Programmation en C

Corrigé TD9

```
 1 /* ------------------  Calculatrice et Pile
 2                              */
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5 #include <string.h>
 6
 7 struct cellule_s {
 8   int val;
 9   struct cellule_s *next;
10 };
11
12 typedef struct cellule_s cellule;
13 typedef cellule *pile;
14 pile nil = NULL;
15
16 void error(char str[]) {
17   printf("Erreur: %s \n",str);
18   exit(1);
19 }
20
21 int empty(pile p) {
22   return (p == nil);
23 }
24
25 void push (int a, pile *pp) {
26   pile p1 = (pile) malloc (sizeof(cellule));
27   p1->val = a; p1->next = (*pp);
28   (*pp) = p1;
29 }
```

```
30
31 int pop(pile *pp) {
32   int a;
33   if (empty(*pp)) error("Pile vide !");
34   a = (*pp)->val;
35   (*pp) = (*pp)->next;
36   return a;
37 }
38
39 int eval(int argc, char* argv[]) {
40   int i,a,b;
41   pile p=nil;
42   for (i=1; (i<argc); i++) {
43     switch (argv[i][0]) {
44     case '-': a = pop(&p); b = pop(&p);
45       push(b-a,&p);
46       break;
47     case '+': a = pop(&p); b = pop(&p);
48       push(a+b,&p);
49       break;
50     case 'x': a = pop(&p); b = pop(&p);
51       push(a*b,&p);
52       break;
53     default: a = atoi(argv[i]);
54       push(a,&p);
55     }
56   }
57   a = pop(&p);
58   if (!empty(p)) error("Pile non vidée !");
59   return a;
60 }
61
62 int main (int argc, char *argv[])
63 {
64   int r;
65   r = eval(argc,argv);
66   printf(" = %d \n",r);
67   return 0;
```

```
68 }

 1 /* ------- Conversion notation infixe -> polonaise inverse
 2                                 */
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5 #include <string.h>
 6
 7 struct cellule_s {
 8   char val;
 9   struct cellule_s *next;
10 };
11
12 typedef struct cellule_s cellule;
13 typedef cellule *pile;
14 pile nil = NULL;
15
16 int empty(pile p) {
17   return (p == nil);
18 }
19
20 void error(char str[]) {
21   printf("Erreur: %s \n",str);
22   exit(1);
23 }
24
25 void push (char a, pile *pp) {
26   pile p1 = (pile) malloc (sizeof(cellule));
27   p1->val = a; p1->next = (*pp);
28   (*pp) = p1;
29 }
30
31 char pop(pile *pp) {
32   char a;
33   if (empty(*pp)) exit(1);
34   a = (*pp)->val;
35   (*pp) = (*pp)->next;
36   return a;
```

```
37 }
38
39 void conversion(int argc, char* argv[]) {
40   int i;
41   char a;
42   pile p=nil;
43   for (i=1; (i<argc); i++) {
44     a = argv[i][0];
45     switch (a) {
46     case '-': push('-',&p); break;
47     case '+': push('+',&p); break;
48     case 'x': push('x',&p); break;
49     case '[': break;
50     case ']': a=pop(&p);
51       printf("%c ",a); break;
52     default: printf("%d ",atoi(argv[i]));
53     }
54   }
55   printf("\n");
56 }
57
58 int main (int argc, char *argv[])
59 {
60   conversion(argc,argv);
61   return 0;}
```

```
 1 /* --------------------  Polynomes creux
 2                               */
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5
 6 struct cellule {
 7   int exp;
 8   float coef;
 9   struct cellule *next;
10 };
11
12 typedef struct cellule monome;
```

```
13 typedef monome *polynome;
14 polynome Zero = NULL;
15
16 polynome add_monome (float coef, int a, polynome P) {
17   polynome Q = (polynome) malloc (sizeof(monome));
18   Q->coef = coef; Q->exp = a; Q->next = P;
19   return Q; }
20
21 void affiche(polynome P) {
22   if (P == Zero) printf("0 \n");
23   else {
24     printf("%3.1f X^%d ",P->coef,P->exp);
25     if (P->next != Zero) {
26       printf("+ "); affiche(P->next);
27     }
28     else printf("\n");
29   }}
30
31 polynome add(polynome P1, polynome P2) {
32   if (P1 == Zero) return P2;
33   return add_monome(P1->coef,P1->exp, add(P1->next,P2)); }
34
35 polynome copy(polynome P) {
36   if (P == Zero) return Zero;
37   return add_monome(P->coef, P->exp, copy(P->next)); }
38
39 polynome mult_monome (float coef, int a, polynome P) {
40   polynome Q = copy(P);
41   polynome R = Q;
42   while (Q != Zero) {
43     Q->coef = Q->coef * coef;
44     Q->exp = Q->exp+a; Q = Q->next;
45   }
46   return R; }
47
48 polynome mult(polynome P1, polynome P2) {
49   polynome Q,T=Zero;;
50   while(P1 != Zero) {
```

```
51     Q = mult_monome(P1->coef,P1->exp,P2);
52     T = add(Q,T);
53     P1 = P1->next;}
54   return T; }
55
56 /************************* reduction
57 polynome insere(float coef, int a, polynome P) {
58   if (P == Zero) return add_monome(coef,a,Zero);
59   if (P->exp == a) {P->coef = P->coef + coef; return P;}
60   if (P->exp < a) return add_monome(coef,a,P);
61   return add_monome(P->coef,P->exp,insere(coef,a,P->next));
62 }
63
64 polynome reduction(polynome P) {
65   if (P == Zero) return Zero;
66   return insere(P->coef, P->exp, reduction(P->next));}
67
68 polynome addition(polynome P, polynome Q) {
69   return reduction(add(P,Q));}
70
71 polynome multiplication(polynome P, polynome Q) {
72   return reduction(mult(P,Q));}
73 ***********************************************/
74
75 int main (int argc, char *argv[]) {
76   polynome P=Zero,Q=Zero,R,T;
77   P = add_monome(1.0,1,P); P = add_monome(3.0,2,P);
78   P = add_monome(1.5,0,P); P = add_monome(2.5,3,P);
79   affiche(P);
80   Q = add_monome(3.0,1,Q); Q = add_monome(1.0,0,Q);
81   affiche(Q);
82   R = add(P,Q); affiche(R);
83   T = mult(P,Q); affiche(T);
84   return 0; }
```