Techniques de Compilation
Projet

David Odin

1 Présentation

Le projet que nous vous proposons consiste à créer un compilateur depuis un langage assez simple vers un langage encore plus simple. Vous devrez commencer par comprendre un mini compilateur fonctionnel, le commenter, l'enrichir d'un certain nombre d'extensions, et faire un rapport (qui devra impérativement être rendu en PDF) sur ce que vous aurez fait.

Les langages d'entrée et de sortie sont imposés (aux extensions près). Vous trouverez en annexe les sources du mini compilateur qui pourra vous servir de base.

1.1 Langage d'entrée

Le langage d'entrée de votre compilateur sera une petite partie du langage C ou presque :
- seules les variables entières et les constantes entières sont supportées,
- les variables sont forcément formées d'une seule lettre en minuscule (on a donc au maximum 26 variables),
- les variables sont automatiquement créées lors de leur première utilisation,
- les opérateurs $+$, $-$, $*$, $/$, $<$ et $>$ sont supportés et ont le même sens qu'en C,
- les instructions supportées sont : l'instruction vide, l'affectation, l'affichage, les suites d'instructions et la construction **while**,1
- le programme source est juste une suite d'instructions, sans marque de début ou de fin de fonction.

1.2 Langage de sortie

Le langage de sortie choisi est du code à trois adresses compatible (ou presque) avec du C.
Les avantages de ce langage sont
- la possibilité d'utiliser un compilateur C pour vérifier la validité du code de sortie (au prix de quelques modifications),
- la simplicité,
- la possibilité de transformer très facilement en différents langages d'assembleur.

Ce langage accepte les instructions suivantes (et uniquement elles !) :
- variable = constante ;
- variable = registre ;
- registre = (constante | variable | registre) Op (constante | variable | registre)
- printf("%d", (constant | variable | registre)) ;
- Etiquette :
- goto Etiquette :
- if (registre == 0) goto Etiquette ;

2 Extensions possibles

Afin d'étoffer votre compilateur, vous devrez implémenter un certain nombre d'extensions. Vous pouvez choisir parmi les suivantes ou nous en proposer d'autres.

Extensions	**Difficulté**
Support du moins unaire.	*
Support des opérateurs <=, >=, ==, ! =.	*
Support des opérateurs ++ et −−.	***
Support du goto et étiquettes dans le langage d'entrée.	**
Support du if.	**
Support de la construction do/while.	**
Support du for.	**
Support du if/else.	***
Support de nom de variable de plus d'une lettre.	**
Déclaration des variables et registres.	***
Rendre le langage de sortie compilable par un compilateur C.	****
Support d'autres types de variables que int.	****
Support des fonctions et appels de fonctions.	*****
Optimisation du code de sortie.	de *** à *********
Proposer d'autres représentations internes pour le code.	à partir de ***
Proposer d'autres sorties que du code à 3 adresses (asm, arbre, etc.)	à partir de ***

Rassurez-vous, il est évidemment très complexe d'implémenter tout cela. Choisissez simplement les extensions qui vous paraissent faisable selon votre niveau. Il est préférable d'avoir des extensions simples bien commentées et expliquées dans le rapport que des extensions très complexes non terminées, non commentées ou non décrites.

3 Exemple

Par exemple, si vous avez implémenté, en plus du minimum, la gestion des **if**, des **if/else** et du **moins unaire**, le programme d'entrée suivant :

```
1  a = 1;
2  if (a > 13)
3    {
4      b = 0;
5      while (b < 5)
6        {
7          print b;
8          b = b + a;
9        }
10   }
11 else
12   {
13     c = -(4 + 3 * 7);
14   }
```

Votre compilateur affichera le programme suivant en sortie.

```
1    a = 1;
2    D.2 = a > 13;
3    if (D.2 == 0) goto E0;
4    b = 0;
5  E2:
6    D.4 = b < 5;
7    if (D.4 == 0) goto E3;
8    printf("%d\n", b);
9    D.6 = b + a;
10   b = D.6;
11   goto E2;
12 E3:
```

```
13    goto E1;
14  E0:
15    D.11 = 3 * 7;
16    D.12 = 4 + D.11;
17    D.13 = -D.12;
18    c = D.13;
19  E1:
```

4 Annexes

4.1 mini-compilo.h

```
1  #ifndef MINI_COMPILO_H
2  #define MINI_COMPILO_H
3
4  typedef enum
5  {
6    TYPE_CONSTANTE,
7    TYPE_VARIABLE,
8    TYPE_OPERATEUR
9  } TypeNoeud;
10
11 // Noeud contenant une constante.
12 typedef struct
13 {
14   int valeur;         // Valeur de la constante.
15 } NoeudConstante;
16
17 // Noeud contenant une variable.
18 typedef struct
19 {
20     int i;            // Numéro de variable (de 0 à 25 pour 'a' à 'z')
21 } NoeudVariable;
22
23 // Noeud contenant un opérateur.
24 typedef struct
25 {
26   int nom;                    // type d'opérateur : +, -, =, while, print, etc.
27   int numero_registre;        //
28   int nb_enfants;             // Nombre d'opérandes pour cet opérateur.
29   struct Noeud *operande[3];  // Des pointeurs vers les operandes de cet opérateur.
30 } NoeudOperateur;
31
32 typedef struct Noeud
33 {
34   TypeNoeud type;             // Le type du noeud.
35   union
36   {
37     NoeudConstante constante; // Noeud contenant une constante.
38     NoeudVariable  variable;  // Noeud contenant une variable.
39     NoeudOperateur operateur; // Noeud contenant un opérateur.
40   };
41 } Noeud;
42
43 void sortie(Noeud *noeud);
```

```
44
45 #endif
```

