Examen de langage C
2024

Gabriel Dauphin

La durée de l'examen est de 2h. Vous pouvez utiliser le polycopié de C que vous aviez l'an dernier ainsi que celui de cette année dans leur version en ligne. Ils sont disponibles sur

`https://www-l2ti.univ-paris13.fr/~dauphin/`

Si vous estimez que l'énoncé a une erreur, proposez une correction en l'indiquant en en-tête du programme. À l'issue de l'examen, vous envoyez un mail à

`gabriel.dauphin@univ-paris13.fr`

Vous mettez en objet (ou subject) HarC: Nom Prénom Ex1,2 lorsque vous avez fait les exercices 1 et 2. Vous mettez en fichier attaché les deux fichiers `.c` intitulés `ex1Nom.c` et `ex2Nom.c`. Ensuite, vous venez me voir pour **vérifier** que j'ai bien reçu le mail avec ses fichiers attachés. Attention dans la compilation avec `gcc`, lorsqu'on utilise la bibliothèque `math.h`, il est parfois très important de rajouter l'option `-lm` dans l'instruction de compilation.

Bon Courage.

Exercice 1 *Le but du programme est de compter le nombre de termes d'une expression entrée juste après le nom du programme. Cette expression est une chaîne de caractères composée de caractères parmi* $0123456789+-*/$. *Un nombre est définie comme une succession de chiffres entre 0 et 9. Un opérateur est défini comme un caractère de type* $+-*/$. *Un terme est soit un nombre soit un opérateur. Ainsi on souhaite que ce résultat d'exécution.*

```
./ex1 35+4*2
35+4*2 avec 5 termes
./ex1 -3+4*2
-3+4*2 avec 6 termes
```

Le programme à réaliser consiste à compléter le programme suivant.

```
typedef enum BOUL {FAUX=0, VRAI=1,} BOUL;
void terme_suivant(const char ** str);
  //modifie l'adresse pointee en y indiquant l'endroit ou commence le terme suivant
BOUL est_op(const char * str);
  //renvoit VRAI si le premier terme de str est un operateur
BOUL est_fini(const char * str);
  //renvoit VRAI si le premier terme de str indique la fin de chaine de caractere..
```

Solution :

```
#include <stdio.h>
#include <assert.h>
#include <string.h>

typedef enum BOUL {
  FAUX=0,
  VRAI=1,
} BOUL;

void terme_suivant(const char ** str);
BOUL est_op(const char * str);
BOUL est_fini(const char * str);

int main(int argc,char * argv[]) {
  const char * str=argv[1];
  int nb_termes=0;
```

```
  assert((2==argc)&&"il faut un parametre a droite\n");
  while(!est_fini(str)) {
    terme_suivant(&str);
    nb_termes+=1;
  }
  printf("%s avec %d termes\n",argv[1],nb_termes);
  getchar();
  return 0;
}

BOUL est_fini(const char * str) {
  if (NULL==str)    return VRAI;
  if (str[0]=='\0') return VRAI;
  else              return FAUX;
}


BOUL est_op(const char * str) {
  const char list_op[]={'+','-','/','*'};
  int i;
  for(i=0;i<sizeof(list_op);i++) {
    if (list_op[i]==str[0]) {
      return VRAI;
    }
  }
  return FAUX;
}

void terme_suivant(const char ** str) {
  BOUL boul_init=est_op(*str);
  while(('\0'!=(*str)[0])&&(boul_init==est_op(*str))) {
    (*str)++;
  }
}
```

Exercice 2 *Le but du programme est d'afficher et de simplifier une expression définie par des données structurées. L'expression est une succession de termes, chaque terme est soit un opérateur* `OPT` *soit un opérande (nombre)* `OPD`*. La simplification consiste d'abord à détecter si après un opérateur + ou -, le nombre qui suit est négatif, puis à chaque fois que cela se produit à changer l'opérateur et le signe du nombre qui suit. Le but de l'exercice est de compléter le programme suivant. Dans cet exercice, un nombre peut être composé de plusieurs chiffres avec éventuellement un point entre les chiffres et éventuellement un signe moins au début.*

Voici le résultat attendu après avoir complété ce qui suit.

```
2.300000 - -6.400000
2.300000 + 6.400000

#include <stdio.h>
typedef enum {OPD=0, OPT=1 } Nature;
typedef struct {
  Nature type;
union { double OPD; char OPT; };
} Terme;
void afficher(const Terme expression[], int taille);
```

```
void adapter(Terme expression[], int taille);
int main(void) {
  Terme expression[]={{OPD,{.OPD=2.3}},{OPT,{.OPT='-'}},{OPD,{.OPD=-6.4}}};
  afficher(expression,sizeof(expression)/sizeof(expression[0]));
  adapter(expression,sizeof(expression)/sizeof(expression[0]));
  afficher(expression,sizeof(expression)/sizeof(expression[0]));
  getchar(); return 0;
}
```

