完整代码

`ast.h`:

```h
#ifndef __AST
#define __AST

#include <stdio.h>

typedef struct _ast ast;
typedef struct _ast *past;
struct _ast{
    int ivalue;
    char* strValue;
    char* nodeType;
    past next;
    past left;
    past right;
};

past newAstNode();
past newNum(int value);
past newExpr(int oper, past left,past right);
past newDoubleExpr(char* logic_oper, past left,past right);
past newBasicNode(char* nodeType, past left, past right, past next);
past newNextNode(char* nodeType, past older, past younger);
past newTypeNode(char* strVal);
past newIDNode(char* strVal);
void showAst(past node, int nest);

#endif

```

`ast.c`:

```c
#include "ast.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
5
6  past newAstNode()
7  {
8      past node = malloc(sizeof(ast));
9      if(node == NULL)
10     {
11         printf("run out of memory.\n");
12         exit(0);
13     }
14     memset(node, 0, sizeof(ast));
15     return node;
16 }
17
18 past newNum(int value)
19 {
20     past var = newAstNode();
21     var->nodeType = "intValue";
22     var->ivalue = value;
23     return var;
24 }
25
26 past newExpr(int oper, past left,past right)
27 {
28     past var = newAstNode();
29     var->nodeType = "expr";
30     var->ivalue = oper;
31     var->left = left;
32     var->right = right;
33     var -> strValue = "@";
34     return var;
35 }
36
37 past newDoubleExpr(char* logic_oper, past left,past right)
38 {
39     past var = newAstNode();
40     var->nodeType = "expr";
41     char *strVal = malloc(sizeof(logic_oper));
42     strcpy(strVal,logic_oper);
43     var -> strValue = strVal;
44     var->left = left;
45     var->right = right;
46     return var;
47 }
48
49 past newBasicNode(char* nodeType, past left, past right, past next)
50 {
51     past root = newAstNode();
```

```
52      char *node_type = malloc(sizeof(nodeType));
53      strcpy(node_type,nodeType);
54      root->nodeType = node_type;
55      root->left = left;
56      root->right = right;
57      root->next = next;
58      return root;
59  }
60
61  past newNextNode(char* nodeType, past older, past younger)
62  {
63      past root = NULL;
64      //还没有根节点
65      if(strcmp(nodeType, older->nodeType) != 0){
66          root = newAstNode();
67          char *node_type = malloc(sizeof(nodeType));
68          strcpy(node_type,nodeType);
69          root->nodeType = node_type;
70          root->left = older; root->left->next = younger;
71          root->ivalue = 1;
72      }
73      //已经有根节点
74      else{
75          root = older;
76          older = older->left;
77          while(older->next != NULL) older = older->next;
78          older->next = younger;
79          root->ivalue++;
80      }
81      return root;
82  }
83
84  past newTypeNode(char* strVal)
85  {
86      past root = newAstNode();
87      root->nodeType = "type";
88      char *buf = malloc(sizeof(strVal));
89      strcpy(buf,strVal);
90      root->strValue = buf;
91      return root;
92  }
93
94  past newIDNode(char* strVal)
95  {
96      past root = newAstNode();
97      root->nodeType = "parameter";
98      char *buf = malloc(sizeof(strVal));
```

```
99      strcpy(buf,strVal);
100     root->strValue = buf;
101     return root;
102 }
103
104 void showAst(past node, int nest)
105 {
106     if(node == NULL) {
107         //printf("node transfer error\n");
108         return;
109     }
110     int i = 0;
111     for(i = 0; i < nest; i ++)
112         printf("  ");
113     if(strcmp(node->nodeType, "expr") == 0){
114         if(strcmp(node->strValue, "@") == 0)
115             printf("%s '%c'\n", node->nodeType, (char)node->ivalue);
116         else
117             printf("%s %s\n", node->nodeType, node->strValue);
118     }
119     else if(strcmp(node->nodeType, "intValue")==0){
120         printf("%s . %d\n", node->nodeType, node->ivalue);
121     }
122     else{
123         if(!node->strValue){
124             if(node->ivalue) printf("%s . %d\n", node->nodeType, node->ivalue);
125             else printf("%s .\n", node->nodeType);
126         }
127         else if(node->ivalue) printf("%s %s %d\n", node->nodeType, node-
    >strValue, node->ivalue);
128         else printf("%s %s .\n", node->nodeType, node->strValue);
129     }
130     showAst(node->left, nest+1);
131     showAst(node->right, nest+1);
132     showAst(node->next,nest);
133 }
134
```

