ECUE «Introduction à la programmation »

Contrôle continu n°3 – 16 janvier 2012
sans document - durée 1 heure 30
Bruno Bouzy

Dans tous les exercices, les entrées clavier sont indiquées en caractères gras.

CORRIGE

Exercice 1 (4 points)

L'algorithme de `Babylone` calcule la racine carrée d'un nombre `A` avec une précision `P`. Il utilise une suite de nombres réels X_n tels que $X_{n+1} = (X_n + A/X_n)/2$. En C, programmer l'algorithme de `Babylone` en respectant les entrées sorties suivantes:

```
Calcul de la racine carree d'un nombre A avec une precision P.
Nombre A ? 2
Precision P ? 0.000001
Valeur initiale ? 1.8
x1 = 1.4555555582
erreur = 0.3444443941
x2 = 1.4148006439
erreur = 0.0407549143
x3 = 1.4142136574
erreur = 0.0005869865
x4 = 1.4142135382
erreur = 0.0000001192
```

L'utilisateur entre le nombre `A`, la précision `P` et la valeur initiale X_0 de la suite. A chaque itération, le programme affiche la valeur de X_n et l'erreur $e = |X_n - X_{n-1}|$ avec 10 décimales. Le programme s'arrête lorsque l'erreur `e` est inférieure à `P`. On pourra utiliser des variables `a`, `p`, `x`, `xsave`, `e` et `n`. On n'utilisera pas de fonction, ni de tableau.

```
#include <stdio.h>

int main() {
  printf("Algorithme de Babylone.\n");
  printf("Calcul de la racine carree d'un nombre A avec une
precision P.\n");

  float x, a, precision;

  printf("Nombre A ? ");
  scanf("%f", &a);

  printf("Precision P ? ");
  scanf("%f", &precision);

  printf("Valeur initiale ? ");
  scanf("%f", &x);

  int i=1;
  float erreur;

  do {
    float xsave = x;
    x = (x + a/x)/2;
    printf("x%d = %.10f\n", i, x);

    erreur = x - xsave;
    if (erreur<0) erreur = -erreur;
    printf("erreur = %.10f\n", erreur);

    i++;
  } while (erreur>precision);

  return 0;
}
```

Exercice 2

Au jeu du `Dobble` simplifié, il y a 9 cartes et 12 symboles. Les symboles sont: `+`, `x`, `*`, `=`, `~`, `!`, `w`, `#`, `&`, `@`, `%`, `$`. Chaque carte contient 4 symboles tous différents.

Par exemple,la carte `!#%+` est valide et la carte `!#%#` est invalide. 2 cartes possèdent exactement un symbole commun. Par exemple, la carte `!#%+` et la carte `@x*#` possèdent le symbole `#` en commun.

Le but de cet exercice est de programmer, en C, la construction d'un jeu de `Dobble` simplifié (questions 1 à 5) et de permettre à un utilisateur d'y jouer (question 6). On utilisera des fonctions et des tableaux.

Question 1 (0.5 point)

1) a) Définir 3 constantes `NB_CARTES`, `NB_SYMBOLES` et `NB_SYMBOLES_PAR_CARTE`.

```
#define NB_SYMBOLES 12
#define NB_SYMBOLES_PAR_CARTE 4
#define NB_CARTES    9
```

Question 2 (1.5 point)

2) a) Ecrire la déclaration et l'initialisation d'un tableau de 12 caractères `ts` contenant les 12 symboles du jeu.

```
char symboles[NB_SYMBOLES]
= {'+', 'x', '*', '=', '~', '!', 'w', '#', '&', '@', '%', '$'};
```

2) b) Ecrire une boucle `for` permettant d'afficher le tableau `ts` en respectant la sortie suivante:

```
symboles: + x * = ~ ! w # & @ % $
```

```
printf("symboles: ");
for(i=0; i<NB_SYMBOLES; i++)
  printf("%c ", symboles[i]);
printf("\n");
```

Question 3 (4 points)

`carte1` est la première carte du jeu. Elle est représentée par un tableau de 4 caractères:

```
char carte1[NB_SYMBOLES_PAR_CARTE];
```

3) a) Ecrire une fonction affichant un tableau de 4 caractères et dont l'en-tête est:

```
void imprimerCarte(char * c)
```

```
void imprimerCarte(char * c)
{
  int i;
  for(i=0; i<NB_SYMBOLES_PAR_CARTE; i++)
    printf("%c ", *(c+i));
  printf("\n");
}
```

