```c
#include <stdio.h>
#include <ctype.h>
#include "nodes.h"
#include "C.tab.h"
#include <string.h>
#include "interpreter.h"
#include "gentac.h"
#include "genmc.h"

char *named(int t)
{
    static char b[100];
    if (isgraph(t) || t==' ') {
      sprintf(b, "%c", t);
      return b;
    }
    switch (t) {
      default: return "???";
    case IDENTIFIER:
      return "id";
    case CONSTANT:
      return "constant";
    case STRING_LITERAL:
      return "string";
    case LE_OP:
      return "<=";
    case GE_OP:
      return ">=";
    case EQ_OP:
      return "==";
    case NE_OP:
      return "!=";
    case EXTERN:
      return "extern";
    case AUTO:
      return "auto";
    case INT:
      return "int";
    case VOID:
      return "void";
    case APPLY:
      return "apply";
    case LEAF:
      return "leaf";
    case IF:
      return "if";
    case ELSE:
      return "else";
    case WHILE:
      return "while";
    case CONTINUE:
      return "continue";
    case BREAK:
      return "break";
    case RETURN:
      return "return";
    }
}

void print_leaf(NODE *tree, int level)
```

```
 61 {
 62     TOKEN *t = (TOKEN *)tree;
 63     int i;
 64     for (i=0; i<level; i++) putchar(' ');
 65     if (t->type == CONSTANT) printf("%d\n", t->value);
 66     else if (t->type == STRING_LITERAL) printf("\"%s\"\n", t->lexeme);
 67     else if (t) puts(t->lexeme);
 68 }
 69
 70 void print_tree0(NODE *tree, int level)
 71 {
 72     int i;
 73     if (tree==NULL) return;
 74     if (tree->type==LEAF) {
 75       print_leaf(tree->left, level);
 76     }
 77     else {
 78       for(i=0; i<level; i++) putchar(' ');
 79       printf("%s\n",named(tree->type));
 80 /*       if (tree->type=='~') { */
 81 /*         for(i=0; i<level+2; i++) putchar(' '); */
 82 /*         printf("%p\n", tree->left); */
 83 /*       } */
 84 /*       else */
 85         print_tree0(tree->left, level+2);
 86       print_tree0(tree->right, level+2);
 87     }
 88 }
 89
 90 void print_tree(NODE *tree)
 91 {
 92     print_tree0(tree, 0);
 93 }
 94
 95 char* tac_ops[] =
    {"","ADD","SUB","DIV","MOD","MULT","PROC","ENDPROC","LOAD","STORE","IF","LABEL
    ","GOTO","CALL","RETURN","INNER_PROC"};
 96
 97 void print_if(TAC* tac){
 98   if(tac->ift.op1->type == IDENTIFIER && tac->ift.op2->type == IDENTIFIER){
 99     printf("%s (%s%s%s) %s\n",
100     tac_ops[tac->op],
101     tac->ift.op1->lexeme,
102     named(tac->ift.code),
103     tac->ift.op2->lexeme,
104     tac->ift.lbl->lexeme);
105   }
106   else if(tac->ift.op1->type == IDENTIFIER){
107     printf("%s (%s%s%d) %s\n",
108     tac_ops[tac->op],
109     tac->ift.op1->lexeme,
110     named(tac->ift.code),
111     tac->ift.op2->value,
112     tac->ift.lbl->lexeme);
113   }
114   else if(tac->ift.op2->type == IDENTIFIER){
115     printf("%s (%d%s%s) %s\n",
116     tac_ops[tac->op],
117     tac->ift.op1->value,
118     named(tac->ift.code),
```

```
119     tac->ift.op2->lexeme,
120     tac->ift.lbl->lexeme);
121   }
122   else{
123     printf("%s (%d%s%d) %s\n",
124     tac_ops[tac->op],
125     tac->ift.op1->value,
126     named(tac->ift.code),
127     tac->ift.op2->value,
128     tac->ift.lbl->lexeme);
129   }
130
131 }
132
133 void print_rtn(TAC* tac){
134   if(tac->rtn.type == tac_call){
135     printf("%s\n",
136     tac_ops[tac->op]);
137   }
138   else if (tac->rtn.type == CONSTANT){
139     printf("%s %i\n",
140     tac_ops[tac->op],
141     tac->rtn.v->value);
142   }
143   else{
144     printf("%s %s\n",
145     tac_ops[tac->op],
146     tac->rtn.v->lexeme);
147   }
148 }
149
150 void print_ic(TAC* tac){
151
152   while(tac!=NULL){
153     switch(tac->op){
154       default:
155         printf("%s %s %s %s\n",
156         tac_ops[tac->op],
157         tac->stac.src1->lexeme,
