الجمهوريــة الجزائريــة الديمقراطيــة الشـعبية
République Algérienne Démocratique et Populaire
وزارة التعليــم العــالي والبحـث العلـمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
Ecole nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique

Projet de Compilation

2 ème année Cycle Supérieur (2CS)

Option : Systèmes Informatiques et Logiciels (SL1)

Livrable 5 : Analyse Sémantique

Membres de l'équipe 6:

- Sefardjelah Serine
- Moulfi Habiba
- Laissaoui Hadjer
- Hathat Meriem

2023 - 2024

Table de matiere

I. Table des symboles

La table des symboles est une structure de données cruciale dans le domaine de la programmation, servant de réceptacle aux informations vitales de toutes les entités d'un programme. Fruit de l'analyse sémantique et syntaxique, elle offre un aperçu organisé des symboles tels que les variables, les fonctions, et autres composants, jouant un rôle fondamental dans la cohérence et la compréhension du code source. Cette structure, au cœur des processus d'analyse, est essentielle pour garantir la validité et la cohésion d'un programme informatique. Explorons de plus près son architecture et son rôle central dans le panorama de la gestion des symboles en programmation.

I.1 Structure NodeSymbol :

- int tokenType : Un entier représentant le type du symbole. Les types possibles sont définis par les constantes TYPE_BOOLEAN, TYPE_INTEGER, TYPE_FLOAT, et TYPE_STRING.
- char symbolName[50] : Un tableau de caractères représentant le nom du symbole, avec une limite de 50 caractères. C'est le nom unique qui identifie le symbole.
- char tokenValue[50] : Un tableau de caractères représentant la valeur du symbole, avec une limite de 50 caractères. Cette valeur peut être vide si le symbole n'a pas encore été initialisé.
- bool isConstant : Un booléen indiquant si le symbole est une constante (true) ou une variable modifiable (false).
- bool hasBeenInitialized : Un booléen indiquant si le symbole a été initialisé avec une valeur (true) ou non (false).
- struct NodeSymbol *next : Un pointeur vers le symbole suivant dans la table. Permet de créer une liste chaînée pour gérer plusieurs symboles.
- struct NodeSymbol *previous : Un pointeur vers le symbole précédent dans la table. Facilite les opérations d'insertion et de suppression.

I.2 Structure SymboleTable :

- NodeSymbol *first : Un pointeur vers le premier symbole dans la table. Permet d'accéder rapidement au début de la liste chaînée.
- NodeSymbol *last : Un pointeur vers le dernier symbole dans la table. Facilite l'ajout efficace de nouveaux symboles à la fin de la liste.

I.3 Macros (Constants) :

- TYPE_BOOLEAN, TYPE_INTEGER, TYPE_FLOAT, TYPE_STRING : Constantes qui définissent les différents types de symboles pouvant être présents dans la table.

I.4. Fonctions associées :

- allocateSymboleTable() : Alloue dynamiquement de la mémoire pour une nouvelle table de symboles, initialise ses pointeurs à NULL, et la renvoie.
- InsertEntry() : Insère un nouveau symbole dans la table avec les détails spécifiés. Gère les liens entre les symboles.
- search() : Effectue une recherche naïve pour trouver un symbole par son nom dans la table. Renvoie un pointeur vers le symbole s'il est trouvé, sinon renvoie NULL.
- displaySymbolTable() : Affiche de manière lisible à l'écran les détails de tous les symboles présents dans la table.
- getName(), getValue(), getType() : Fonctions utilitaires pour récupérer respectivement le nom, la valeur et le type d'un symbole.
- getRealType() : Convertit un type numérique en son équivalent en chaîne de caractères (par exemple, TYPE_INTEGER devient "integer").
- setValue() : Modifie la valeur d'un symbole et marque qu'il a été initialisé. Gère également les contraintes telles que l'impossibilité de réassigner une valeur à une constante.

II. Analyse Sémantique

II.1. Typage des non-terminaux

Afin de stocker et d'accéder aux diverses données requises à l'intérieur des routines sémantiques, la nécessité de typer certains non-terminaux se fait sentir. Par défaut, un non-terminal hérite de la valeur de %union. Ainsi, nous procéderons à la déclaration de structures distinctes pour chaque non-terminal en fonction de leurs besoins spécifiques.

II.1.1 SimpleType :

Le non-terminal "SimpleType" englobe les quatre types primitifs de notre langage, à savoir : entier, flottant, chaîne de caractères et booléen. La représentation de ces types se fait à travers des entiers distincts (0 : Booléen, 1 : Entier, ...etc). Par conséquent, son type intrinsèque est entier

II.1.2 Expression :

L'expression peut revêtir diverses formes telles qu'un booléen, une valeur entière, un flottant, ou même une chaîne de caractères. Afin de simplifier la manipulation de ces diverses formes,

nous attribuons un champ distinct à chaque type, accompagné d'un cinquième champ spécifiant le type global de l'expression.

II.1.3. Déclaration :

La déclaration peut être suivie d'une instanciation. À cet effet, le type de la déclaration sera "Symbole". Cette approche facilite l'assignation efficace de la valeur instanciée au symbole sans nécessiter une recherche supplémentaire dans la table des symboles.

II.1.4. Variable :

Une variable peut représenter l'identifiant d'une variable déjà déclarée ou un élément d'un tableau. Étant donné qu'un tableau est déjà représenté par un symbole. La solution adoptée consiste à créer une structure qui contiendra le pointeur du symbole.

