

Compilateur pour le langage COREC

Projet de Compilation 2024-25
Master Informatique
Université de Strasbourg





Le langage COREC (*COmputed RECurrences*)

- langage “métier” (*Domain Specific Language, DSL*)
- pour faciliter la programmation de calculs sur des tableaux multi-dimensionnels
- où les calculs sont définis par des relations de récurrence





Structure d'un programme en COREC

```
prog MonProg {  
    def Fonc1 { ...  
    }  
  
    def Fonc2 { ...  
    }  
  
    ...  
  
    def Main { ...  
    }  
}
```



Les fonctions en COREC

```
def Fonc {  
  /* arguments */  
  
  ...  
  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

Les arguments de fonction

- arguments d'entrée :

in: *liste_argts*

- arguments de sortie :

out: *liste_argts*

- arguments d'entrée/sortie :

inout: *liste_argts*

```
def Fonc {  
  /* arguments */  
  
  ...  
  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

Les arguments de fonction

- tableaux : implicitement de type flottant
- variables simples : entier ou flottant, mais flottants déclarés comme un tableau à 1 dimension de taille 1
- pas d'arguments pour **Main**
- argument de la liste **out**:
 - peut apparaître dans une liste **in**: ou **inout**: d'une autre fonction
 - ne peut pas apparaître dans une liste **out**: d'une autre fonction
 - sa portée est définie par les fonctions appelantes
 - le même nom doit être utilisé par toutes les autres fonctions y ayant accès

```
def Fonc {  
  /* arguments */  
  ...  
  /* variables locales */  
  Loc: ...  
  /* section Dom */  
  Dom: ...  
  /* section Rec */  
  Rec: ...  
}
```

Les arguments de fonction

- une fonction ne peut accéder qu'à ses arguments, à ses variables locales, et aux arguments de sortie des fonctions qu'elle appelle
- une fonction ne peut appeler une autre fonction que si elle peut lui fournir ses arguments **in:** et **inout:**

⇒ une suite d'appels de fonctions doit respecter la chaîne des dépendances entre les fonctions !

- la valeur retournée d'une fonction est l'ensemble de ses arguments de sortie et d'entrée/sortie
- cette valeur ne peut être affectée à une variable simple que si la valeur retournée est unique et simple


```
def Fonc {  
  /* arguments */  
  ...  
  /* variables locales */  
  Loc: ...  
  /* section Dom */  
  Dom: ...  
  /* section Rec */  
  Rec: ...  
}
```

```
prog doitgen {  
  def Inita {  
    [in: NR,NQ,NP; out: (a,NR,NQ,NP)]  
    ... }  
  def Initc4 {  
    [in: NP; out: (c4,NP,NP)]  
    ... }  
  def Eq2 {  
    [in: (c4,NP,NP), NR,NQ,NP;  
    inout: (a,NR,NQ,NP)]  
    ... }
```

```
def Main {  
  ...  
  print(Eq2(Initc4,Inita))  
  ... }  
}
```

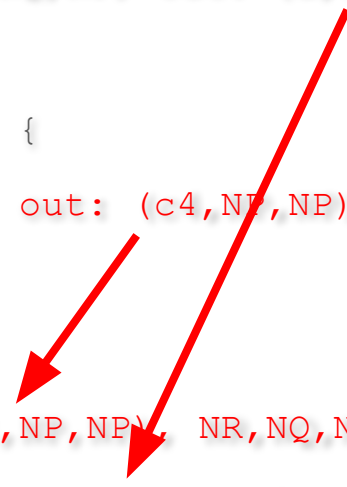


```
prog doitgen {  
  def InitA {  
    [in: NR,NQ,NP; out: (a,NR,NQ,NP)]  
    ... }  
  def Initc4 {  
    [in: NP; out: (c4,NP,NP)]  
    ... }  
  def Eq2 {  
    [in: (c4,NP,NP), NR,NQ,NP;  
    inout: (a,NR,NQ,NP)]  
    ... }  
}
```



