Pokročilá tabulka symbolů, dědičné atributy a generování tříadresového kódu

(195.113.207.171:32)

Cíle:

1. Osvojit si práci s dědičnými atributy v syntaxi generátoru *yacc*
2. Seznámit se se základy generování jednoduchého mezikódu

Příklad 1: Typové deklarace a dědičné atributy

Za použití překladačových generátorů vytvořte program, který je na základě následující gramatiky:

Program	**-->**	**declarationlist statementlist**
Statementlist	**-->**	**empty**
Declarationlist	**-->**	**declarationlist declaration | declaration**
declaration	**-->**	**TYPE variablelist ';'**
variablelist	**-->**	**ID | variablelist ',' ID**

a s využitím statické tabulky symbolů schopen přiřadit základní datové typy *int, float* a *char* jednotlivým proměnným ze vstupních deklaračních seznamů a po zpracování celého vstupního kódu vytiskne aktuální tabulku symbolů.
Zajistěte také elementární zotavení po chybě a program opatřete vypovídajícím chybovým hlášením. Pro urychlení práce můžete použít přiložené vzorové soubory, do kterých stačí doplnit jen sémantická pravidla.

Příklad zdrojového textu ze souboru *test1*:

```
int a, b, c;
float d, e, f;
flat chyba;
int g;
float a;
char var, _bx4;
```

Příkazový řádek:

```
> ./cv1 <test1
```

Výstup:

```
Problems encountered: syntax error
Line 3: syntax error.
line 5: Error: a is already defined

** Symbol table **
Dcl("a", "int")
Dcl("b", "int")
Dcl("c", "int")
Dcl("d", "float")
Dcl("e", "float")
Dcl("f", "float")
Dcl("g", "int")
Dcl("var", "char")
Dcl("_bx4", "char")
```

Příklad 2: Generování tříadresového kódu

Z hlediska konstrukce překladačů programovacích jazyků je generování kódu důležitější, než jeho dosud procvičovaná přímá interpretace. Mějme následující soubor instrukcí TAC:

(TAC_ADD, value1, value2, result)	součet
(TAC_SUB, value1, value2, result)	rozdíl
(TAC_DIV, value1, value2, result)	podíl
(TAC_MUL, value1, value2, result)	součin
(TAC_ASS, value, -1, result)	přiřazení (-1 značí prázdnou pozici)
(TAC_PRI, value, -1, -1)	příkaz tisku
(TAC_LBL, value, -1, -1)	generování návěští
(TAC_JZ, value1, value2, -1)	podmíněný skok
(TAC_JMP, value, -1, -1)	nepodmíněný skok

Do gramatiky, kterou najdete v přiloženém *yacc_ovém* souboru, doplňte v souladu se specifikací úlohy odpovídající akce tak, aby se po zpracování vstupního zdrojového kódu vytiskly:

a) Čtveřice, odpovídající instrukcím TAC
b) Instrukce TAC
c) Tabulka symbolů

Na základě rozboru syntaxe sestavte vlastní zdrojový text, realizující:

a) Přiřazení složitějšího aritmetického výrazu do proměnné,
b) Tisk obsahu této proměnné,
c) Analyzujte implementaci cyklu *repeat*, pracujícího s nenulovou hodnotou svého argumentu. Jde o typický příklad realizace překladu jednoduché řídicí struktury.

Příklad zdrojového textu ze souboru *test2:*

```
y := a*(x+64)/(x-c);
x := a+b*a/b+8/(s+t*j);
print x+4;
repeat x {x:=x-1; print x;}
```

(Barvy rozlišují jednotlivé řádky zdrojového kódu)

Příkazový řádek:

```
> ./cv2 <test2
```

Příklad výstupu je na následující straně.