3) b) Ecrire une fonction remplissant aléatoirement un tableau de 4 caractères et dont l'en-tête est:

```
void remplirCarteAleatoirement(char * c, char * ts)
```

`c` est le tableau (la carte) à remplir et `ts` le tableau des 12 symboles dans lequel on tire les 4 symboles aléatoirement. On utilisera la fonction `rand()` pour le tirage aléatoire. NB: la fonction `remplirCarteAleatoirement` ne remplit pas nécessairement la carte de manière valide.

```
void remplirCarteAleatoirement(char * c, char * ts)
{
  int i;
  for(i=0; i<NB_SYMBOLES_PAR_CARTE; i++)
    c[i] = ts[rand() % NB_SYMBOLES];
}
```

3) c) Ecrire une fonction retournant `1` si tous les symboles d'une carte sont distincts, `0` sinon, et dont l'en-tête est:

```
int carteValide(char * c)
```

```
int carteValide(char * c)
{
  int i, j;
  for(i=0; i<NB_SYMBOLES_PAR_CARTE; i++)
    for(j=0; j<i; j++)
      if (c[i]==c[j]) return 0;
  return 1;
}
```

3) d) Dans le programme `main` écrire un traitement itératif remplissant aléatoirement `carte1` tant que `carte1` n'est pas valide. Après ce traitement, afficher `carte1`. On utilisera les fonctions ci-dessus. La sortie écran de ce traitement correspondra par exemple à :

```
carte1: ! # % +
```

```
  do {
    remplirCarteAleatoirement(carte1, symboles);
  } while (carteValide(carte1)==0);
  printf("carte1: ");
  imprimerCarte(carte1);
```

Question 4 (4 points)

Dans cette question on s'intéresse à la validité d'une paire de cartes. `carte2` est la seconde carte du jeu. Elle est représentée par un tableau de 4 caractères:

```
char carte2[NB_SYMBOLES_PAR_CARTE];
```

4) a) Ecrire une fonction retournant le nombre de couples de symboles communs entre deux cartes. Son en-tête est:

```
int nombreSymbolesCommuns(char * c1, char * c2)
```

Exemple: pour `k1: @ # % +` et `k2: @ x * #` , la fonction retourne `2`.

```
int nombreSymbolesCommuns(char * c1, char * c2)
{
  int i, j, nc=0;
  for(i=0; i<NB_SYMBOLES_PAR_CARTE; i++)
    for(j=0; j<NB_SYMBOLES_PAR_CARTE; j++)
      if (c1[i]==c2[j]) nc++;
  return nc;
}
```

4) b) Ecrire une fonction retournant `1` si une paire de cartes est valide, `0` sinon. Son en-tête est:

```
int paireValide(char * c1, char * c2)
```

Pour ce faire, en utilisant les fonctions précédentes, cette fonction vérifiera que les deux cartes sont valides et que le nombre de couples de symboles communs est `1`.

```
int paireValide(char * c1, char * c2)
{
  if (carteValide(c1)==0) return 0;
  if (carteValide(c2)==0) return 0;
  if (nombreSymbolesCommuns(c1, c2)!=1) return 0;
  return 1;
}
```

4) c) Ecrire une fonction retournant le symbole commun d'une paire de cartes valides. Si la paire n'est pas valide, elle retourne le caractère `'\0'`. Son en-tête est:

```
char symboleCommun(char * c1, char * c2)
```

```
char symboleCommun(char * c1, char * c2)
{
  if (paireValide(c1, c2)==0) return '\0';
  int i, j;
  for(i=0; i<NB_SYMBOLES_PAR_CARTE; i++)
    for(j=0; j<NB_SYMBOLES_PAR_CARTE; j++)
      if (c1[i]==c2[j]) return c1[i];
}
```

4) d) Dans le programme `main`, à la suite de la question 3) d), écrire le traitement itératif remplissant aléatoirement `carte2` tant que la paire `{carte1, carte2}` n'est pas valide. Après ce traitement, afficher `carte2`. On utilisera les fonctions ci-dessus. La sortie écran de ce traitement correspondra par exemple à :

```
carte1: ! # % +
carte2: @ x * #
```

```
  char carte2[NB_SYMBOLES_PAR_CARTE];
  do {
    remplirCarteAleatoirement(carte2, symboles);
  } while (paireValide(carte1, carte2)==0);
  printf("carte2: ");
  imprimerCarte(carte2);
```

