Programmation C (E. Lazard)
Examen du 20 janvier 2016

CORRECTION

(durée 2h)

I. Expressions

Donner les résultats affichés par le programme suivant :

Listing 1. *expr*

```
#include <stdio.h>

int f1(int i)  { return i++;                      }
int f2(int i)  { return --i;                      }
int f3(int *i) { printf("%d\n",(*i)--); return *i; }
int f4(int *i) { printf("%d\n",*i==0); return *i;  }
int f5(int *i) { ++(*i); return (*i)++;           }

int main() {
    int a, b;
    a=f1(0);  b=f2(2);  printf("a=%d, b=%d\n",a,b);
    a=f3(&b); printf("a=%d, b=%d\n",a,b);
    b=f4(&b);
    b=f5(&a);
    printf("a=%d, b=%d\n",a,b);
}
```

Corrigé :

- `f1` renvoie la valeur de `i`, la variable locale est modifiée. Donc `a` vaut 0
- `f2` renvoie la valeur de `i-1`, la variable locale est modifiée. Donc `b` vaut 1.
- `f3` affiche la valeur de `*i` (donc `b`), puis décrémente cette valeur (donc `b` passe à 0) et la renvoie (donc `a` passe à 0).
- `f4` affiche le résultat du test `*i==0` (0 ou 1) ; ne modifie pas `*i`.
- `f5` incrémente la valeur pointée par `i` (`a` passe à 1) puis renvoie cette valeur (`b` passe à 1) en incrémentant encore une fois `*i` (donc `a` passe à 2).

Ce programme affiche donc :

```
a=0, b=1
1
a=0, b=0
1
a=2, b=1
```

II. Conversion

Écrire une fonction `char *int2Str(int nbr);` qui transforme un entier dans la chaîne de caractères correspondante. On supposera que le type `int` est sur 4 octets et peut donc contenir des valeurs de l'intervalle [−2 147 483 648 ; 2 147 483 647], ces deux bornes étant respectivement représentées par les constantes `INT_MIN`

et `INT_MAX`. La fonction doit renvoyer un pointeur sur une nouvelle chaîne de caractères allouée contenant la représentation de la valeur numérique sous forme de chaîne de caractères. La zone allouée peut être choisie plus grande que nécessaire pour stocker la chaîne. Par exemple, avec un argument de `12345`, la fonction renvoie `"12345"` ; avec `-654`, la fonction renvoie `"-654"`… On pensera à gérer les cas particuliers.

On pourra utiliser les fonctions `strlen()` et `strcpy()`.

Corrigé :

Pour construire la chaîne, on récupère chaque chiffre en effectuant le modulo 10 de la valeur mais cela construit la chaîne en ordre inverse ; il faudra donc ensuite inverser la chaîne. Pour un nombre négatif, on travaille avec la valeur absolue et on ajoute le signe `'-'` avant d'inverser la chaîne. Deux cas particuliers : la valeur minimale pour laquelle on ne peut pas prendre la valeur absolue, et zéro.