Příklad výstupu:

```
Intermediate code:
Quadruples                          TAC
(TAC_ADD,   2,   3,   4)            _T0 := x + 64
(TAC_MUL,   1,   4,   5)            _T1 := a * _T0
(TAC_SUB,   2,   6,   7)            _T2 := x - c
(TAC_DIV,   5,   7,   8)            _T3 := _T1 / _T2
(TAC_ASS,   8,  -1,   0)            y := _T3

(TAC_MUL,   9,   1,  10)            _T4 := b * a
(TAC_DIV,  10,   9,  11)            _T5 := _T4 / b
(TAC_ADD,   1,  11,  12)            _T6 := a + _T5
(TAC_MUL,  15,  16,  17)            _T7 := t * j
(TAC_ADD,  14,  17,  18)            _T8 := s + _T7
(TAC_DIV,  13,  18,  19)            _T9 := 8 / _T8
(TAC_ADD,  12,  19,  20)            _T10 := _T6 + _T9
(TAC_ASS,  20,  -1,   2)            x := _T10

(TAC_ADD,   2,  21,  22)            _T11 := x + 4
(TAC_PRI,  22,  -1,  -1)            print _T11

(TAC_LBL,  23,  -1,  -1)            label _L0
(TAC_JZ,    2,  24,  -1)            if x is zero, jump to _L1
(TAC_SUB,   2,  25,  26)            _T12 := x - 1
(TAC_ASS,  26,  -1,   2)            x := _T12
(TAC_PRI,   2,  -1,  -1)            print x
(TAC_JMP,  23,  -1,  -1)            jump to _L0
(TAC_LBL,  24,  -1,  -1)            label _L1
```

```
Symbol table:
   0: y
   1: a
   2: x
   3: 64
   4: _T0
   5: _T1
   6: c
   7: _T2
   8: _T3
   9: b
  10: _T4
  11: _T5
  12: _T6
  13: 8
  14: s
  15: t
  16: j
  17: _T7
  18: _T8
  19: _T9
  20: _T10
  21: 4
  22: _T11
  23: _L0
  24: _L1
  25: 1
  26: _T12
```

Poznámky k př 1

```
smrcka@NTB-SMRCKA /tmp/cv10/pr1
$ ./h10_1.exe < test1
Problems encountered: syntax error
Line 3: syntax error.
line 5: Error: a is already defined

** Symbol table **
Dcl("a", "int")
Dcl("b", "int")
Dcl("c", "int")
Dcl("d", "float")
Dcl("e", "float")
Dcl("f", "float")
Dcl("g", "int")
Dcl("var", "char")
Dcl("_bx4", "char")

smrcka@NTB-SMRCKA /tmp/cv10/pr1
$
```

Ty hlášené chyby v pořádku, protože tam jsou.

Př 1

h10_-.h

```
#define NSYMS 20    /* maximum number of symbols */

struct symtab
{
  char *name;
  int type;
} symtab[NSYMS];
/*
struct printparam
{
  int number;
  char *string;
  struct printparam *next;
};
 */
void freebuffer();
```

h10_1.l

```
%{
#include <string.h>
#include "h10_1.tab.h"   /* definitions from YACC */
#define YY_NO_UNPUT
```

```
#define YY_NO_INPUT

int lineno=1;

%}

%%
int             yylval.type = 0; return TYPE;
float                yylval.type = 1; return TYPE;
char            yylval.type = 2; return TYPE;
[_a-zA-Z][_a-zA-Z0-9]*   {
                yylval.name = strdup(yytext);
                return ID;
                }

[ \t]                ;
\n              lineno++;
.               return yytext[0];
%%
```

h10_1.y

```
%{
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "h10.h"

int yylex();
void yyerror (char *s);
void addsymb(int, char *);
void symprint();

extern int lineno;
char *types[3]={"int", "float", "char"};
%}

%token <type> TYPE
%token <name> ID

%union {
  int type;
  char *name;
}

%%

program:        declarationlist statementlist
```

```
        ;

statementlist:   /* empty */
        ;

/* Definition part */

declarationlist: declarationlist declaration
        |                declaration
        ;


declaration:    TYPE variablelist ';'
        |               error ';' {fprintf(stderr, "Line %i: syntax error.\n", lineno);}
        ;

variablelist:   ID {addsymb($<type>0, $1);}
        |               variablelist ',' ID {addsymb($<type>0, $3);}
        ;
            /* při dědění je třeba použít syntaxi $<tokentype>0 */

%%

void addsymb(int type, char *s)
{
 struct symtab *sp;

 for(sp = symtab; sp < &symtab[NSYMS]; sp++)
  {
   /* is it already here? */
   if(sp->name && !strcmp(sp->name, s))
        {
         fprintf(stderr, "line %i: Error: %s is already defined\n", lineno,s);
         return;
        }
   if(!sp->name)
        {
         sp->name = s;
         sp->type = type;
         return ;
        }
  }
 yyerror("Too many symbols");
 exit(1); /* cannot continue */
}

void symprint()
{
 struct symtab *sp;
```

```
  printf("\n** Symbol table **\n");
  for(sp = symtab; sp < &symtab[NSYMS]; sp++)
    if(sp->name)
      printf("Dcl(\"%s\", \"%s\")\n", sp->name, types[sp->type]);
    else
      return;
}

int main()
{
  yydebug = 0;
  yyparse();
  symprint();
  return 0;
}

void yyerror (char *s)
{
  fprintf (stderr, "Problems encountered: %s\n", s);
}
```