Solution :

```
#include <stdio.h>

typedef enum Nature {
  OPD=0,   //OPD operande, c'est un nombre
  OPT=1    //OPT operateur, c'est un charactere
} Nature;

typedef struct Terme {
  Nature type;
  union {
    double OPD;
    char OPT;
  };
} Terme;

void afficher(const Terme expression[], int taille);
void adapter(Terme expression[], int taille);

 int main(void) {
   Terme expression[]={{OPD,{.OPD=2.3}},
                       {OPT,{.OPT='-'}},
                       {OPD,{.OPD=-6.4}}
                      };
   afficher(expression,sizeof(expression)/sizeof(expression[0]));
   adapter(expression,sizeof(expression)/sizeof(expression[0]));
   afficher(expression,sizeof(expression)/sizeof(expression[0]));
   getchar();
   return 0;
 }



 void afficher(const Terme expression[],int taille) {
   int i;
   for(i=0; i<taille; i++) {
     if (OPD==expression[i].type)
       printf("%lf ",expression[i].OPD);
     else
       printf("%c ",expression[i].OPT);
   }
   printf("\n");
 }

void adapter(Terme expression[], int taille){
```

```
   int i;
   for(i=1; i<taille; i++) {
     if ((OPD==expression[i].type)&&(expression[i].OPD<0)) {
       if ((OPT==expression[i-1].type)&&(expression[i-1].OPT=='+'))
         expression[i-1].OPT='-';
       else if ((OPT==expression[i-1].type)&&(expression[i-1].OPT=='-')) {
         expression[i-1].OPT='+';
         expression[i].OPD*=-1;
       }
     }
   }
 }
```

Exercice 3 *Le but du programme est de calculer une approximation* [1] *de la fonction exponentielle. Celle-ci est définie par*

$$exp(x) = \sum_{n=0}^{+\infty} \frac{x^n}{n!} \tag{1}$$

où $n!$ *désigne la factorielle. Mais cette expression pose des problèmes numériques, on la remplace par le système suivant*

$$a_0 = y_0 = 1 \quad et \quad \begin{cases} a_{n+1} = a_n \frac{x}{n} \\ y_{n+1} = y_n + a_n \frac{x}{n} \end{cases} \tag{2}$$

L'expérimentation indique qu'il est intéressant de calculer un nombre important de termes, par exemple $N = \lfloor 20|x|+3 \rfloor$ *où* $\lfloor \ldots \rfloor$ *est l'arrondi par en dessous. Vérifiez que le programme donne une bonne approximation pour* $e^0 = 1$, $e^1 \approx 2.718$, $e^1 e^{-1} = 1$.

Solution avec un fichier .h et .c pour la fonction testée :

```
####################Fichier ex3a.h
double c_exp(double x);
####################Fichier ex3a.c
#include <math.h>
double c_exp(double x) {
  int n;
  const int N=(int)(20*fabs(x)+3);
  double an=1,yn=1;
  for(n=0;n<N;n++){
    an=an*x/(n+1);
    yn=yn+an;
  }
  return yn;
}
####################Fichier ex3.c
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <math.h>
#ifndef __ex3a_h__
  #define __ex3a_h__
  #include "ex3a.h"
```

[1] La technique proposée ici n'est pas celle qui est généralement utilisé dans les processeurs.

```
#endif
//A compiler avec ex3a.c
//gcc -Wall -o ex3 sim/ex3.c sim/ex3a.h sim/ex3a.c
int main(int argc, char * argv[]) {
  assert(fabs(c_exp(1)*c_exp(-1)-1)<1e-2&&"calcul incorrect");
  assert((2==argc)&&"il faut entrer le nombre en parametre");
  double val=atof(argv[1]);
  printf("exp(valeur)=%lf\n",c_exp(val));
  getchar();
  return 0;
}
```

Exercice 4 *Le but du programme est maintenant de tester si l'approximation de la fonction exponentielle fonctionne bien en estimant son erreur quadratique moyenne au moyen de l'expression suivante :*

$$\mathcal{E} = \sqrt{E\left[(f(U) - e^U)^2\right]} \tag{3}$$

U est variable aléatoire suivant la loi uniforme sur $[-5, 5]$, f est la fonction approximée de l'exponentielle et E désigne l'espérance mathématique. La fonction f peut être celle implémentée dans l'exercice précédent ou bien la fonction exponentielle légèrement modifiée. Dans les deux cas il est souhaité que cette fonction soit implémentée en utilisant deux fichiers séparés `.h` *et* `.c`*, le premier incluant la déclaration de la fonction. $\mathcal{E}$ est estimée en s'inspirant du pseudo-code suivant, il est conseillé de ne pas utiliser de structures de tableaux.*

1: *Choisir $N = 10^4$*
2: *Tirer aléatoirement $u_1 \dots u_N$ suivant la loi uniforme sur $[-5, 5]$.*
3: *Calculer* $\mathcal{E} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (f(u_n) - e^{u_n})^2}$

Complétez le programme suivant

```
#include <stdlib.h>
#include <math.h>
#ifndef __ex3a_h__
  #define __ex3a_h__
  #include "ex3a.h"
#endif
double c_ecart_type(double (*)());
int main(void) {
  printf("%le\n",c_ecart_type(c_exp));
  return 0;
}
:
   rand()/(double)RAND_MAX;
:
```

Solution :

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <math.h>
#ifndef __ex21_h__
  #define __ex21_h__
  #include "ex21.h"
#endif

//gcc -Wall -o ex4 sim/ex4.c sim/ex3a.h sim/ex3a.c -lm
```

```
double c_ecart_type(double (*)());

int main(void) {
  //printf("%lf\n",exp(0));
  printf("%le\n",c_ecart_type(c_exp));
  return 0;
}


double c_ecart_type(double (*f)()) {
  double x,y=0;
  int i;
  int N=1e4;
  for(i=0;i<N; i++){
    x=rand()/(double)RAND_MAX;
    x=5*x-10;
    //printf("%lf ",f(x)-exp(x));
    y+=pow((f(x)-exp(x)),2);
  }
  return sqrt(y/N);
}
```