III. Traduction des structures de contrôle :

Boucle While : Trois routines ont été intégrées au niveau des règles de production qui génèrent la boucle while.
Routine ConditionWhile : À ce stade, la routine se contente de sauvegarder l'adresse du quadruplet résultant de l'évaluation de la condition de la boucle while. Cela permet ultérieurement de réaliser un saut vers la fin de l'itération.

```
ConditionWhile:
    TOK_WHILE TOK_PAR_OUV {
    ajouter(StockSauv,qc);
}
;
```

Routine DébutWhile : Dans cette étape, un quadruplet de branchement est inséré en direction de la fin du while si la condition d'arrêt se révèle fausse. De plus, l'adresse (numéro) correspondante est sauvegardée. En cas de non-conformité du type de l'expression de condition (qui devrait normalement être booléen), une erreur sémantique est affichée.

```
DebutWhile :
    ConditionWhile Expression TOK_PAR_FER  TOK_ACC_OUV {
```

```
    if($2.type == TYPE_BOOLEAN){
        char r[10];
        sprintf(r,"R%d",qc-1);
        Quadruplet =  addQuad (Quadruplet,"BZ","tmp","",r,qc);
        ajouter(StockSauv,qc);
        qc++;
    }else{
        yyerrorSemantic( "Non boolean expression found");
    }
};
```

Routine FinWhile :À ce niveau, l'adresse de branchement vers la fin du while dans le quadruplet correspondant est mise à jour. En parallèle, un nouveau quadruplet est inséré pour réaliser un saut inconditionnel vers le quadruplet d'évaluation de la condition.

```
While:
    DebutWhile Bloc TOK_ACC_FER {
    char adresse[10];
    char adresseCondWhile [10];
    int sauvAdrDebutWhile = supprimer(StockSauv);
    int sauvAdrCondWhile = supprimer(StockSauv);
    sprintf(adresseCondWhile,"%d",sauvAdrCondWhile);
    Quadruplet =  addQuad (Quadruplet,"BR",adresseCondWhile,"","",qc);
    hola();
    qc++;
    sprintf(adresse,"%d",qc);
    Quadruplet = updateQuad(Quadruplet ,qc, adresse);
};
```

IV. Jeu d’essai

Input

```
int a = 2;
while( true ){
    a--;
};
str marmar = "sssssssssss" + "mmmm" ;
bool m = false and false ;
bool r = (5 >= 6) ;
bool c = (5 != 7) ;
bool d = (5 > 7) ;
// Test addition
int a = 5;
int b = 3;
int c = 5 + 3;  // c should be 8

// Test subtraction
int d = 10;
int e = 4;
int f = 10 - 4;  // f should be 6

int i = 2 * 5;  // i should be 10



int l = 15 / 3;  // l should be 5



int o = 17 % 4;  // o should be 1


str r = "Hello, " + "world!";  // r should be "Hello, world!"

bool u = true and false;


bool x = true or false;  // x should be true

bool z = not true;  // z should be false
```

Output

```
|Symbol Name          |Token Type|Token Value          |Initialized    |Constant  |
|a                    |integer   |1                    |true           |false     |
|marmar               |String    |"sssssssssss""mmmm"|true           |false     |
|m                    |Boolean   |false                |true           |false     |
|r                    |Boolean   |false                |true           |false     |
|c                    |Boolean   |true                 |true           |false     |
|d                    |Boolean   |false                |true           |false     |
|b                    |integer   |3                    |true           |false     |
|e                    |integer   |4                    |true           |false     |
|f                    |integer   |6                    |true           |false     |
|i                    |integer   |10                   |true           |false     |
|l                    |integer   |5                    |true           |false     |
|o                    |integer   |1                    |true           |false     |
|u                    |Boolean   |false                |true           |false     |
|x                    |Boolean   |true                 |true           |false     |
|z                    |Boolean   |false                |true           |false     |

========================= Affichage des quads ========================
------------------------------------------------------------------------
| Quad No | Operateur | Operande1 | Operande2 | Resultat |
------------------------------------------------------------------------
        Quad[34]=[ := , R33 ,  , z ]
        Quad[33]=[ NOT , true ,  , R33 ]
        Quad[32]=[ := , R31 ,  , x ]
        Quad[31]=[ OR , true , false , R31 ]
        Quad[30]=[ := , R29 ,  , u ]
        Quad[29]=[ AND , true , false , R29 ]
        Quad[28]=[ + , "Hello, " , "world!" , R28 ]
        Quad[27]=[ := , R26 ,  , o ]
        Quad[26]=[ - , 17 , R25 , R26 ]
        Quad[25]=[ * , R24 , 4 , R25 ]
        Quad[24]=[ / , 17 , 4 , R24 ]
        Quad[23]=[ := , R22 ,  , l ]
        Quad[22]=[ / , 15 , 3 , R22 ]
        Quad[21]=[ := , R20 ,  , i ]
        Quad[20]=[ * , 2 , 5 , R20 ]
        Quad[19]=[ := , R18 ,  , f ]
        Quad[18]=[ - , 10 , 4 , R18 ]
        Quad[17]=[ := , 4 ,  , e ]
        Quad[16]=[ + , 5 , 3 , R16 ]
        Quad[15]=[ := , 3 ,  , b ]
        Quad[14]=[ := , R13 ,  , d ]
        Quad[13]=[ > , 5 , 7 , R13 ]
        Quad[12]=[ := , R11 ,  , c ]
        Quad[11]=[ != , 5 , 7 , R11 ]
        Quad[10]=[ := , R9 ,  , r ]
        Quad[9]=[ >= , 5 , 6 , R9 ]
```