`genllvm.h`:

```
1  #ifndef GENLLVM_H
2  #define GENLLVM_H
3
4  #include "ast.h"
```

```
5
6  enum {T_INT = 1};
7  #define true 1
8  #define false 0
9
10 int genExpr(past node);
11
12 #endif
```

genllvm.c:

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdbool.h>
5  #include "genllvm.h"
6
7  int stack_top = 0;
8  int regCount = 1;
9  int varCount = 0;
10 int reg[40];
11 char *variables[40];
12 char *variable_type[40];
13 int whilecheckpoint[10];
14 int ifcheckpoint[10];
15 int if_stack_top = 0;
16 int process(past node, char *result);
17
18 void addLLVMCodes(char *codes)
19 {
20     printf("%s", codes);
21 }
22 int checkVariable(char *variable_name)
23 {
24     int res = 0;
25     for (int i = 1; i < regCount; i++)
26     {
27         if (strcmp(variable_name, variables[i]) == 0)
28         {
29             res = i;
30         }
31     }
32
33     return res;
```

```
}
int genExprStmt(past node, char *result)
{
    char *key = (char *)malloc(sizeof(char) * 5);
    char *left = (char *)malloc(sizeof(char) * 5);
    char *right = (char *)malloc(sizeof(char) * 5);
    int left_record = 0, right_record = 0;
    if (strcmp(node->strValue, "@") == 0 || strcmp(node->strValue, "==") == 0 || strcmp(node->strValue, "!=") == 0 || strcmp(node->strValue, ">=") == 0 || strcmp(node->strValue, "<=") == 0)
    {
        switch (node->ivalue)
        {
        case '+':
            sprintf(key, "add");
            break;
        case '-':
            sprintf(key, "sub");
            break;
        case '*':
            sprintf(key, "mul");
            break;
        case '/':
            sprintf(key, "div");
            break;
        default:
            sprintf(key, "else");
            break;
        }
        if (strcmp(node->left->nodeType, "intValue") == 0)
        {
            sprintf(left, "%d", node->left->ivalue);
        }
        else if (strcmp(node->left->nodeType, "parameter") == 0)
        {
            left_record = checkVariable(node->left->strValue);
            if (left_record != 0)
            {
                ++varCount;
                sprintf(left, "%%%d", varCount);
                char *tmp = (char *)malloc(sizeof(char) * 200);
                sprintf(tmp, "%%%d = load i32, i32* %%%d, align 4\n", varCount, reg[left_record]);
                result = strcat(result, tmp);
                free(tmp);
            }
        }
```

```
78          else if (strcmp(node->left->nodeType, "expr") == 0)
79          {
80              sprintf(left, "%%%d", genExprStmt(node->left, result));
81          }
82          if (strcmp(node->right->nodeType, "intValue") == 0)
83          {
84              sprintf(right, "%d", node->right->ivalue);
85          }
86          else if (strcmp(node->right->nodeType, "parameter") == 0)
87          {
88              right_record = checkVariable(node->right->strValue);
89              if (right_record != 0)
90              {
91                  ++varCount;
92                  sprintf(right, "%%%d", varCount);
93                  char *tmp = (char *)malloc(sizeof(char) * 200);
94                  sprintf(tmp, "%%%d = load i32, i32* %%%d, align 4\n",
    varCount, reg[right_record]);
95                  result = strcat(result, tmp);
96                  free(tmp);
97              }
98          }
99          else if (strcmp(node->right->nodeType, "expr") == 0)
100         {
101             sprintf(right, "%%%d", genExprStmt(node->right, result));
102         }
103         char *tmp = (char *)malloc(sizeof(char) * 200);
104         if (strcmp(key, "else") == 0)
105         {
106             if (strcmp(node->strValue, "@") == 0 && node->ivalue == '>')
107                 sprintf(tmp, "%%%d = icmp %s i32 %s, %s\n", ++varCount, "sgt",
    left, right);
108             else if (strcmp(node->strValue, "@") == 0 && node->ivalue == '<')
109                 sprintf(tmp, "%%%d = icmp %s i32 %s, %s\n", ++varCount, "slt",
    left, right);
110             else if (strcmp(node->strValue, "==") == 0)
111                 sprintf(tmp, "%%%d = icmp %s i32 %s, %s\n", ++varCount, "eq",
    left, right);
112             else if (strcmp(node->strValue, "!=") == 0)
113                 sprintf(tmp, "%%%d = icmp %s i32 %s, %s\n", ++varCount, "ne",
    left, right);
114             else if (strcmp(node->strValue, ">=") == 0)
115                 sprintf(tmp, "%%%d = icmp %s i32 %s, %s\n", ++varCount, "sge",
    left, right);
116             else if (strcmp(node->strValue, "<=") == 0)
117                 sprintf(tmp, "%%%d = icmp %s i32 %s, %s\n", ++varCount, "sle",
    left, right);
```

