 2:** Testez votre implémentation avec des listes fournies. Vous pouvez aussi utiliser des nom de fleurs, de médicaments ou de plantes trouvés sur wikipédia, et vous amuser à mélangez les champs lexicaux.

Réponse

J'aime bien les plantes suivantes: Héborgoumaches, Bugle hallepe ou encore Pin mare Alline. Je les imagine volontiers dans une forêt d'altitude sur un plateau de Maltavékina occidentale.

Pour bien faire, il faudrait des listes de noms un peu plus grandes.

Fin réponse

▷ **Question 3: (optionnelle)** Au lieu de chercher et recopier une seule lettre à la fois, utilisez deux lettres pour chaque. Par exemple, après le groupe "ar" de **ar**gentine, on sélectionnera le "mé" de **ar**ménie. On peut faire varier le nombre de lettres cherchées et le nombre de lettres recopiées. La fonction `getopt(3)` est pratique pour doter votre programme de paramètres en ligne de commande.

▷ **Question 4: (optionnelle)** Pour de meilleurs résultats, on peut compliquer l'algorithme en le découpant en deux phases. Lors d'une phase d'apprentissage, on mesure dans la liste de mots la probabilité de trouver chacune des lettres après un groupe de deux lettres données. Par exemple, après "in", on peut trouver dans la liste complète des pays les lettres suivantes : e (10 occurrences), a, m, r, s (1 occurrence chacune) ou une fin de mot (2 occurrences). On utilise ensuite ces probabilités dans un second temps pour générer les mots. Pour plus d'informations, voir "Markov name generator" dans un moteur de recherche.

▷ **Question 5: (optionnelle)** Mélangez des proverbes (*Après la pluie, rien d'impossible ; L'union vient en mangeant ; veni, vedi, j'y reste*), des petites annonces (*Etudiant ch. dame sensible avec remorque*) ou des partitions de musique.

★ **Exercice 4: La suite de Fibonacci.** Il s'agit d'une suite d'entiers définie par $fib(0) = fib(1) = 1$ et $fib(n+2) = fib(n+1)+fib(n)$: chaque nouvel élément est la somme des deux précédents. On trouve les premiers termes de cette suite à l'adresse suivante: `https://r-knott.surrey.ac.uk/Fibonacci/fibtable.html`

▷ **Question 1:** Écrivez un programme qui demande un nombre n à l'utilisateur puis calcule $fib(n)$ de façon itérative, en ne conservant que trois valeurs successives en mémoire.

▷ **Question 2:** Quelle est la valeur de $fib(100)$ d'après votre programme?
Pourquoi n'est-ce pas 354 224 848 179 261 915 075?

Réponse

On atteint la limite des entiers représentatifs avec le type `int` : c'est F(44). Les `int` sont codés sur 32 bits (le plus grand est donc 2 147 483 647). Les nombres suivants sont donc incorrects, voire négatifs pour certains.

Fin réponse

▷ **Question 3:** (Optionnelle) Modifiez votre programme pour qu'il calcule correctement $fib(92)$, c'est-à-dire 7 540 113 804 746 346 429.

Réponse

Utiliser des `long long unsigned` permet d'aller jusqu'à $fib(92) = 7\,540\,113\,804\,746\,346\,429$ grâce à un bon usage des 64 bits.

Fin réponse

▷ **Question 4:** (Optionnelle) Saurez-vous corriger votre programme C pour qu'il calcule correctement $fib(186) = 332\,825\,110\,087\,067\,562\,321\,196\,029\,789\,634\,457\,848$? Comment aller encore plus loin?

Réponse

Cela demande cette fois d'utiliser des entiers non-signés sur 128 bits, ce qui n'est pas une partie de plaisir. En particulier, printf ne sait pas afficher ces monstres et il faut le faire à la main.

```c
#include <inttypes.h>
#include <stdio.h>
typedef unsigned __int128 uint128_t;
// typedef long long unsigned int longint_t;

// La libc ne sait pas afficher des entiers 128 bits, il faut le faire a la main
char* uint128_tostring(uint128_t n) {
  static char str[40] = {0};   // log10(1 << 128) + '\0'
  if (n == 0) {
    sprintf(str, "0");
    return &str;
  }

  char *s = str + sizeof(str) - 1; // start at the end
  while (n != 0) {
    if (s == str)
      return NULL; // never happens

    *--s = "0123456789"[n % 10]; // save last digit
    n /= 10;                     // drop it
  }

  return s;
}

int main() {
  int n = 2;        /* compteur */
  uint128_t x = 1; /* Fibonacci au rang n */
  uint128_t y = 1; /* Fibonacci au rang n+1 */

  int rank;
  printf("Valeur de n ? ");
  scanf("%d", &rank);

  printf("sizeof(long long unsigned)=%zu\n", sizeof(long long unsigned));
  printf("F(0) = 0\n");
  printf("F(1) = 1\n");
  printf("F(2) = 1\n");
  while (n < rank) {
    /* on passe du rang n au rang n+1 */
    uint128_t z = x; /* il faut sauvegarder x */
    x = y;
    y = y + z;
    n = n + 1;

    // le calcul de la valeur est fini, mais il faut maintenant l'afficher
    printf("F(%i) = %s\n", n, uint128_tostring(y));
  }
  return 0;
}
```

Pour aller encore plus loin, il faudrait implémenter également l'addition d'un champ de bits plus long, ou utiliser une bibliothèque déjà existante.

Fin réponse

★ Exercice 5: Rien ne sert de courir (optionnel – d'après Anne-Cécile Orgerie).

