**Důležité pro projekt.**

**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**

Finta – dědičný atribut má číslo menší než 1

($<type>0

Nová tabulka symbolů;

dostanou se tam $0,…$-1

Jak probíhá dědění.

vyzkoušet

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:

Daná instrukční sada

jsou to čtvreřice – mezikod

Dříve se hned udělali akce

Nyní z toho vypadně mezikód, sahá do tabulky symbolu

Definuje se struktura čtveřice

Ta se plní podle instrukce

-1 není argument, může být trojice.

generátor musí generovat všechny čtveřice

(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:

1. Čtveřice, odpovídající instrukcím TAC
2. Instrukce TAC
3. Tabulka symbolů

Na základě rozboru syntaxe sestavte vlastní zdrojový text, realizující:

1. Přiřazení složitějšího aritmetického výrazu do proměnné,
2. Tisk obsahu této proměnné,
3. 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:

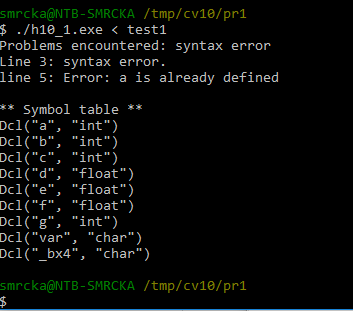
> ./cv2 <test2

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



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);

}

Příklad 2

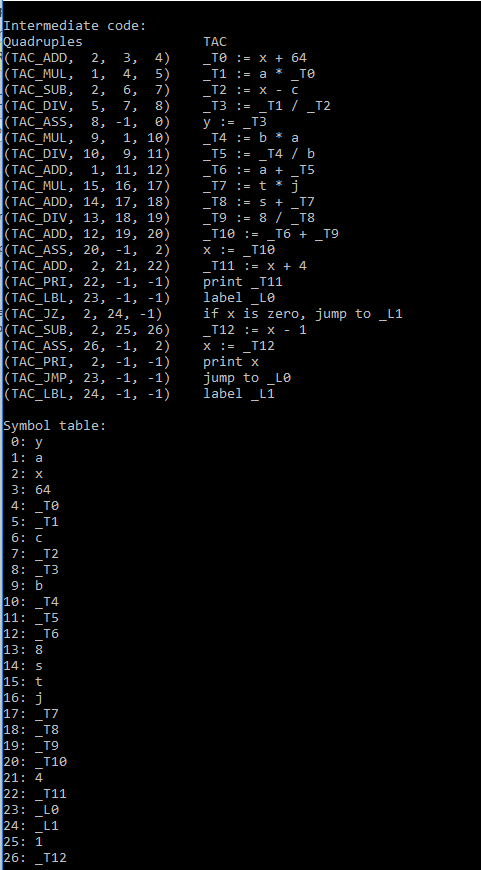
ty téčka jsou dočasné proměnné

Vrcho příkaz repeat – umožňuje statement generovat vícekrát, generuje se statement

processor

jak se generují návěští

labely



h11.h

#ifndef \_H11\_H

#define \_H11\_H

typedef enum

{

TAC\_ADD,

TAC\_SUB,

TAC\_MUL,

TAC\_DIV,

TAC\_ASS,

TAC\_PRI,

TAC\_LBL,

TAC\_JMP,

TAC\_JZ

} tac\_operation;

/\* Function Prototypes \*/

void yyerror(char \*);

int yylex(void);

void makequad(tac\_operation, int, int, int);

int gettemp(void);

int getlabel(void);

void list\_quads(void);

void list\_table(void);

int symlook(char \*);

/\* Definitions \*/

char symtable[64][9]; /\* Symbol table \*/

extern int symcount; /\* Number of symbols in symbol table \*/

extern int quadcount; /\* Number of quadruples\*/

extern int tempcount; /\* Number of temporary variables \*/

extern int labelcount; /\* Number of labels \*/

extern int lineno;

#define MAXQUADS 32 /\* Size of quad array \*/

struct { /\* Array of quadruples \*/

int op; /\* Operator \*/

int o1; /\* 1st operand \*/

int o2; /\* 2nd operand \*/

int o3; /\* Result \*/

} quad[MAXQUADS];

#endif

h11.l

%{

#include <string.h>

#include "h11.tab.h" /\* definitions from YACC \*/

#include "h11.h"

#define symbol\_len 20

#define YY\_NO\_UNPUT

#define YY\_NO\_INPUT

int symcount = 0; /\* The amount of symbols in the symbol table or the index

of the first free place in the symbol table \*/

int lineno=1;

%}

%%

repeat return REPEAT;

print return PRINT;

":=" return ASSIGN;

[\_a-zA-Z][\_a-zA-Z0-9]{0,7} yylval = symlook(yytext); return VAR;

[0-9]+ yylval = symlook(yytext); return NUM;

[ \t] ; /\* ignore white space \*/

\n lineno++;

. return yytext[0];

%%

int symlook(char \*symbol)

{

int i=0;

while(i<symcount && strcmp(symbol, symtable[i]))

i++;

if(i<symcount)

return i;

else

{

strncpy(symtable[symcount], symbol, 8); /\* Add to table \*/

return symcount++;