```
118             strcat(result, tmp);
119             free(tmp);
120             free(key);
121             free(left);
122             free(right);
123             return 0;
124         }
125         sprintf(tmp, "%%%d = %s i32 %s, %s\n", ++varCount, key, left, right);
126         result = strcat(result, tmp);
127 
128         free(tmp);
129         free(key);
130         free(left);
131         free(right);
132         return varCount;
133     }
134 
135     return 0;
136 }
137 int genDeclStmt(past node, char *result)
138 {
139     if (strcmp(node->left->left->strValue, "int") == 0)
140     {
141         past l = node->left->right->left;
142         for (int i = 0; i < node->left->right->ivalue; i++)
143         {
144             if (strcmp(l->right->nodeType, "intValue") == 0)
145             {
146                 variables[regCount] = l->left->strValue;
147                 variable_type[regCount] = "int";
148                 reg[regCount++] = ++varCount;
149                 char *tmp = malloc(sizeof(char) * 200);
150                 sprintf(tmp, "%c%d = alloca i32, align 4\nstore i32 %d, i32*
    %c%d, align 4\n", '%', varCount, l->right->ivalue, '%',
151                         varCount);
152                 strcat(result, tmp);
153                 free(tmp);
154             }
155             l = l->next;
156         }
157         return 0;
158     }
159 }
160 int genAssignStmt(past node, char *result)
161 {
162     char *tmp = malloc(sizeof(char) * 200);
163     if (strcmp(node->right->nodeType, "expr") == 0)
```

```
164     {
165         int pos = checkVariable(node->left->strValue);
166         sprintf(tmp, "store i32 %c%d , i32* %c%d, align 4\n", '%',
    genExprStmt(node->right, result), '%', reg[pos]);
167         result = strcat(result, tmp);
168         free(tmp);
169         return 0;
170     }
171     else if (strcmp(node->right->nodeType, "intValue") == 0)
172     {
173         int pos = checkVariable(node->left->strValue);
174         sprintf(tmp, "store i32 %d , i32* %c%d, align 4\n", node->right-
    >ivalue, '%', reg[pos]);
175         result = strcat(result, tmp);
176         free(tmp);
177         return 0;
178     }
179     else if (strcmp(node->right->nodeType, "parameter") == 0)
180     {
181         char *tmp1 = malloc(sizeof(char) * 200);
182         sprintf(tmp1, "%%%d = load i32, i32* %%%d, align 4\n", ++varCount,
    reg[checkVariable(node->right->strValue)]);
183         result = strcat(result, tmp1);
184         free(tmp1);
185         int pos = checkVariable(node->left->strValue);
186         sprintf(tmp, "store i32 %c%d , i32* %c%d, align 4\n", '%', varCount,
    '%', reg[pos]);
187         result = strcat(result, tmp);
188         free(tmp);
189         return 0;
190     }
191     else
192     {
193         free(tmp);
194         result = "";
195         return 0;
196     }
197 }
198 int genIfStmt(past node, char *result)
199 {
200     char *res = (char *)malloc(200);
201     char *tmp = (char *)malloc(200);
202     genExprStmt(node->left, result);
203     char *tmp2 = (char *)malloc(sizeof(char) * 200);
204     sprintf(tmp2, "br i1 %c%d, label %c%d, label %c%d\n; <label>:%d:\n", '%',
    varCount, '%', varCount + 1, '%', varCount + 2, varCount + 1);
205     varCount++;
```

