Programmation II : SMI (S4)

TP N° 6 : Les Piles (Correction)

(Evaluation d'une expression arithmétique)

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

// Pile des operandes

typedef struct PileOpd{
    int tab[50];
    int sommet;
}PileOpd;

void initPileOpd(PileOpd * adrP){
    adrP->sommet = 0;
}

int estVidePileOpd(PileOpd P){
    return (P.sommet == 0)?1:0;
}

int estPleinePileOpd(PileOpd P){
    return (P.sommet == 50)?1:0;
}

void empilerOpd(PileOpd * adrP, int x){
    if(!estPleinePileOpd(*adrP)){
        adrP->tab[adrP->sommet++] = x;
    }
}

int depilerOpd(PileOpd * adrP){
    if(!estVidePileOpd(*adrP)){
        int x = adrP->tab[adrP->sommet-1];
        adrP->sommet --;
        return x;
    }
}

void afficherPileOpd(PileOpd adrP){
    int i;
    if(!estVidePileOpd(adrP)){
        for(i=0;i<adrP.sommet;i++){
            printf("%d ", adrP.tab[i]);
        }
        printf("\n");
    }
}

// Pile des operateurs

typedef struct PileOpr{
    char tab[50];
    int sommet;
}PileOpr;
```

```
void initPileOpr(PileOpr * adrP){
    adrP->sommet = 0;
}

int estVidePileOpr(PileOpr P){
    return (P.sommet == 0)?1:0;
}

int estPleinePileOpr(PileOpr P){
    return (P.sommet == 50)?1:0;
}

void empilerOpr(PileOpr * adrP, char x){
    if(!estPleinePileOpr(*adrP)){
        adrP->tab[adrP->sommet++] = x;
    }
}

char depilerOpr(PileOpr * adrP){
    if(!estVidePileOpr(*adrP)){
        return adrP->tab[--adrP->sommet];
    }
}

void afficherPileOpr(PileOpr adrP){
    int i;
    if(!estVidePileOpr(adrP)){
        for(i=0;i<adrP.sommet;i++){
            printf("%c ", adrP.tab[i]);
        }
        printf("\n");
    }
}

char * convertirInf2Post(char * expInf){
    int i, j;
    char * expPost = (char*)malloc(strlen(expInf)*sizeof(char));
    PileOpr pr;
    initPileOpr(&pr);
    i = 0;
    j = 0;
    char c;
    while((c = expInf[j++])!='\0'){

        if(c=='('){

        }
        if(isdigit(c)){
            expPost[i++] = c;
        }
        if(c == '+' || c == '-' || c == '*' || c == '/' || c == '%'){
            empilerOpr(&pr, c);
        }
        if(c==')'){
            char e = depilerOpr(&pr);
            expPost[i++] = e;
        }
    }
    expPost[i] = '\0';
    return expPost;
}
```

```
int calculer(int x, int y, char op){
    // printf("Evaluation de %d %c %d\n", x, op, y );
    switch(op){
        case '+' : return x + y;
        case '-' : return x - y;
        case '*' : return x * y;
        case '/' : return x / y;
        case '%' : return x % y;
    }
}

int evaluer(char * exp){
    char * expPost = convertirInf2Post(exp);
    // printf("%s ==> %d\n", expPost, strlen(expPost));
    PileOpd pd;
    initPileOpd(&pd);
    int i, a, b, val;
    char c;
    i = 0;
    while((c = expPost[i++])!='\0'){
        if(isdigit(c)){
            char s[2];
            s[0] = c; s[1] = '\0';
            empilerOpd(&pd, atoi(s));
            // afficherPileOpd(pd);
            // getch();
        }
        else{
            a = depilerOpd(&pd);
            // afficherPileOpd(pd);
            // getch();
            b = depilerOpd(&pd);
            // afficherPileOpd(pd);
            // getch();
            val = calculer(b, a, c);
            empilerOpd(&pd, val);
            // afficherPileOpd(pd);
            // getch();
        }
    }
    // afficherPileOpd(pd);
    val = depilerOpd(&pd);
    return val;
}

main(){
    char *A = "(3+(4*(8-6)))";
    printf("%d", evaluer(A));

    getch();
}
```