Jean Delepine – Nikola Marchal

# Rapport de Projet LO21

## Choix de conception des structures de données :

```
typedef struct prop {
   char proposition[32];
   struct prop *next;
} Proposition;
```

Cette structure permet de créer proposition et d'avoir un pointeur suivant afin de dresser une liste de ces propositions.

---

```
typedef struct {
   Proposition *head;
} Premisse;
```

Comme indiqué précédemment, la structure Premisse permet de pointer sur la tête afin de former une liste de propositions ce qui est par définition une premisse.

---

```
typedef struct {
   Premisse *premisse;
   char conclusion[32];
} Rule;
```

La structure Rule est formée d'une premisse (structure Premisse) et d'une conclusion qui ne nécessite pas de structure

---

```
typedef struct BC {
   Rule regle;
   struct BC *next;
} BC;
```

Cette structure permet de créer notre base de connaissances composée de règles.

---

Au départ du projet, ces structures étaient bien plus complexes mais en prenant du recul il a été vu que des choix de direction n'étaient pas optimal, notamment l'implémentation d'un système de booléen, les structures ont donc été réduites de 7 à 4. La structure Premisse sert également à composer la base de faits car cette dernière n'est au final qu'une liste de propositions.

## **Algorithmes :**

```
Fonction moteurInference (baseConnaissances: BC, baseFaits: Premisse) : pas de renvoi
Début
    nouveauFaitAjoute: Booleen <- FAUX
    faire
      nouveauFaitAjoute <- FAUX
      rule: BC <- baseConnaissances
      Tant que(non estVide(r)) faire :
        r: Rule <- regle(rule)
        Si toutesPremissesVraies(r, baseFaits) alors :
          Si non propositionDansPremisse(baseFaits, conclusion(r) alors :
            ajouterProposition(baseFaits, conclusion(r))
            nouveauFaitAjoute <- VRAI
          finSi
        finSi
        rule <- suivant(rule)
      FAIT
    Tant que(nouveauFaitAjoute)
FIN
```

```
Fonction toutesPremissesVraies (r: Regle, baseFaits: Premisse): Booleen
Début
    Si estVide(premisses(r)) alors :
      toutesPremissesVraies <- VRAI
    finSi
    p: Proposition <- tete(premisse(r))
    Tant que(non estVide(p)) faire :
      Si non propositionDansPremisse(baseFaits, proposition(p)) alors :
        toutesPremissesVraies <- FAUX
      finSi
      p <- suivant(p)
    FAIT
    toutesPremissesVraies <- VRAI
FIN

Fonction creerBaseConnaissances (): BC
Début
    creerBaseConnaissances <- ∅
FIN
```

```
Fonction ajouterRegleBaseConnaissances (base: BC, r: Regle): BC
Début
    newRule: BC <- allocateMemory()          “fonction fictive pour allocation“
    regle(newRule) <- r
    suivant(newRule) <- ∅
    Si estVide(base) alors :
      ajouterRegleBaseConnaissances <- newRule
    Sinon
      tmp: BC <- base
      Tant que(non estVide(suivant(tmp))) faire :
            tmp <- suivant(tmp)
      FAIT
      suivant(tmp) <- newRule
      ajouterRegleBaseConnaissances <- base
    finSi
FIN

Fonction teteBaseConnaissance (baseConnaissance : BC) : Rule
Début
        Si estVide(baseConnaissance) alors :
              empty: Regle <- creerRegle()
              teteBaseConnaissance <- empty
        finSi
        teteBaseConnaissance <- regle(baseConnaissance)
FIN
```

```
Fonction supprimerBaseConnaissance (baseConnaissance : BC) : ne renvoie rien
Début
        actuel: BC <- baseConnaissance
        next: BC
        Tant que(non estVide(actuel)) faire :
                next <- suivant(actuel)
                supprimerRegle(regle(actuel))
                libérer(actuel)
                actuel <- next
        FAIT
        baseConnaissance <- ∅
FIN

Fonction creerPremisse ():Premisse
Début
    p: Premisse
    tete(p) <- ∅
    creerPremisse <- p
FIN

Fonction propositionDansPremisse (p: Premisse, prop: caracteres): Booleen
Début
    actuel : Proposition <- tete(p)
    Tant que(non estVide(actuel)) faire :
      Si proposition(actuel) = prop alors :