```
206     result = strcat(result, tmp2);
207     process(node->right, result);
208     char *tmp3 = (char *)malloc(sizeof(char) * 200);
209     ++varCount;
210     sprintf(tmp3, "br label %c%d\n; <label>:%d:\n", '%', varCount, varCount);
211     result = strcat(result, tmp3);
212     free(tmp);
213     free(res);
214     return 0; // res
215 }
216 int genIfElseStmt(past node, char *result)
217 {
218     char *res = (char *)malloc(200);
219     char *tmp = (char *)malloc(200);
220     genExprStmt(node->left, result);
221     ifcheckpoint[if_stack_top++] = varCount;
222     char *tmp2 = (char *)malloc(sizeof(char) * 200);
223     sprintf(tmp2, "br i1 %c%d, label %c%d, label %c%d\n; <label>:%d:\n", '%',
    varCount, '%', varCount + 1, '%', varCount + 2, varCount + 1);
224     varCount++;
225     result = strcat(result, tmp2);
226     process(node->right->left, result);
227     char *tmp3 = (char *)malloc(sizeof(char) * 200);
228     ++varCount;
229     sprintf(tmp3, "br label %c%d\n; <label>:%d:\n", '%', varCount, varCount);
230     result = strcat(result, tmp3);
231     process(node->right->left->next, result);
232     char *tmp4 = (char *)malloc(sizeof(char) * 200);
233     ++varCount;
234     sprintf(tmp4, "br label %c%d\n; <label>:%d:\n", '%', varCount, varCount);
235     result = strcat(result, tmp4);
236     free(tmp);
237     free(res);
238     return 0; // res
239 }
240 int genWhileStmt(past node, char *result)
241 {
242     char *res = (char *)malloc(200);
243     char *tmp = (char *)malloc(200);
244     sprintf(res, "br label %c%d\n", '%', ++varCount);
245     whilecheckpoint[stack_top++] = varCount;
246     sprintf(tmp, "; <label>:%d:\n", varCount);
247     res = strcat(res, tmp);
248     result = strcat(result, res);
249     genExprStmt(node->left, result);
250     char *tmp2 = (char *)malloc(sizeof(char) * 200);
```

```
251     sprintf(tmp2, "br i1 %c%d, label %c%d, label %c%d\n; <label>:%d:\n", '%',
    varCount - 1, '%', varCount, '%', varCount + 1, varCount + 1);
252     varCount++;
253     result = strcat(result, tmp2);
254     process(node->right, result);
255     char *tmp3 = (char *)malloc(sizeof(char) * 200);
256     sprintf(tmp3, "br label %c%d\n; <label>:%d:\n", '%',
    whilecheckpoint[stack_top - 1], ++varCount);
257     result = strcat(result, tmp3);
258     free(tmp);
259     free(res);
260     return 0;
261 }
262 int genReturnStmt(past node, char *result)
263 {
264     char *tmp = (char *)malloc(sizeof(char) * 200);
265     if (strcmp(node->left->nodeType, "expr") == 0)
266     {
267         sprintf(tmp, "ret i32 %c%d\n", '%', genExprStmt(node->left, result));
268         result = strcat(result, tmp);
269         free(tmp);
270         return 0;
271     }
272     else if (strcmp(node->left->nodeType, "intValue") == 0)
273     {
274         sprintf(tmp, "ret i32 %d\n", node->left->ivalue);
275         result = strcat(result, tmp);
276         free(tmp);
277         return 0;
278     }
279     else if (strcmp(node->left->nodeType, "parameter") == 0)
280     {
281         char *tmp = (char *)malloc(sizeof(char) * 200);
282         sprintf(tmp, "%%%d = load i32, i32* %%%d, align 4\n", ++varCount,
    reg[checkVariable(node->left->strValue)]);
283         result = strcat(result, tmp);
284         free(tmp);
285         char *tmp2 = (char *)malloc(sizeof(char) * 200);
286         sprintf(tmp2, "ret i32 %c%d\n", '%', varCount++);
287         result = strcat(result, tmp2);
288         free(tmp2);
289         return 0;
290     }
291     else
292     {
293         free(tmp);
294         result = "";
```