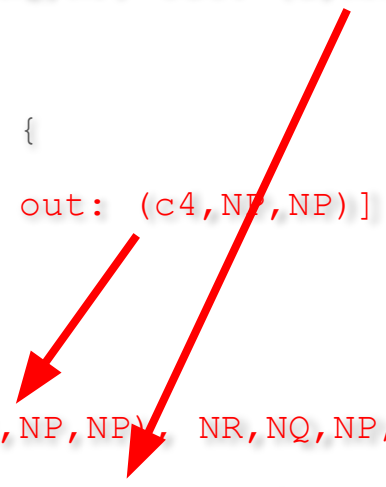
```
def Main {  
  ...  
  print(Eq2(Initc4,InitA))  
  ... }  
}
```

```
prog doitgen {  
  def InitA {  
    [in: NR,NQ,NP; out: (a,NR,NQ,NP)]  
    ... }  
  def Initc4 {  
    [in: NP; out: (c4,NP,NP)]  
    ... }  
  def Eq2 {  
    [in: (c4,NP,NP), NR,NQ,NP;  
     inout: (a,NR,NQ,NP)]  
    ... }  
}
```

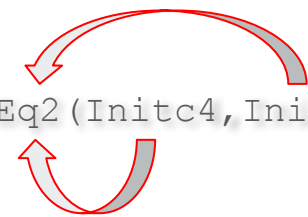


```
def Main {  
  ...  
  print(Eq2(Initc4,InitA))  
  ... }  
}
```

```
prog doitgen {  
  def InitA {  
    [in: NR,NQ,NP; out: (a,NR,NQ,NP)]  
    ... }  
  def Initc4 {  
    [in: NP; out: (c4,NP,NP)]  
    ... }  
  def Eq2 {  
    [in: (c4,NP,NP), NR,NQ,NP;  
    inout: (a,NR,NQ,NP)]  
    ... }  
}
```



```
def Main {  
  ...  
  print(Eq2(Initc4,InitA))  
  ... }  
}
```



Les variables locales

- optionnelles
- tableaux (flottant), variables simples de types entier ou flottant, mais flottants déclarés comme un tableau à 1 dimension de taille 1
- peuvent être référencées comme arguments d'entrée ou d'entrée/sortie des fonctions appelées

```
def Fonc {  
  /* arguments */  
  ...  
  /* variables locales */  
  Loc: ...  
  /* section Dom */  
  Dom: ...  
  /* section Rec */  
  Rec: ...  
}
```

```
prog doitgen {  
  def InitA {  
    [in: NR,NQ,NP; out: (a,NR,NQ,NP)]  
    ... }  
  def Initc4 {  
    [in: NP; out: (c4,NP,NP)]  
    ... }
```

```
def Eq2 {  
  [in: (c4,NP,NP), NR,NQ,NP;  
   inout: (a,NR,NQ,NP)]  
  Loc: r,q,  
        (sum,NP)  
  ... }  
  
def Main {  
  Loc: NR,NQ,NP  
  ...  
  print(Eq2(Initc4,InitA))  
  ... }  
}
```

La section *Dom*

- optionnelle
- définit l'espace de récurrence
 - valeurs des indices pour lesquelles les calculs (définis dans la section *Rec*) doivent être effectués
 - valeurs données à l'aide d'intervalles
- l'ordre de déclaration des intervalles détermine l'ordre d'exécution des calculs
 - exemple :

Dom: i in $[0..NR-1]$, j in $[0..NQ-1]$, k in $[0..NP-1]$

équivalent à :

```
for (i=0; i<NR; i++)  
  for (j=0; j<NQ; j++)  
    for (k=0; k<NP; k++)  
      ...
```

```
def Fonc {  
  /* arguments */  
  
  ...  
  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

La section *Dom*

- bornes dans les intervalles :
constantes entières, variables entières, indices de récurrence définis précédemment, expressions arithmétiques
- Exemple : **Dom:** i in $[0..N-1]$, k in $[0..i-2]$