        propositionDansPremisse <- VRAI
      finSi
      actuel <- suivant(actuel)
    FAIT
    propositionDansPremisse <- FAUX
FIN
```

```
Fonction supprimerProposition (p : Premisse, prop : caracteres) : pas de renvoi
Début
    actuel: Proposition <- tete(p)
    precedent: Proposition <- ∅
    Tant que(non estVide(actuel)) faire :
        Si proposition(actuel) = prop alors :
            Si estVide(precedent) alors :
                tete(p) <- suivant(actuel)
            Sinon
                suivant(precedent) <- suivant(actuel)
            finSi
            libérer(actuel)
        finSi
        precedent <- actuel
        actuel <- suivant(actuel)
    FAIT
FIN

Fonction premisseVide (p: Premisse): Booleen
Début
    premisseVide <- tete(p) <- ∅
FIN
```

```
Fonction ajouterProposition (p: Premisse, prop: caracteres): pas de renvoi
Début
    newProp: Proposition <- allocateMemory()        “fonction fictive pour allocation“
    proposition(newProp) <- prop
    suivant(newProp) <- ∅
    Si estVide(tete(p)) alors :
        tete(p) <- newProp
    Sinon
        actuel: Proposition <- tete(p)
        Tant que(non estVide(suivant(actuel))) faire :
            actuel <- suivant(actuel)
        FAIT
        suivant(actuel) <- newProp
    finSi
FIN

Fonction accesTetePremisse (p: Premisse): Proposition
Début
    Si non estVide(tete(p)) alors :
        accesTetePremisse <- tete(p)
    Sinon
        propvide: Proposition
        proposition(propvide) <- “ “
        suivant(propvide) <- ∅
        accesTetePremisse <- propvide
    finSi
FIN
```

```
Fonction supprimerPremisse (p: Premisse): pas de renvoi
Début
        actuel: Proposition <- tete(p)
        next: Proposition
        Tant que(non estVide(actuel)) faire :
                next <- suivant(actuel)
                libérer(actuel)
                actuel <- next
        FAIT
        tete(p) <- ∅
FIN

Fonction creerRegle (): Rule
Début
        r: Rule
        premisse(r) <- allocateMemory()          “fonction fictive pour allocation“
        premisse(r) <- creerPremisse()
        conclusion(r) <- " "
        creerRegle <- r
FIN

Fonction supprimerRegle (r: Rule): pas de renvoi
Début
        Si non estVide(premisse(r)) alors :
                supprimerPremisse(premisse(r))
                libérer(premisse(r))
                premisse(r) <- ∅
        finSi
        conclusion(r) <- ""
FIN
```

```
Fonction ajouterPropositiondansPremisse (r: Rule, prop: caracteres): pas de renvoi
Début
        newProp: Proposition <- allocateMemory()        “fonction fictive pour allocation“
        proposition(newProp) <- prop
        suivant(newProp) <- ∅
        Si estVide(tete(premisse(p))) alors :
                tete(premisse(p)) <- newProp
        Sinon
                actuel: Proposition <- tete(premisse(p))
                Tant que(non estVide(suivant(actuel))) faire :
                        actuel <- suivant(actuel)
                FAIT
                suivant(actuel) <- newProp
        finSi
FIN

Fonction creerConclusion (r: Rule, prop: caracteres): pas de renvoi
Début
        conclusion(r) <- prop
FIN

Fonction accesConclusion (r: Rule): caracteres
Début
        accesConclusion <- conclusion(r)
FIN
```

## Jeux d'essais :

D'abord on lance le programme pour voir ceci.

```
Welcome to the Inference Engine
1. Add rule to knowledge base
2. Add fact to fact base
3. Deduce new facts
4. Show rules
5. Show facts
6. Delete rules
7. Delete facts
8. Exit
Choose an option:
```

Ensuite on choisit l'option 1 afin de remplir notre base de connaissance de règle ici on donne deux conditions.

```
Welcome to the Inference Engine
1. Add rule to knowledge base
2. Add fact to fact base
3. Deduce new facts
4. Show rules
5. Show facts
6. Delete rules
7. Delete facts
8. Exit
Choose an option: 1
Enter premise: chocolat
Do you want to add another premise? (y/n): y
Enter premise: caramel
Do you want to add another premise? (y/n): n
Enter conclusion: melange_gourmand
```