}

}

**h11.y**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "h11.h"

char taccodes[][8]={"TAC\_ADD", "TAC\_SUB", "TAC\_MUL", "TAC\_DIV", "TAC\_ASS", "TAC\_PRI", "TAC\_LBL", "TAC\_JMP", "TAC\_JZ"};

int quadcount = 0; /\* Number of quadruples\*/

int tempcount = 0; /\* Number of temporary variables \*/

int labelcount = 0; /\* Number of labels \*/

%}

%token VAR NUM ASSIGN PRINT REPEAT

%left '-' '+'

%left '\*' '/'

%%

/\* = = = = = Grammar Section = = = = = = = = = = = = \*/

/\* Productions \*/ /\* Semantic actions \*/

stmts: stmts stmt

| stmt

;

stmt: VAR ASSIGN expr ';' { makequad(TAC\_ASS, $3, -1, $1);}

| PRINT expr ';' { makequad(TAC\_PRI, $2, -1, -1); }

| repeat

| error ';'

;

expr : expr '+' expr { $$=gettemp(); makequad(TAC\_ADD, $1, $3, $$); }

| expr '-' expr { $$=gettemp(); makequad(TAC\_SUB, $1, $3, $$); }

| expr '\*' expr { $$=gettemp(); makequad(TAC\_MUL, $1, $3, $$); }

| expr '/' expr { $$=gettemp(); makequad(TAC\_DIV, $1, $3, $$); }

| '(' expr ')' { $$ = $2; }

| term

;

term : NUM | VAR

;

repeat: REPEAT VAR {

/\* Storing start label index into repeat symbol, which

don't have any other useful value - because of loops

inside loops we cannot use global variables to store

labels, we would need a stack. End label is naturally

startlabel + 1, because it's created after startlabel \*/

$1 = getlabel(); /\* Start label \*/

getlabel(); /\* End label \*/

makequad(TAC\_LBL, $1, -1, -1);

makequad(TAC\_JZ, $2, $1+1, -1);

}

'{' stmts '}' {

makequad(TAC\_JMP, $1, -1, -1);

makequad(TAC\_LBL, $1+1, -1, -1);

}

;

%%

int main()

{

yydebug = 0;

yyparse(); /\* Parse a statement \*/

list\_quads(); /\* Print generated code \*/

list\_table(); /\* Print symbol table \*/

return 0;

}

/\* Supporting Functions \*/

void yyerror(char \*mesg)

{

printf("%s\n", mesg);

}

/\* Assembles quadruple \*/

void makequad(tac\_operation op, int op1, int op2, int res)

{

quad[quadcount].op = op; /\* Operator \*/

quad[quadcount].o1 = op1; /\* Operands \*/

quad[quadcount].o2 = op2;

quad[quadcount].o3 = res; /\* Result \*/

quadcount++;

}

/\* Gets new temporary \*/

int gettemp(void)

{

char str[6];

snprintf(str, 5, "\_T%i", tempcount++); /\* Assemble its name \*/

str[5] = '\0'; /\* Adding the end of the string -mark \*/

strcpy (symtable[symcount], str); /\* Add to the symbol table \*/

return symcount++;

}

/\* Gets new label \*/

int getlabel(void)

{

char str[6];

snprintf(str, 5, "\_L%i", labelcount++); /\* Assemble its name \*/

str[5] = '\0'; /\* Adding the end of the string -mark \*/

strcpy (symtable[symcount], str); /\* Add to the symbol table \*/

return symcount++;

}

/\* Lists intermediate code as quadruples & TAC \*/

void list\_quads(void)

{

int i;

printf ("\nIntermediate code:\n");

printf ("Quadruples\t\t TAC\n");

for ( i = 0; i < quadcount; i++){ /\* List quadruple & interpret it \*/

printf ("(%s, %2d, %2d, %2d)\t ", taccodes[quad[i].op], quad[i].o1, quad[i].o2, quad[i].o3);

switch (quad[i].op)

{

case TAC\_ADD:

printf ("%s := %s + %s\n", symtable[quad[i].o3], symtable[quad[i].o1], symtable[quad[i].o2]);

break;

case TAC\_SUB:

printf ("%s := %s - %s\n", symtable[quad[i].o3], symtable[quad[i].o1], symtable[quad[i].o2]);

break;

case TAC\_MUL:

printf ("%s := %s \* %s\n", symtable[quad[i].o3], symtable[quad[i].o1], symtable[quad[i].o2]);

break;

case TAC\_DIV:

printf ("%s := %s / %s\n", symtable[quad[i].o3], symtable[quad[i].o1], symtable[quad[i].o2]);

break;

case TAC\_ASS:

printf("%s := %s\n", symtable[quad[i].o3], symtable[quad[i].o1]);

break;

case TAC\_PRI:

printf("print %s\n", symtable[quad[i].o1]);

break;

case TAC\_LBL:

printf("label %s\n", symtable[quad[i].o1]);

break;

case TAC\_JMP:

printf("jump to %s\n", symtable[quad[i].o1]);

break;

case TAC\_JZ:

printf("if %s is zero, jump to %s\n", symtable[quad[i].o1], symtable[quad[i].o2]);

break;

default:

printf("Not a valid TAC code!\n");

}

}

}

/\* Lists symbol table contents \*/

void list\_table (void)

{

int i;

printf("\nSymbol table:\n");

for (i = 0; i < symcount; i++)

printf ("%2d: %s\n", i, symtable[i]);

printf ("\n");

}