```
def Fonc {  
  /* arguments */  
  
  ...  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

```

prog doitgen {
    def Inita {
        [in: NR,NQ,NP; out: (a,NR,NQ,NP)]

        Dom: i in [0..NR-1], j in [0..NQ-1],
              k in [0..NP-1]

        ... }

    def Initc4 {
        [in: NP; out: (c4,NP,NP)]

        Dom: i in [0..NP-1], j in [0..NP-1]

        ... }

    def Eq1 { [in: (sum,NP), r,q,NR,NQ,NP;
                inout: (a,NR,NQ,NP)]

        Dom: k in [0..NP-1]

        ... }

```

```

def Eq2 {
    [in: (c4,NP,NP), NR,NQ,NP;
     inout: (a,NR,NQ,NP)]

    Loc: r,q,
          (sum,NP)

    Dom: i in [0..NR-1], j in [0..NQ-1],
          k in [0..NP-1], l in [0..NP-1]

    ... }

def Main {
    Loc: NR,NQ,NP

    ...

    print(Eq2(Initc4,Inita))

    ... }

}

```


La section *Rec*

- instructions de la fonction :
affectations, instructions conditionnelles ou appels de fonction
- les instructions accèdent uniquement aux :
variables locales, arguments de la fonction, arguments de sortie des fonctions appelées précédemment à l'instruction

```
def Fonc {  
  /* arguments */  
  
  ...  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

La section *Rec*

Les appels de fonction :

- écriture fonctionnelle :

$\text{fonc}(\text{fonc}_1, \text{fonc}_2, \dots, \text{fonc}_i(\text{fonc}_{i1}, \text{fonc}_{i2}, \dots), \dots)$

- écriture par références relatives d'appels :

fonc_{i1}
 fonc_{i2}
 $\text{fonc}_1(\%2, \%1)$
 $\text{fonc}(\%1)$

```
def Fonc {  
  /* arguments */  
  
  ...  
  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

La section *Rec*

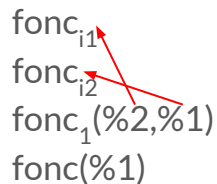
Les appels de fonction :

- écriture fonctionnelle :

$\text{fonc}(\text{fonc}_1, \text{fonc}_2, \dots, \text{fonc}_i(\text{fonc}_{i1}, \text{fonc}_{i2}, \dots), \dots)$

- écriture par références relatives d'appels :

fonc_{i1}
 fonc_{i2}
 $\text{fonc}_1(\%2, \%1)$
 $\text{fonc}(\%1)$



```
def Fonc {  
  /* arguments */  
  
  ...  
  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

La section *Rec*

Les appels de fonction :

- écriture fonctionnelle :
 $\text{fonc}(\text{fonc}_1, \text{fonc}_2, \dots, \text{fonc}_i(\text{fonc}_{i1}, \text{fonc}_{i2}, \dots), \dots)$
- écriture par références relatives d'appels :
 fonc_{i1}
 fonc_{i2}
 $\text{fonc}_1(\%2, \%1)$
 $\text{fonc}(\%1)$
- écriture combinée :
 fonc_{i1}
 fonc_{i2}
 $\text{fonc}(\text{fonc}_1(\%2, \%1))$

```
def Fonc {  
  /* arguments */  
  
  ...  
  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

La section *Rec*

Les appels de fonction :