158         tac->stac.src2->lexeme,
159         tac->stac.dst->lexeme);
160         break;
161       case tac_load:
162         if(tac->ld.src1->type == CONSTANT){
163           printf("%s %i %s\n",
164           tac_ops[tac->op],
165           tac->ld.src1->value,
166           tac->ld.dst->lexeme);
167         }
168         else{
169           printf("%s %s %s\n",
170           tac_ops[tac->op],
171           tac->ld.src1->lexeme,
172           tac->ld.dst->lexeme);
173         }
174         break;
175       case tac_store:
176         printf("%s %s %s\n",
177         tac_ops[tac->op],
178         tac->ld.src1->lexeme,
```

```
179         tac->ld.dst->lexeme);
180         break;
181       case tac_proc:
182         printf("%s %s %i\n",
183         tac_ops[tac->op],
184         tac->proc.name->lexeme,
185         tac->proc.arity);
186         break;
187       case tac_innerproc:
188         printf("%s %s %i\n",
189         tac_ops[tac->op],
190         tac->proc.name->lexeme,
191         tac->proc.arity);
192         break;
193       case tac_endproc:
194         printf("%s\n",
195         tac_ops[tac->op]);
196         break;
197       case tac_if:
198         print_if(tac);
199         break;
200       case tac_lbl:
201         printf("%s %s\n",
202         tac_ops[tac->op],
203         tac->lbl.name->lexeme);
204         break;
205       case tac_goto:
206         printf("%s %s\n",
207         tac_ops[tac->op],
208         tac->gtl.lbl->lexeme);
209         break;
210       case tac_call:
211         printf("%s %s %i\n",
212         tac_ops[tac->op],
213         tac->call.name->lexeme,
214         tac->call.arity);
215         break;
216       case tac_rtn:
217         print_rtn(tac);
218         break;
219     }
220     tac = tac->next;
221   }
222
223 }
224
225 void print_token(TOKEN *i){
226   if (i->type == CONSTANT){
227     printf("%d",i->value);
228   }
229   else {
230     printf("%s",i->lexeme);
231   }
232 }
233 void print_mc(MC* i)
234 {
235   for(;i!=NULL;i=i->next) printf("%s\n",i->insn);
236 }
237
238 void print_bbs(BB** bbs){
```

```
239   int i = 0;
240   while(bbs[i] != NULL){
241     printf("\033[0;31m");
242     printf("BLOCK #");print_token(bbs[i]->id);
243     printf("\n\033[0m");
244     print_ic(bbs[i]->leader);
245     if(bbs[i]->nexts[0] != NULL){
246       printf("\033[0;31m");
247       printf("LINKS TO : ");
248       print_token(bbs[i]->nexts[0]->id);
249     }
250     if(bbs[i]->nexts[1] != NULL){
251       printf(" ");
252       print_token(bbs[i]->nexts[1]->id);
253     }
254     i++;
255     printf("\n\n");
256   }
257   printf("\033[0m");
258 }
259
260 extern int yydebug;
261 extern NODE* yyparse(void);
262 extern NODE* ans;
263 extern void init_symbtable(void);
264 extern VALUE* interpret(NODE*);
265 extern TAC* gen_tac(NODE*);
266 extern MC* gen_mc(TAC*);
267
268 int main(int argc, char** argv)
269 {
270     NODE* tree;
271     FRAME* e = malloc(sizeof(FRAME));
272     if (argc>1 && strcmp(argv[1],"-d")==0) yydebug = 1;
273     init_symbtable();
274     printf("--C COMPILER\n");
275     yyparse();
276     tree = ans;
277     printf("parse finished with %p\n", tree);
278     print_tree(tree);
279     printf("\n");
280     printf("Calling interpreter...\n");
281     VALUE* result = interpret(tree);
282     if(result != NULL){
283       printf("RESULT : %i\n",result->integer);
284     }
285     else{
286       printf("RESULT: NULL\n");
287     }
288
289
    printf("-----------------------------------------------------------------\n");
290     printf("Generating TAC...\n");
291     TAC* tac = gen_tac(tree);
292     print_ic(tac);
293     //print_bbs(tac);
294     printf("Generating machine code...\n");
295     print_mc(gen_mc(tac));
296     return 0;
297 }
```

298