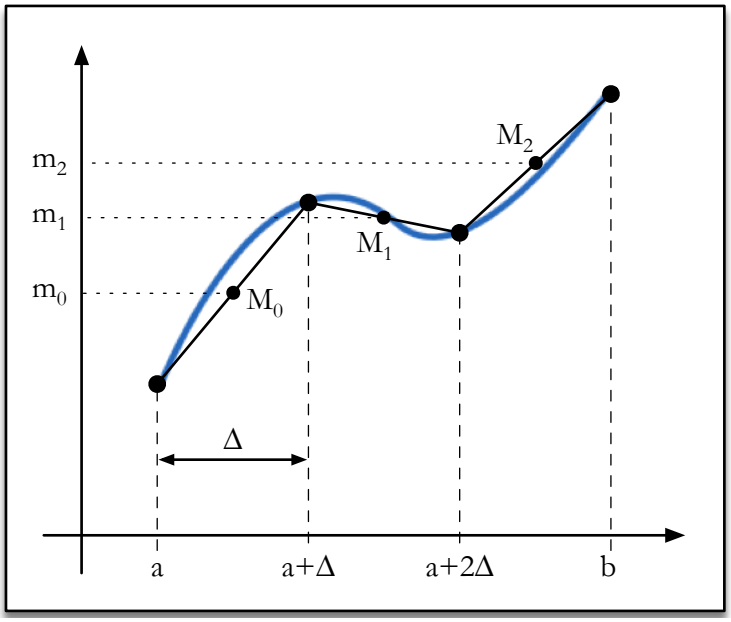
Listing 2. ***Conversion***

```
char *int2Str(int nbr) {
    char stockageValAbs[11];     /* car val abs max = 2147483647, soit 10 car. */
    char *finalStr = malloc(12); /* car il peut y avoir le signe '-' en plus   */
    char *save = finalStr;       /* on sauvegarde l'adresse de retour          */
    int valAbs, len, i = 0;
    if (nbr == INT_MIN) { /* premier cas particulier */
        strcpy(finalStr, "-2147483648");
        return finalStr;
    }
    if (nbr == 0) { /* 2e cas particulier */
        strcpy(finalStr, "0");
        return finalStr;
    }

    /* on travaille avec la valeur absolue */
    valAbs = (nbr >= 0 ? nbr : -nbr);
    while (valAbs > 0) {
        stockageValAbs[i++] = valAbs%10 + '0';
        valAbs /= 10;
    }

    stockageValAbs[i] = '\0';

    /* On a maintenant la valeur absolue du nombre écrite à l'envers */
    if (nbr < 0)
        *finalStr++ = '-';

    /* on recopie les caractères depuis le dernier */
    len = strlen(stockageValAbs);
    for (i = len-1; i >= 0; i--)
        *finalStr++ = stockageValAbs[i];
    *finalStr = '\0';
    return save;
}
```

III. Calculs flottants

On souhaite écrire une fonction ayant pour but de calculer l'intégrale d'une fonction f donnée, continue de $\mathbb{R}$ dans $\mathbb{R}$, par la méthode des trapèzes.

Rappel de la méthode



L'intervalle d'intégration est découpé en n petits intervalles (3 dans la figure ci-dessus) de largeur $\Delta = (b-a)/n$. On effectue ensuite une interpolation linéaire de f sur cet intervalle. Le point milieu du i^e trapèze (i allant de 0 à $n-1$) M_i a pour ordonnée

$$m_i = \frac{f(a+i\Delta) + f(a+(i+1)\Delta)}{2}$$

et l'aire du i^e trapèze est

$$\Delta \times m_i$$

On a donc une intégrale :

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} \Delta \left(\frac{f(a+i\Delta) + f(a+(i+1)\Delta)}{2} \right)$$

Chaque point milieu des trapèzes apparaît deux fois dans la somme, on obtient donc au final :

$$\int_a^b f(x)\,dx \approx \Delta \left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+i\Delta) \right)$$

Implémentation

Soit

```
double f(double);
```

une fonction réelle, continue, dont le code est déjà écrit et disponible.

Écrire une fonction

```
double Trapezes(double a, double b, int n);
```

qui calcule et renvoie l'intégrale de `f` par la méthode ci-dessus. `a` et `b` sont les bornes d'intégration et `n` est le nombre d'intervalles à utiliser pour le calcul.

Écrire une fonction

```
double Integrale(double a, double b, double epsilon);
```

qui calcule et renvoie l'intégrale de `f` entre `a` et `b`, avec une précision `epsilon`, en appelant de manière répétée la fonction précédente en augmentant de 1 le nombre d'intervalles à chaque appel, jusqu'à ce que la différence entre deux appels successifs soit inférieure à `epsilon` (on pourra utiliser la fonction `double fabs(double)` qui donne la valeur absolue de son argument).