4.2 mini-compilo-lexer.l

```
1  %option noyywrap
2
3  %{
4    #include <stdlib.h>
5    #include "mini-compilo.h"
6    #include "y.tab.h"
7    void yyerror(const char *);
8  %}
9
10 %%
11
12 [a-z]        { yylval.numvariable = *yytext - 'a'; return VARIABLE; }
13
14 [0-9]+       { yylval.valeur = atoi(yytext); return INTEGER; }
15
16 [-()<>=+*/;{}] { return *yytext; }
17
18 "while"          return WHILE;
19 "print"          return PRINT;
20
21 [ \t\n]+         ;       // On ignore les blancs.
22
23 .                yyerror("Unknown character");
24
25 %%
```

4.3 mini-compilo-grammaire.y

```
1  %{
2    #include <stdio.h>
3    #include <stdlib.h>
4    #include <stdarg.h>
5    #include "mini-compilo.h"
6
7    static Noeud *nouvel_operateur_unaire(int type, Noeud *enfant);
8    static Noeud *nouvel_operateur_binaire(int type, Noeud *enfant_gauche,
9                                           Noeud *enfant_droit);
10   static Noeud *nouvelle_variable(int i);
11   static Noeud *nouvelle_constante(int value);
12   static void libere_noeud(Noeud *noeud);
13
14   int yylex(void);
15   void yyerror(const char *s);
16   static int num_registre;
17 %}
18
19 %union
20 {
21   int     valeur;       // Constante entière.
```

```
22    char    numvariable;  // Numéro de variable (de 0 à 25).
23    Noeud *noeud;         // Noeud général.
24  };
25
26  %token <valeur> INTEGER
27  %token <numvariable> VARIABLE
28  %token WHILE PRINT
29
30  %left '>' '<'
31  %left '+' '-'
32  %left '*' '/'
33
34  %type <noeud> instruction exp suite_instructions
35
36  %%
37
38  program: function                { exit(0); }
39          ;
40
41  function: function instruction         { sortie($2); libere_noeud($2); }
42          | /* Rien */
43          ;
44
45  instruction:     ';'              { $$ = nouvel_operateur_binaire(';', NULL, NULL); }
46    | exp ';'                       { $$ = $1; }
47    | PRINT exp ';'                 { $$ = nouvel_operateur_unaire(PRINT, $2); }
48    | VARIABLE '=' exp ';'          { $$ = nouvel_operateur_binaire('=',
49                                                    nouvelle_variable($1), $3); }
50    | WHILE '(' exp ')' instruction { $$ = nouvel_operateur_binaire(WHILE, $3, $5); }
51    | '{' suite_instructions '}'    { $$ = $2; }
52    ;
53
54  suite_instructions: instruction           { $$ = $1; }
55    | suite_instructions instruction { $$ = nouvel_operateur_binaire(';', $1, $2); }
56          ;
57
58  exp:      INTEGER               { $$ = nouvelle_constante($1); }
59          | VARIABLE              { $$ = nouvelle_variable($1); }
60          | exp '+' exp           { $$ = nouvel_operateur_binaire('+', $1, $3); }
61          | exp '-' exp           { $$ = nouvel_operateur_binaire('-', $1, $3); }
62          | exp '*' exp           { $$ = nouvel_operateur_binaire('*', $1, $3); }
63          | exp '/' exp           { $$ = nouvel_operateur_binaire('/', $1, $3); }
64          | exp '<' exp           { $$ = nouvel_operateur_binaire('<', $1, $3); }
65          | exp '>' exp           { $$ = nouvel_operateur_binaire('>', $1, $3); }
66          | '(' exp ')'           { $$ = $2; }
67          ;
68  %%
69
70  static Noeud *nouvelle_constante(int valeur)
71  {
72    Noeud *noeud = malloc(sizeof(Noeud));
73
74    noeud->type = TYPE_CONSTANTE;
75    noeud->constante.valeur = valeur;
76    return noeud;
77  }
```

```
78
79  static Noeud *nouvelle_variable(int i)
80  {
81    Noeud *noeud = malloc(sizeof(Noeud));
82
83    noeud->type = TYPE_VARIABLE;
84    noeud->variable.i = i;
85    return noeud;
86  }
87
88  static Noeud *nouvel_operateur_unaire(int type, Noeud *enfant)
89  {
90    Noeud *noeud = malloc(sizeof(Noeud));
91
92    noeud->type = TYPE_OPERATEUR;
93    noeud->operateur.nom = type;
94    noeud->operateur.numero_registre = ++num_registre;
95    noeud->operateur.nb_enfants = 1;
96    noeud->operateur.operande[0] = enfant;
97
98    return noeud;
99  }
100
101 static Noeud *nouvel_operateur_binaire(int type,
102                                        Noeud *enfant_gauche,
103                                        Noeud *enfant_droit)
104 {
105   Noeud *noeud = malloc(sizeof(Noeud));
106
107   noeud->type = TYPE_OPERATEUR;
108   noeud->operateur.nom = type;
109   noeud->operateur.numero_registre = ++num_registre;
110   noeud->operateur.nb_enfants = 2;
111   noeud->operateur.operande[0] = enfant_gauche;
112   noeud->operateur.operande[1] = enfant_droit;
113
114   return noeud;
115 }
116
117 static void libere_noeud(Noeud *noeud)
118 {
119   int i;
120
121   if (!noeud) return;
122   if (noeud->type == TYPE_OPERATEUR)
123     {
124       for (i = 0; i < noeud->operateur.nb_enfants; i++)
125         libere_noeud(noeud->operateur.operande[i]);
126     }
127   free(noeud);
128 }
129
130 void yyerror(const char *s)
131 {
132   puts(s);
133 }
```

```
134
135 int main(void)
136 {
137   yyparse();
138   return 0;
139 }
```

