Programmation : TP 15

Objectifs du TP : type somme en C (union).

Expression arithmétique (TP9, le retour)

On considère des expressions arithmétiques : soit un nombre (flottant), soit une variable (un identificateur représenté par une chaîne), soit une somme ou un produit de deux expressions, soit l'opposé ou l'inverse d'une expression. On utilise les définitions de type suivantes :

```
typedef char variable;
struct expr_ {
  enum genre_ { Var, Float, Som, Prod, Opp, Inv } genre;
  union {
    float flottant;
    variable var;
    struct { struct expr_ *e1, *e2; } binaire;
    struct expr_ * unaire;
  } val;
};
typedef struct expr_* expr;
```

Il est judicieux de fabriquer une fonction de création pour chacun des cas de l'union. Ces fonctions doivent allouer la mémoire nécessaire et remplir les champs voulus. Par exemple, pour le cas `Var`, on pourra écrire la fonction suivante :

```
#define malloc_expr() ((expr)malloc(sizeof(struct expr_)))

expr var(variable v) {
  expr e = malloc_expr();
  e->genre = Var;
  e->val.var = v;
  return e;
}
```

Le code décrit précédemment se trouve dans le répertoire `~serveur/PROG/C`

1. Sur le modèle de `var()`, écrire les fonctions `expr flottant(float)`, `expr opp(expr)`, ...
   Remarque : `opp()` et `inv()` (resp. `som()` et `prod()`) peuvent être définies avec des macros utilisant une unique fonction `expr unaire(enum genre_, expr)` (resp. `expr binaire(enum genre_, expr, expr)`).
2. Construire avec ces fonctions l'expression :
   $$-\frac{1}{2}(1+3x)$$
3. Écrire une fonction `void print(expr)` d'affichage sur la sortie standard d'une expression. On obtiendra par exemple :
   ```
   sepia[327]%  ./tp15
   (-(1/(2.00))*(1.00+(3.00*x)))
   ```
4. Écrire une fonction `expr derive(expr, variable)` qui dérive une expression par rapport à une variable donnée.