```
295         return 0;
296     }
297 }
298 
299 int process(past node, char *result)
300 {
301     if (strcmp(node->nodeType, "Decl") == 0)
302     {
303         genDeclStmt(node, result);
304     }
305     else if (strcmp(node->nodeType, "DeclList") == 0)
306     {
307         process(node->left, result);
308         process(node->left->next, result);
309     }
310     else if (strcmp(node->nodeType, "Assign_Stmt") == 0)
311     {
312         genAssignStmt(node, result);
313     }
314     else if (strcmp(node->nodeType, "Block_list") == 0)
315     {
316         past l = node->left;
317         for (int i = 0; i < node->ivalue; i++)
318         {
319             process(l, result);
320             l = l->next;
321         }
322     }
323     else if (strcmp(node->nodeType, "While_Stmt") == 0)
324     {
325         genWhileStmt(node, result);
326     }
327     else if (strcmp(node->nodeType, "Return_Stmt") == 0)
328     {
329         genReturnStmt(node, result);
330     }
331     else if (strcmp(node->nodeType, "If_Stmt") == 0)
332     {
333         genIfStmt(node, result);
334     }
335     else if (strcmp(node->nodeType, "IfElse_Stmt") == 0)
336     {
337         genIfElseStmt(node, result);
338     }
339 }
340 int genExpr(past node)
341 {
```

```
342     if (node == NULL)
343         return -1;
344 
345     if (strcmp(node->nodeType, "Block_list") == 0)
346     {
347         char *result = (char *)malloc(sizeof(char) * 2000);
348         process(node, result);
349         addLLVMCodes(result);
350     }
351     else
352     {
353         if (node->left != NULL)
354         {
355             genExpr(node->left);
356         }
357         if (node->right != NULL)
358         {
359             genExpr(node->right);
360         }
361     }
362 
363     return -1;
364 }
365 
```

`main.c`:

```
1  #include "ast.h"
2  #include "genllvm.h"
3  #include <stdio.h>
4  
5  extern int yyparse();
6  extern FILE *yyin;
7  past astRoot;
8  void yyerror(char *s)
9  {
10     printf("%s\n", s);
11 }
12 
13 int main(int argc, char **argv)
14 {
15     if (argc > 2)
16     {
17         printf("argcs too many!.\n");
```

```
        return 0;
    }
    if (argc == 2)
    {
        yyin = fopen(argv[1], "r");
    }
    else
    {
        yyin = fopen("./test.c", "r");
    }

    // printf("before yyparse\n");
    yyparse();
    fclose(yyin);
    // printf("before show  &  after yyparse\n");
    genExpr(astRoot);
    showAst(astRoot, 0);
    // printf("after show\n");

    return 0;
}

```

`lrlex.l`:

```
%{

#include "ast.h"
#include <string.h>
#include "lrparser.tab.h"

%}

INTERGER   [0-9]
OCTALCONS  0[0-7]+
HEXCONS    0[xX][0-9a-fA-F]+
NOTE_S     \/\/(.)*\n
NOTE_M     \/\*(.|\n)*?\*\/
IDENTIFIER [_a-zA-Z][_a-zA-Z0-9]*

%%

"("                         {return '(';}
")"                         {return ')';}
```

```
"{"                     {return '{';}
"}"                     {return '}';}
"["                     {return '[';}
"]"                     {return ']';}
","                     {return ',';}
";"                     {return ';';}
"+"                     {return '+';}
"-"                     {return '-';}
"*"                     {return '*';}
"/"                     {return '/';}
"%"                     {return '%';}
"<"                     {return '<';}
">"                     {return '>';}
"!"                     {return '!';}
"="                     {return '=';}

"int"                   {return INT;}
"continue"              {return CONTINUE;}
"const"                 {return CONST;}
"else"                  {return ELSE;}
"if"                    {return IF;}
"return"                {return RETURN;}
"void"                  {return VOID;}
"while"                 {return WHILE;}
"break"                 {return BREAK;}

"<="                    {return LESSEQ;}
">="                    {return GREATEQ;}
"!="                    {return NOTEQ;}
"=="                    {return EQ;}
"&&"                    {return AND;}
"||"                    {return OR;}

" "                     {  /*no action and no return*/ }
"\t"                    {  /*no action and no return*/ }
"\n"                    {  /*no action and no return*/ }
{NOTE_S}*               {  /*no action and no return*/ }
{NOTE_M}*               {  /*no action and no return*/ }

{INTERGER}+"."*{INTERGER}*  |
{OCTALCONS}             |
{HEXCONS}               {yylval.number = atoi(yytext); return NUMBER;}

{IDENTIFIER}            {strcpy(yylval.strValue, yytext); return ID;}

%%

```

```
int yywrap(){
    return 1;
}

```

`lrparser.y`:

```
%{

#include "ast.h"
#include <stdio.h>

void yyerror(char *);
int yylex(void);
extern char* yytext;
extern past astRoot;
%}

%union{
        int                 number;
        char                strValue[50];
        past                pAst;
};

%token                          IF
%token                          ELSE
%token                          INT
%token                          VOID
%token                          CONST
%token                          WHILE
%token                          BREAK
%token                          RETURN
%token                          CONTINUE
%token                          LESSEQ
%token                          GREATEQ
%token                          NOTEQ
%token                          EQ
%token                          AND
%token                          OR
%token      <strValue>    ID
%token      <number>      NUMBER
%type       <pAst>          CompUnit CompUnits Decl ConstDecl ConstDef
ConstInitVal VarDecl VarDef InitVal FuncDef FuncFParams FuncFParam Block
BlockItem Stmt Exp Cond LVal PrimaryExp Number UnaryExp FuncRParams MulExp
```

```
AddExp RelExp EqExp LAndExp LOrExp ConstExp ConstDeclMul ConstDefMul
ConstInitValMul VarDeclMul VarDefMul InitValMul BlockMul LValMul

%%

CompUnits: CompUnit                         {$$ = newNextNode("CompUnit", $1,
NULL); astRoot = $$;}
        | CompUnits CompUnit                {$$ = newNextNode("CompUnit", $1,
$2); astRoot = $$;}
        ;

CompUnit: Decl                          {$$ = $1;}
      | FuncDef                         {$$ = $1;}
      ;

Decl: ConstDecl                 {$$ = newBasicNode("Decl", $1, NULL, NULL);}
      | VarDecl                 {$$ = newBasicNode("Decl", $1, NULL, NULL);}
      ;

ConstDeclMul: ConstDef                      {$$ = newNextNode("ConstDecl_list",
$1, NULL);}
      | ConstDeclMul ',' ConstDef           {$$ = newNextNode("ConstDecl_list",
$1, $3);}
      ;

ConstDecl: CONST INT ConstDeclMul ';'               {$$ =
newBasicNode("ConstDecl",newTypeNode("const_int"), $3, NULL);}
      ;

ConstDefMul: '[' ConstExp ']'                   {$$ =
newNextNode("ConstDef_list", $2, NULL);}
      | ConstDefMul '[' ConstExp ']'            {$$ =
newNextNode("ConstDef_list", $1, $3);}
      ;

ConstDef: ID '=' ConstInitVal                   {$$ =
newBasicNode("ConstDef",newIDNode($1),$3,NULL);}
      | ID ConstDefMul '=' ConstInitVal         {$$ =
newBasicNode("ConstDef",newNextNode("ConstDef_para", newIDNode($1), $2), $4,
NULL);}
      ;

ConstInitValMul: ConstInitVal                   {$$ =
newNextNode("ConstInitVal_list", $1, NULL);}
      | ConstInitValMul ',' ConstInitVal        {$$ =
newNextNode("ConstInitVal_list", $1, $3);}
      ;
```

```
69
70 ConstInitVal: ConstExp                    {$$ = $1;}
71       | '{' '}'                          {$$ =
   newBasicNode("ConstInitVal_empty",NULL, NULL, NULL);}
72       | '{' ConstInitValMul '}'          {$$ = $2;}
73       ;
74
75 VarDeclMul: VarDef                        {$$ = newNextNode("VarDecl_list", $1,
   NULL);}
76       | VarDeclMul ',' VarDef            {$$ = newNextNode("VarDecl_list", $1,
   $3);}
77       ;
78
79 VarDecl: INT VarDeclMul ';'              {$$ = newBasicNode("VarDecl",
   newTypeNode("int"), $2, NULL);}
80       ;
81
82 VarDefMul: '[' ConstExp ']'              {$$ = newNextNode("VarDef_list", $2,
   NULL);}
83       | VarDefMul '[' ConstExp ']'       {$$ = newNextNode("VarDef_list", $1,
   $3);}
84       ;
85
86 VarDef: ID                                {$$ = newBasicNode("VarDef",
   newIDNode(yylval.strValue), NULL,NULL);}
87       | ID '=' InitVal                   {$$ = newBasicNode("VarDef",
   newIDNode($1), $3, NULL);}
88       | ID VarDefMul                     {$$ =
   newBasicNode("VarDef",newNextNode("VarDef_para", newIDNode($1), $2), NULL,
   NULL);}
89       | ID VarDefMul '=' InitVal         {$$ =
   newBasicNode("VarDef",newNextNode("VarDef_para", newIDNode($1), $2), $4,
   NULL);}
90       ;
91
92 InitValMul: InitVal                       {$$ = newNextNode("InitVal_list", $1,
   NULL);}
93       | InitValMul ',' InitVal           {$$ = newNextNode("InitVal_list", $1,
   $3);}
94       ;
95
96 InitVal: Exp                              {$$ = $1;}
97       | '{' '}'                          {$$ = newBasicNode("InitVal_empty",
   NULL, NULL, NULL);}
98       | '{' InitValMul '}'               {$$ = $2;}
99       ;
100
```