4.4 mini-compilo-sortie.c

```
1  #include <stdio.h>
2  #include "mini-compilo.h"
3  #include "y.tab.h"
4
5  static int num_etiquette;
6
7  void affiche_nom_noeud(Noeud *noeud)
8  {
9    switch (noeud->type)
10   {
11     case TYPE_CONSTANTE: printf("%d",   noeud->constante.valeur); break;
12     case TYPE_VARIABLE:  printf("%c",   noeud->variable.i + 'a'); break;
13     case TYPE_OPERATEUR: printf("D.%d", noeud->operateur.numero_registre); break;
14   }
15 }
16
17 void sortie(Noeud *noeud)
18 {
19   if (!noeud) return;
20   if (noeud->type == TYPE_CONSTANTE || noeud->type == TYPE_VARIABLE) return;
21
22   switch (noeud->operateur.nom)
23   {
24     case WHILE:
25       {
26         int etiquette1 = num_etiquette, etiquette2 = num_etiquette + 1;
27         num_etiquette += 2;
28         printf("E%d:\n", etiquette1);
29         sortie(noeud->operateur.operande[0]);
30         printf("  if (");
31         affiche_nom_noeud(noeud->operateur.operande[0]);
32         printf(" == 0) goto E%d;\n", etiquette2);
33         sortie(noeud->operateur.operande[1]);
34         printf("  goto E%d;\n", etiquette1);
35         printf("E%d:\n", etiquette2);
36       }
37       break;
38
39     case PRINT:
40       sortie(noeud->operateur.operande[0]);
41       printf("  printf(\"%%d\\n\", ");
42       affiche_nom_noeud(noeud->operateur.operande[0]);
43       printf(");\n");
44       break;
45
46     case '=':
```

```
47        sortie(noeud->operateur.operande[1]);
48        printf("␣␣");
49        affiche_nom_noeud(noeud->operateur.operande[0]);
50        printf("␣=␣");
51        affiche_nom_noeud(noeud->operateur.operande[1]);
52        puts(";");
53        break;
54
55      case ';':
56        sortie(noeud->operateur.operande[0]);
57        sortie(noeud->operateur.operande[1]);
58        break;
59
60      default:
61        sortie(noeud->operateur.operande[0]);
62        sortie(noeud->operateur.operande[1]);
63        printf("␣␣");
64        affiche_nom_noeud(noeud);
65        printf("␣=␣");
66        affiche_nom_noeud(noeud->operateur.operande[0]);
67        switch (noeud->operateur.nom)
68        {
69          case '+':   printf("␣+␣"); break;
70          case '-':   printf("␣-␣"); break;
71          case '*':   printf("␣*␣"); break;
72          case '/':   printf("␣/␣"); break;
73          case '<':   printf("␣<␣"); break;
74          case '>':   printf("␣>␣"); break;
75        }
76        affiche_nom_noeud(noeud->operateur.operande[1]);
77        puts(";");
78    }
79  }
```

4.5 mini-compilo-sortie.c

```
1  CFLAGS = -Wall -Wextra -ggdb3 -Wno-unused-function
2  YFLAGS = -v -d
3
4  OBJECTS = mini-compilo-grammaire.o mini-compilo-lexer.o mini-compilo-sortie.o
5  .INTERMEDIATE: $(OBJECTS)
6
7  mini-compilo: $(OBJECTS)
8          gcc $^ -o $@
9
10 $(OBJECTS): mini-compilo.h
11
12 .PHONY: clean
13 clean:
14         @rm -f mini-compilo *.o mini-compilo-lexer.c mini-compilo-grammaire.c
15         @rm -f y.tab.h y.output
```