- l'ordre des appels doit respecter les dépendances de données entre les appels
- cas d'une écriture fonctionnelle : l'appelant n'a accès qu'aux sorties de la fonction appelée en dernier
- cas d'appels successifs par références relatives : chaque sortie des fonctions appelées est accessible à l'appelant
- un appel de fonction peut être affecté à une variable simple que si la fonction possède un seul argument de sortie de type simple
- les fonctions récursives sont possibles (voir exemples)

```
def Fonc {  
  /* arguments */  
  
  ...  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

La section *Rec*

Les appels de fonction :

- 3 fonctions prédéfinies :
 - **read** : saisie au clavier d'une variable simple (entier ou flottant)
 - **print** : affichage à l'écran d'une variable simple, d'un tableau, ou de sorties de fonction
 - **printstr** : affichage d'une chaîne de caractères

```
def Fonc {  
  /* arguments */  
  
  ...  
  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

La section *Rec*

Les instructions conditionnelles :

condition?instructions_si_vrai:instructions_si_faux

- la condition se limite à une expression de comparaison de 2 opérandes avec un opérateur relationnel
- les opérandes peuvent être des expressions arithmétiques, des constantes, des variables simples ou des éléments de tableau
- pas d'opérateurs logiques (et, ou, ...)

```
def Fonc {  
  /* arguments */  
  
  ...  
  
  /* variables locales */  
  
  Loc: ...  
  
  /* section Dom */  
  
  Dom: ...  
  
  /* section Rec */  
  
  Rec: ...  
  
}
```

```

prog doitgen {
  def Inita { [in: NR,NQ,NP;
              out: (a,NR,NQ,NP)]
    Dom: i in [0..NR-1], j in [0..NQ-1],
          k in [0..NP-1]
    Rec: a[i,j,k]=((i*j + k) % NP) / NP
  }
  def Initc4 { [in: NP; out: (c4,NP,NP)]
    Dom: i in [0..NP-1], j in [0..NP-1]
    Rec: c4[i,j]=(i*j % NP) / NP
  }
  def Eq1 { [in: (sum,NP),r,q,NR,NQ,NP;
            inout: (a,NR,NQ,NP)]
    Dom: k in [0..NP-1]
    Rec: a[r,q,k] = sum[k]
  }

```

```

def Eq2 { [in: (c4,NP,NP), NR,NQ,NP;
          inout: (a,NR,NQ,NP)]
  Loc: r,q,
        (sum,NP)
  Dom: i in [0..NR-1], j in [0..NQ-1],
        k in [0..NP-1], l in [0..NP-1]
  Rec: {
    l==0?sum[k]=0: ;
    sum[k] += a[i,j,l] * c4[l,k];
    k==NP-1?l==NP-1?
    { r=i;
      q=j;
      Eq1 }::
  }

```



```
def Main {  
    Loc: NR,NQ,NP  
    Rec: {  
        printstr("Entrez une valeur entière NR : ");  
        read(NR);  
        printstr("Entrez une valeur entière NQ : ");  
        read(NQ);  
        printstr("Entrez une valeur entière NP : ");  
        read(NP);  
        printstr("Le résultat est : ");  
        print(Eq2(Initc4,Inita))  
    }  
}  
}
```



La détection des erreurs

- Importante, avec affichage de messages les plus explicites possibles :
 - dépendances entre fonctions non respectées
 - utilisation d'indices non définis dans les intervalles de la section Dom
 - utilisation de variables ou de tableaux inaccessibles par la fonction courante
 - utilisations de mêmes noms de variables ou de tableaux dans plusieurs arguments de sortie de fonctions différentes, ou dans des variables locales et des arguments de la même fonction.





Réalisation du projet

- Équipes de 4 étudiant.e.s
- Génération de code assembleur **MIPS** ou **RISC-V** (au choix)
- Le compilateur réalisé doit générer un assembleur correct, en toute circonstance, même s'il implémente seulement un petit sous-ensemble du langage COREC
- La fonction **print** doit être implémentée impérativement, pour pouvoir vérifier les résultats de vos programmes
- Rapport + codes sources à déposer sur moodle au plus tard le 12/01/2025
- Évaluation du projet dans la journée du 13/01/2025

Bon travail à tou.te.s !