Corrigé :

Listing 3. ***intégrale***

```
double Trapezes(double a, double b, int n) {
    double delta = (b - a) / n;
    double somme = 0;
    int i;
    for (i = 1; i < n; i++)
        somme += f(a + i*delta);
    return delta * ( (f(a)+f(b))/2 + somme );
}

double Integrale(double a, double b, float epsilon) {
    int i = 1;
    double oldSomme;
    double somme = 1e30;
    do {
        oldSomme = somme;
        somme = Trapezes(a, b, i++);
    } while (fabs(oldSomme - somme) > epsilon);
    return somme;
}
```

IV. Cryptographie

On souhaite écrire une fonction permettant le chiffrement/déchiffrement par substitution d'une chaîne de caractères. On supposera par la suite que la chaîne de départ, constituant le message que l'on veut chiffrer ou déchiffrer, n'est composée que de lettres (majuscules et minuscules), d'espaces et de sauts de ligne (caractère `'\n'`). Le principe du chiffrement est le suivant :

- Le chiffrement se fait à l'aide d'une clé (qui est une chaîne uniquement composée de chiffres). On parcourt simultanément les caractères du message et de la clé. Chaque caractère du message est alors décalé dans l'alphabet de la valeur numérique correspondant au caractère courant de la clé.
- Les espaces et les sauts de ligne sont ignorés : on les recopie tels quels dans le texte chiffré sans avancer la clé.
- Si le décalage imposé à un caractère du message dépasse `Z` (respectivement `z`), on reboucle sur `A` (respectivement `a`).
- Lorsqu'on arrive à la fin de la clé, on reboucle au début.

Ainsi, le message `"Architecture des Ordis"` codé à l'aide de la clé `"1985"` devient `"Bakmjcmhudzj ena Tsmqx"` :

```
Architecture des Ordis
198519851985 198 51985
Bakmjcmhudzj ena Tsmqx
```

La première lettre (`'A'`) est décalée de 1 position et donne B, la deuxième lettre (`'r'`) doit se décaler de 9 positions mais reboucle en fait sur `a`...

On pourra utiliser la fonction `strlen()`, ainsi que `isupper()` et `islower()` qui indiquent respectivement si le caractère passé en argument est une majuscule ou une minuscule. La fonction `atoi()` n'est pas utile dans cet exercice ; vous n'avez pas besoin du code ASCII de `a`, `A` ou `0`, utilisez les constantes caractères si nécessaire.

Écrire une fonction

```
char *crypto(char *msg, char *cle, int inverse);
```

qui chiffre ou déchiffre la chaîne `msg` avec la clé passée en paramètre suivant l'algorithme donné plus haut et renvoie une **nouvelle** chaîne composant le message chiffré/déchiffré. Si le paramètre `inverse` est à 0, la fonction doit chiffrer, s'il est à 1, elle doit déchiffrer.

On pourra supposer que `inverse` vaut forcément 0 ou 1 et que les deux chaînes sont au bon format : le premier pointeur pointe sur une chaîne non-vide composée uniquement de majuscules/minuscules, espaces et sauts de ligne, et la seconde ne contient que des chiffres.

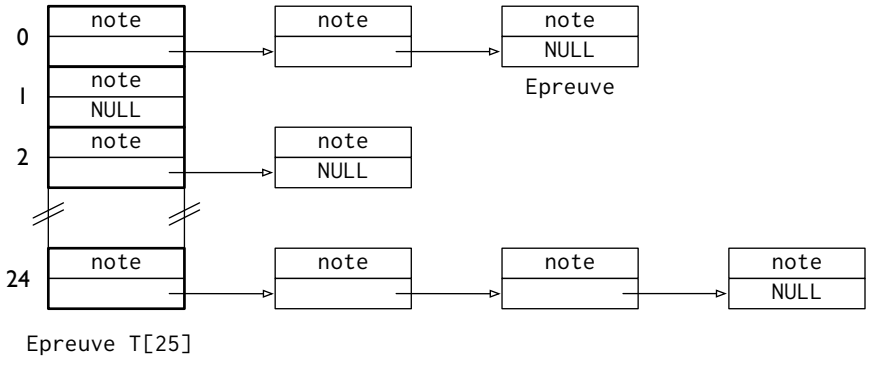
Corrigé :