Et on vérifie notre ensemble de règle pour vérifie qu'elle y a bien été rajouté avec l'option 4.

```
Welcome to the Inference Engine
1. Add rule to knowledge base
2. Add fact to fact base
3. Deduce new facts
4. Show rules
5. Show facts
6. Delete rules
7. Delete facts
8. Exit
Choose an option: 4
Rules:
IF chocolat AND caramel THEN melange_gourmand
IF boue AND caramel THEN melange_etrange
```

Puis maintenant testons l'une de nos règles en rajoutant les critères du mélange étrange à nos connaissance avec l'option 2 tout en vérifiant qu'ils sont bien enregistrés avec l'option 5.

```
Welcome to the Inference Engine
1. Add rule to knowledge base
2. Add fact to fact base
3. Deduce new facts
4. Show rules
5. Show facts
6. Delete rules
7. Delete facts
8. Exit
Choose an option: 2
Enter fact: caramel

Welcome to the Inference Engine
1. Add rule to knowledge base
2. Add fact to fact base
3. Deduce new facts
4. Show rules
5. Show facts
6. Delete rules
7. Delete facts
8. Exit
Choose an option: 2
Enter fact: boue
```

```
Welcome to the Inference Engine
1. Add rule to knowledge base
2. Add fact to fact base
3. Deduce new facts
4. Show rules
5. Show facts
6. Delete rules
7. Delete facts
8. Exit
Choose an option: 5
Facts:
- caramel
- boue
```

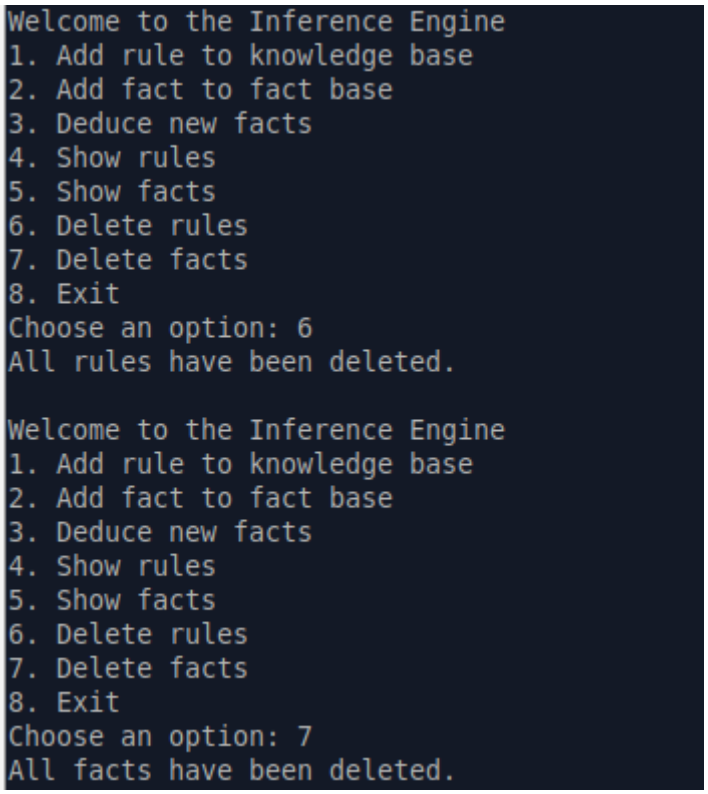
L’option 3 nous permet de déduire les conclusions de nos faits par rapport à nos règles dans notre basse de connaissance.

```
Welcome to the Inference Engine
1. Add rule to knowledge base
2. Add fact to fact base
3. Deduce new facts
4. Show rules
5. Show facts
6. Delete rules
7. Delete facts
8. Exit
Choose an option: 3
Inference completed.
```

Et ici on peut voir grâce à l’option 5 que notre conclusion fait désormais partie de nos faits.

```
Welcome to the Inference Engine
1. Add rule to knowledge base
2. Add fact to fact base
3. Deduce new facts
4. Show rules
5. Show facts
6. Delete rules
7. Delete facts
8. Exit
Choose an option: 5
Facts:
- caramel
- boue
- melange_etrange
```

Puis maintenant à l'aide des option 6 et 7 on peut vider notre base de connaissance de ses faits et règles respectivement.




## Commentaires sur les résultats :

Après plusieurs mises en situation, le programme semble bien fonctionner mais malgré tout plusieurs améliorations peuvent y être apportées, comme la gestion des textes avec espace ou encore des interactions plus aisées entre l'utilisateur et le menu.