La tortue est un animal ayant des mouvements limités et sur un terrain plat. Elle possède une position (`x` et `y` de type `double`) et une direction (un entier `dir` où `0` signifie nord, `1` est, `2` sud et `3` ouest).

▷ Question 1: Déplacements

Programmer les trois fonctions suivantes :

- `void go(double d)` qui fait avancer la tortue de d dans la direction courante;
- `void right()` qui change la direction de la tortue de 90 degrés vers la droite sans la déplacer;
- `void left()` qui change la direction de la tortue de 90 degrés vers la gauche sans la déplacer.

▷ Question 2: Affichage postscript

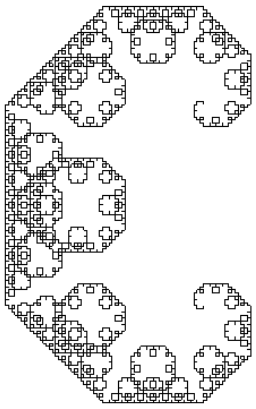
Modifiez votre programme pour qu'à chaque appel de `go()`, il affiche une ligne de la forme suivante: `` $x_1$ $y_1$ `moveto` $x_2$ $y_2$ `lineto stroke` `` avec (x_1, y_1) les coordonnées de départ et (x_2, y_2) celles d'arrivée du `go`.
Tout au début de votre programme, vous afficherez la ligne `%!postscript` ainsi que `showpage` tout à la fin.

La sortie de votre programme est à écrire dans un fichier d'extension `.ps`, que vous pourrez visualiser avec les programmes `gv` ou `evince` par exemple.

▷ **Question 3: Le premier déplacement.** Écrire un programme qui fait faire à la tortue un carré de 10 unités de côté et qui l'affiche dans un fichier postscript. Position initiale: `x = 297, y = 419`, et `dir = 0`.

▷ **Question 4: Tortue récursive.** Après d'intenses recherches, on a réussi à caractériser le mouvement de la tortue : la tortue a un mouvement récursif. Pour n entier, elle évolue selon la fonction `chemin(n)`. Si $n \leq 0$, avancer de 2 unités; sinon, tourner à droite, faire `chemin(n-1)`, tourner à gauche et faire `chemin(n-1)`.
Programmer la tortue pour qu'elle effectue son chemin avec $n = 12$.

▷ **Question 5: Tortue fatiguée.** Quand la tortue est fatiguée, son chemin est différent : Si $n \leq 0$, avancer de 2 unités; sinon, faire `chemin(n-1)`, tourner à gauche, faire `chemin(n-1)`, tourner à droite, faire `chemin(n-1)`, tourner à droite, faire `chemin(n-1)`, tourner à gauche et faire `chemin(n-1)`.
Programmer le chemin de la tortue fatiguée pour $n = 4$, représenté à droite.



Le programme `src/tortue.c` :

```
#include <stdio.h>

double x = 297,y = 419; /* milieu de la page */
int dir = 0; /* vers le nord */

void right ()
{
  dir = (dir + 1) % 4;
}

void left ()
{
  dir = (dir + 3) % 4; /* -1 modulo 4 */
}

void go (double pas)
{
  printf("%f␣%f␣moveto␣",x,y);
  if (dir == 0) y += pas; /* nord */
  else if (dir == 1) x += pas; /* est */
  else if (dir == 2) y -= pas; /* sud */
  else if (dir == 3) x -= pas; /* ouest */
  printf("%f␣%f␣lineto␣stroke\n",x,y);
}

void courbe (int n)
{
  if (n == 0) go(2);
  else {
    right();
    courbe(n - 1);
    left();
    courbe(n - 1);
  }
}

void koch(int n)
{
  if (n == 0) {
    go(2);
  }
  else {
    koch(n - 1);
    left();
    koch(n - 1);
    right();
    koch(n - 1);
    right();
    koch(n - 1);
    left();
    koch(n - 1);
  }
}

int main()
```

```
{
  printf("%%!postscript\n");
  //courbe(12);
  koch(4);
  printf("showpage\n");
  return 0;
}
```

★ **Exercice 6: Nombres en virgule flottante: Racines de Newton (optionel).**

La suite $x_{n+1} = \frac{1}{2}(x_n + \frac{y}{x_n})$ avec $x_0 = 1$ converge vers $\sqrt{y}$.

▷ **Question 1:** Écrivez un programme qui calcule $\sqrt{2}$ en utilisant cette formule.

Réponse

```
#include <stdio.h>
int main()
{
  double x=1;
  double y=(x+2/x)/2;
  while (y>=x+1e-10 || y<=x-1e-10)
  {
    double temp = y;
    y = (y+2/y)/2;
    x = temp;

    printf("%.10f\n",x);
  }

  return 0;
}
```

Fin réponse

▷ **Question 2:** Combien d'itérations sont nécessaires pour obtenir un résultat avec 10 chiffres significatifs? Il s'agit du premier entier n pour lequel $u_{n+1} - u_n < 10^{-10}$. Le format d'affichage `printf("%.12f", u)` permet d'afficher 12 chiffres après la virgule.

Réponse

La suite converge très vite (4 itérations). On trouve 1.4142135624.

Fin réponse

▷ **Question 3:** Écrire un programme C qui détermine la précision de l'ordinateur (la précision est telle que $1.0 + precision = 1.0$).

Réponse

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  double precision = 1.0;

  while (1.0 + precision != 1.0){
    printf("%.20lf\n", precision);
    precision /= 2.0;
  }
  printf("La␣precision␣est␣:␣%g\n", precision);

  return 0;
}
```

Fin réponse