Listing 4. ***Chiffrement/déchiffrement***

```
char *crypto(char *msg, char *cle, int inverse) {
    char newC, c;
    char *newStr = malloc(strlen(msg)+1); /* on alloue la place pour la nouvelle chaine */
    char *p = newStr;
    char *q = cle;
    int decalage;
    while (*msg) {
        if ((*msg == ' ') || (*msg == '\n'))
            *p++ = *msg++; /* on recopie sans rien faire */
        else {
            decalage = (*q++ - '0'); /* de combien décale-t-on ? */
            c = *msg++;
            /* avec le sens chiffrement/déchiffrement */
            newC = c + (inverse ? -decalage : decalage);

            if ( (isupper(c) && !isupper(newC)) || (islower(c) && !islower(newC)))
                /* et on reboucle sur A/a si nécessaire avec l'opération inverse */
                newC = newC + (inverse ? 26 : -26);

            *p++ = newC;
            if (*q == '\0') /* Est-on arrivé à la fin de la cle ? */
                q = cle;   /* Si oui, on recommence au début de la cle */
        }
    }
    *p = '\0';
    return newStr;
}
```

V. Listes chaînées

On souhaite gérer les résultats d'un examen d'une classe de 25 élèves. Chaque élève a passé au moins une fois l'examen mais a le droit de le repasser autant de fois qu'il le veut. Ces 25 élèves portent tous un numéro (de 0 à 24) et pour chacun, on mémorise ses résultats dans une liste chaînée d'entiers.




```
typedef struct epreuve {
    int note;
    struct epreuve *next;
} Epreuve;

Epreuve T[25];
```

`T[]` est un tableau global (donc accessible de toute fonction) dont chaque élément est une structure (du type `struct epreuve` dont le *typedef* est `Epreuve`) contenant la première note de l'examen et un pointeur sur

la cellule `Epreuve` suivante (si elle existe, c'est-à-dire si l'élève a repassé l'examen). Si l'élève repasse plusieurs fois l'examen, sa note est ajoutée à sa liste chaînée. Chaque tentative est notée par un entier entre 0 et 20.

1. Écrire deux fonctions

   ```
   int maxEleve(int eleve);
   ```

   ```
   double moyEleve(int eleve);
   ```

   qui renvoient respectivement la note maximum et la moyenne, calculées sur l'ensemble des tentatives effectuées pour l'examen par l'élève (dont le numéro entre 0 et 24 est passé en argument). (Comme chaque élève a passé au moins une fois l'examen, ces valeurs existent.)
2. Écrire une fonction

   ```
   double moyExamen(void);
   ```

   qui renvoie la moyenne de l'examen pour toute la classe, sachant que la note finale de chaque élève correspond à la note maximale qu'il a obtenue lors de toutes ses tentatives.

Corrigé :

Listing 5. ***Listes chaînées***

```c
typedef struct epreuve {
    int note;
    struct epreuve *next;
} Epreuve;

Epreuve T[25];

int maxEleve(int eleve) {
    int max = 0;
    Epreuve *p = &T[eleve];
    while (p != NULL) {
        if (p->note > max)
            max = p->note;
        p = p->next;
    }
    return max;
}

double moyEleve(int eleve) {
    int somme = 0;
    int nbr = 0;
    Epreuve *p = &T[eleve];
    while (p != NULL) {
        somme += p->note;
        nbr++;
        p = p->next;
    }
    return ((double) somme)/nbr;
}

double moyExamen(void) {
    int i;
    int somme = 0;
    for (i = 0; i < 25; i++)
        somme += maxEleve(i);
    return somme/25.0;
}
```