Question 5 (1 point)

Dans cette question, on s'intéresse à la validité d'un triplet de cartes. `carte3` est la troisième carte du jeu. Elle est représentée par un tableau de 4 caractères:

```
char carte3[NB_SYMBOLES_PAR_CARTE];
```

5) Dans le programme `main`, à la suite de la question 4) d), écrire le traitement itératif remplissant aléatoirement `carte3` tant que l'ensemble `{carte1, carte2, carte3}` n'est pas valide. Après ce traitement, afficher `carte3`. On utilisera les fonctions de la question 4. La sortie écran de ce traitement correspondra par exemple à :

```
carte1: ! # % +
carte2: @ x * #
carte3: @ + ~ &
```

```
char carte3[NB_SYMBOLES_PAR_CARTE];

do {
  remplirCarteAleatoirement(carte3, symboles);
} while (paireValide(carte1, carte3)==0
      || paireValide(carte3, carte2)==0);

printf("carte3: ");
imprimerCarte(carte3);
```

Question 6 (5 points)

Dans cette question, on suppose que les 9 cartes (c'est-à-dire 9 tableaux de 4 caractères) ont été remplies de manière valide. Ensuite, on a déclaré et initialisé un tableau supplémentaire `cartes` représentant le jeu de cartes de la manière suivante:

```
char * cartes[NB_CARTES];
cartes[0] = carte1;  cartes[1] = carte2;  cartes[2] = carte3;
cartes[3] = carte4;  cartes[4] = carte5;  cartes[5] = carte6;
cartes[6] = carte7;  cartes[7] = carte8;  cartes[8] = carte9;
```

`cartes` est un tableaux de pointeurs sur `char`. `carte[i]` contient l'adresse du tableau `cartei+1`.

On veut écrire une procédure de jeu dont l'en-tête est:

```
void jouer(char ** tc)
```

Dans le `main`, après la construction valide des cartes, cette procédure est appelée de la manière suivante:

```
jouer(cartes);
```

L'interface clavier écran de la procédure `jouer` correspond à l'exécution ci-dessous:

```
@ x * #
@ + ~ &
symbole commun ? @
Bravo!
@ x * #
! # % +
symbole commun ? !
symbole commun ? #
Bravo!
! # % +
% * = &
symbole commun ? +
symbole commun ? =
symbole commun ? a
symbole commun ? %
Bravo!
@ + ~ &
% * = &
symbole commun ? q
```

La procédure `jouer` est itérative. A chaque itération, elle tire au hasard 2 cartes distinctes parmi les 9 cartes. Elle les affiche à l'écran. Elle demande à l'utilisateur quel est le symbole commun aux 2 cartes. Tant que l'utilisateur n'a pas trouvé le symbole commun ou tapé `q`, elle repose la question. Si l'utilisateur a trouvé le symbole commun, elle affiche `Bravo!`. Si l'utilisateur a tapé `q`, la procédure se termine.

6) a) Ecrire une procédure donnant deux nombres aléatoires distincts compris entre `0` et `NB_CARTES-1` **(1 point)**. On utilisera un passage de paramètres par adresse:

```
void tirer2NombresDistincts(int * pn1, int * pn2)
```

```
void tirer2NombresDistincts(int * pn1, int * pn2) {
  *pn1 = rand() % NB_CARTES;
  do
    *pn2 = rand() % NB_CARTES;
  while (*pn2==*pn1);
}
```

6) b) Ecrire la procédure `jouer` en utilisant des `do while`, des appels aux fonctions `tirer2NombresDistincts`, `imprimerCarte`, `symboleCommun`, `scanf` et `printf` **(4 points)**.

```
void jouer(char ** tc) {
  srand(time(NULL));

  int q=0;
  do {
    int n1, n2;
    tirer2NombresDistincts(&n1, &n2);

    imprimerCarte(tc[n1]);

    imprimerCarte(tc[n2]);

    char sc = symboleCommun(tc[n1], tc[n2]);

    char cu;
    do {
      printf("symbole commun ? ");
      do scanf("%c", &cu); while (cu=='\n');
      if (cu=='q') return;
    } while (cu!=sc);

    printf("Bravo!\n");

  } while (1);
}
```