```
101 FuncDef: VOID ID '('  ')' Block                {$$ =
    newBasicNode("FuncDef",newNextNode("FuncDef_para", newTypeNode("void"),
    newIDNode($2)), $5, NULL);}
102       | INT ID '('  ')' Block                  {$$ =
    newBasicNode("FuncDef",newNextNode("FuncDef_para", newTypeNode("int"),
    newIDNode($2)), $5, NULL);}
103       | VOID ID '(' FuncFParams ')' Block      {past id = newIDNode($2); id
    -> left = $4;
104                                                 $$ =
    newBasicNode("FuncDef",newNextNode("FuncDef_para", newTypeNode("void"), id),
    $6, NULL);}
105       | INT ID '(' FuncFParams ')' Block       {past id = newIDNode($2); id
    -> left = $4;
106                                                 $$ =
    newBasicNode("FuncDef",newNextNode("FuncDef_para", newTypeNode("int"), id),
    $6, NULL);}
107       ;
108
109 FuncFParams: FuncFParam                        {$$ =
    newNextNode("FuncFParams_list", $1, NULL);}
110       |  FuncFParams ',' FuncFParam            {$$ =
    newNextNode("FuncFParams_list", $1, $3);}
111       ;
112
113 FuncFParam: INT ID                       {$$ =
    newBasicNode("FuncFParam",newNextNode("FuncFParam_para", newTypeNode("int"),
    newIDNode($2)), NULL, NULL);}
114       | INT ID '[' ']'                   {$$ =
    newBasicNode("FuncFParam",newNextNode("FuncFParam_para", newTypeNode("int"),
    newIDNode($2)), NULL, NULL);}
115       | INT ID '[' ']' LValMul           {$$ =
    newBasicNode("FuncFParam",newNextNode("FuncFParam_para", newTypeNode("int"),
    newIDNode($2)), $5, NULL);}
116       ;
117
118 BlockMul: BlockItem                {$$ = newNextNode("Block_list", $1, NULL);}
119       | BlockMul BlockItem         {$$ = newNextNode("Block_list", $1, $2);}
120       ;
121
122 Block:  '{' '}'                    {$$ = newBasicNode("Block_empty", NULL,
    NULL, NULL);}
123       | '{' BlockMul '}'           {$$ = $2;}
124       ;
125
126 BlockItem: Decl         {$$ = $1;}
127       | Stmt            {$$ = $1;}
128       ;
```

```
129
130 Stmt: LVal '=' Exp ';'                              {$$ =
    newBasicNode("Assign_Stmt", $1, $3, NULL);}
131       | Exp ';'                                      {$$ = $1;}
132       | ';'                                          {$$ =
    newBasicNode("Stmt_empty", NULL, NULL, NULL);}
133       | Block                                        {$$ = $1;}
134       | IF '(' Cond ')' Stmt                          {$$ = newBasicNode("If_Stmt",
    $3, $5, NULL);}
135       | IF '(' Cond ')' Stmt ELSE Stmt                {$$ =
    newBasicNode("IfElse_Stmt", $3, newNextNode("If_Else", $5, $7), NULL);}
136       | WHILE '(' Cond ')' Stmt                      {$$ =
    newBasicNode("While_Stmt", $3, $5, NULL);}
137       | BREAK ';'                                    {$$ =
    newBasicNode("Break_Stmt",NULL,NULL,NULL);}
138       | CONTINUE ';'                                 {$$ =
    newBasicNode("Continue_Stmt",NULL,NULL,NULL);}
139       | RETURN Exp ';'                               {$$ =
    newBasicNode("Return_Stmt",$2,NULL,NULL);}
140       | RETURN ';'                                   {$$ =
    newBasicNode("Return_Stmt",NULL,NULL,NULL);}
141       ;
142
143 Exp: AddExp             {$$ = $1;}
144       ;
145
146 Cond: LOrExp              {$$ = $1;}
147       ;
148
149 LValMul: '[' Exp ']'                  {$$ = newNextNode("Exp_list", $2, NULL);}
150       |  LValMul '[' Exp ']'          {$$ = newNextNode("Exp_list", $1, $3);}
151       ;
152
153 LVal: ID                {$$ = newIDNode(yylval.strValue);}
154       | ID LValMul      {$$ = newBasicNode("LVal_SEG", newIDNode($1), $2,
    NULL);}
155       ;
156
157 PrimaryExp: '(' Exp ')'       {$$ = $2;}
158             | LVal            {$$ = $1;}
159             | Number          {$$ = $1;}
160             ;
161
162 Number: NUMBER                {$$ = newNum(yylval.number);}
163       ;
164
165 UnaryExp: PrimaryExp                {$$ = $1;}
```

```
166       | ID '(' ')'                {$$ = newBasicNode("UnaryExp",
    newIDNode($1), NULL, NULL);}
167       | ID '(' FuncRParams ')'    {$$ = newBasicNode("UnaryExp",
    newIDNode($1), $3, NULL);}
168       | '+' UnaryExp              {$$ = newBasicNode("UnaryExp",
    newExpr('+', NULL, $2), NULL, NULL);}
169       | '-' UnaryExp              {$$ = newBasicNode("UnaryExp", newExpr('-
    ', NULL, $2), NULL, NULL);}
170       | '!' UnaryExp              {$$ = newBasicNode("UnaryExp",
    newExpr('!', NULL, $2), NULL, NULL);}
171       ;
172
173 FuncRParams: Exp                      {$$ =
    newNextNode("FuncRParams_list", $1, NULL);}
174            | FuncRParams ',' Exp      {$$ =
    newNextNode("FuncRParams_list", $1, $3);}
175            ;
176
177 MulExp: UnaryExp                      {$$ = $1;}
178       | MulExp '*' UnaryExp           {$$ = newExpr('*', $1, $3);}
179       | MulExp '/' UnaryExp           {$$ = newExpr('/', $1, $3);}
180       | MulExp '%' UnaryExp           {$$ = newExpr('%', $1, $3);}
181       ;
182
183 AddExp: MulExp                       {$$ = $1;}
184       | AddExp '+'  MulExp           {$$ = newExpr('+', $1, $3);}
185       | AddExp '-'  MulExp           {$$ = newExpr('-', $1, $3);}
186       ;
187
188 RelExp: AddExp                        {$$ = $1;}
189       | RelExp '<' AddExp             {$$ = newExpr('<', $1, $3);}
190       | RelExp LESSEQ AddExp          {$$ = newDoubleExpr("<=", $1, $3);}
191       | RelExp GREATEQ AddExp         {$$ = newDoubleExpr(">=", $1, $3);}
192       | RelExp '>' AddExp             {$$ = newExpr('>', $1, $3);}
193       ;
194
195 EqExp: RelExp                        {$$ = $1;}
196       | EqExp EQ RelExp               {$$ = newDoubleExpr("==", $1, $3);}
197       | EqExp NOTEQ RelExp            {$$ = newDoubleExpr("!=", $1, $3);}
198       ;
199
200 LAndExp: EqExp                       {$$ = $1;}
201         | LAndExp AND EqExp          {$$ = newDoubleExpr("&&", $1, $3);}
202         ;
203
204 LOrExp: LAndExp                      {$$ = $1;}
205        | LOrExp OR LAndExp           {$$ = newDoubleExpr("||", $1, $3);}
```

```
206         ;
207
208 ConstExp: AddExp               {$$ = $1;}
209       ;
210
211 %%
212
```

`Makefile`:

```
1 all: lrparser.tab.c lex.yy.c ast.c main.c genllvm.c
2     gcc -o genllvm lrparser.tab.c lex.yy.c ast.c genllvm.c main.c
3
4 lrparser.tab.c : lrparser.y
5     bison -d lrparser.y
6
7 lex.yy.c : lrlex.l
8     flex lrlex.l
```