

final.1

```
1  %{
2      #include<stdio.h>
3      #include<string.h>
4      #include <stdlib.h>
5      #include <stdbool.h>
6      #include <ctype.h>
7      #include "final.tab.h"
8
9      // header, single, multi, var, print, dis, calc_val, if, till else
10     int tempCounter[11];
11
12     int blockBalancer = 0;
13     int stack[100];
14     int stackSize = 0;
15
16     const int MAXIMUM_VAR_SIZE = 10;
17
18     char error[200];
19
20     // extern
21     char lastDataType[10];
22     char *outBuffer = NULL;
23     int outBufferSize = 0;
24     bool isLastIfValid = false;
25     //extern
26
27     int commentCounter = 0;
28     int comment_depth = 0;
29     char *comment_buffer = NULL;
30
31     bool canDeclareHeader = true;
32     bool isMainBlockFound = false;
33     bool isCommimgFromMain = false;
34     bool isCommimgFromVar = false;
35
36     size_t buffer_length = 0;
37
38     int bracketCounter=0;
39
40     char *removedRedundant(char*);
41     bool getValidity();
42     void insertImport(char*);
43
44     // variable
```

```
45     const int MAXIMUM_VARIABLE_LENGTH = 200;
46
47     int pushState(int id){
48         stackSize++;
49         stack[stackSize] = id;
50         return id;
51     }
52
53     int popState(){
54         if(stackSize > 0) stackSize--;
55         return stack[stackSize];
56     }
57
58     void inc(int index){
59         tempCounter[index]++;
60     }
61
62     void initSingleComment(){
63         commentCounter++;
64         printf("\nSingle line comment: %s\n",yytext);
65     }
66
67     void appendToBuffer(char ch){
68         comment_buffer = (char*) realloc(comment_buffer, buffer_length + 1);
69         buffer_length += sprintf(comment_buffer + buffer_length, "%c", ch);
70     }
71
72     void appendTextToBuffer(char* ch){
73
74         if( strcmp(ch, /*",2) == 0) comment_depth++;
75         if( strcmp(ch, /*",2) == 0) comment_depth--;
76         comment_buffer = (char*) realloc(comment_buffer, buffer_length +
strlen(ch) );
77         buffer_length += sprintf(comment_buffer + buffer_length, ch);
78     }
79
80     void initMultiComment(){
81         comment_depth = 1;
82         buffer_length = 2;
83         comment_buffer = (char*) malloc(3);
84
85         memset(comment_buffer, 0, 3);
86         strcat(comment_buffer, "/*");
87     }
88
89     void resetBuffer(){
90         free(comment_buffer);
```

```

91 comment_buffer = NULL;
92 buffer_length = 0;
93 }
94
95 void process_comment() {
96     commentCounter++;
97     printf("\nCaptured comment: %s\n\n", comment_buffer);
98     //printf("Multi-line comment found\n");
99 }
100
101 void stopProgram(char *error){
102     printf("%s\n", error);
103     free(comment_buffer);
104     exit(1);
105 }
106
107 void checkForEnd(){
108     if (comment_depth == 0) {
109         BEGIN_INITIAL;
110         process_comment();
111         resetBuffer();
112     }
113     else if(comment_depth < 0){
114         resetBuffer();
115         stopProgram("Invalid comment found");
116     }
117 }
118
119 void initMain(){
120     canDeclareHeader = false;
121     //printf("execution started\n");
122 }
123
124 void processHeader(){
125     if(!canDeclareHeader){
126         stopProgram("Header must be at top");
127         return;
128     }
129 }
130
131
132 void initVarSec(char *temp){
133     char *type = removeRedundant(temp);
134     strcpy(lastDataType, type);
135     canDeclareHeader = false;
136 }
137

```

```

138 void sendNumber(char *text){
139     double num = 0;
140     sscanf(text, "%lf", &num);
141     yylval.num = num;
142     //printf("=====%lf=====", num);
143 }
144
145 void initOutBuffer(){
146     outBuffer = (char*) malloc(1);
147     memset(outBuffer, 0, 1);
148     outBufferSize = 1;
149 }
150
151 char* removeLpRp(char *val){
152     char *op = malloc(14*sizeof(char));
153     char *temp = strdup(val);
154     strncpy(op, temp+1, strlen(temp) - 2 );
155     return op;
156 }
157
158 %}
159
160 %x COMMENT
161 %x MAIN
162 %x VAR_SEC
163 %x IF_SEC
164 %x ELSE_SEC
165 %x IGNORE_SEC
166 %x DISCARD_SEC
167 %x LOOP_SEC
168 %x LOOP_BODY_SEC
169 %x OUT_SEC
170 %x PROTO_SEC
171 %x FUNC_SEC
172 %x FUNC_SEC_VAR
173
174 DQ \"
175 NUMBER [-]?([0-9]+(\".\"[0-9]+)|[-]?([0-9]+)|(\".\"[0-9]+)
176 VARIABLE [a-zA-Z][a-zA-Z0-9]*
177 VAR_NUM ({NUMBER}){VARIABLE}
178 COND_OP ("<lt>"|"<gt>"|"<eq>"|"<neq>"|"<le>"|"<ge>"
179 ARITH_OP ("<add>"|"<sub>"|"<mul>"|"<div>"|"<rem>")
180
181 HEADER "import "[a-zA-Z]+";"
182
183 OUT_START "println("[ ]*"
184 OUT_BODY_CONST {DQ}[^\"]*{DQ}

```

```

185 OUT_BODY_VAR {VAR_NUM}
186 OUT_SEP [ ]*(","|" ")[ ]*
187 OUT_END [ ]**)[ ]*;[ ]*
188
189 MAIN_START "static void entryPoint"[ ]*"("[ ]**)"[ ]*{"[ ]*
190 MAIN_END "}"
191
192 SINGLE_LINE_COMMENT ("///<").*(\n)?
193
194 VARIABLE_ONLY [ ]*{VARIABLE}*[ ]*(,.)
195 VARIABLE_VALUE [ ]*{VARIABLE}[ ]*(="")[ ]*{NUMBER},.)
196 VARIABLE_VALUE_ASSIGN_CONST [ ]*{VARIABLE}[ ]*(="")[ ]*{NUMBER};
197 VARIABLE_VALUE_ASSIGN_VAR [ ]*{VARIABLE}[ ]*(="")[ ]*{VARIABLE};
198 VARIABLE_VALUE_ASSIGN_CALC [ ]*{VARIABLE}[ ]*(="")[ ]*{VAR_NUM}[ ]*
199 {ARITH_OP}[ ]*{VAR_NUM}[ ]*,
200 VARIABLE_VALUE_ASSIGN_CALC_LAST [ ]*{VARIABLE}[ ]*(="")[ ]*{VAR_NUM}[ ]*
201 {ARITH_OP}[ ]*{VAR_NUM}[ ]*;
202
203 DISCARD_START "discard "
204
205 VARIABLE_ONLY_LAST [ ]*[a-zA-Z][a-zA-Z0-9]*[ ]*(;.)
206 VARIABLE_VALUE_LAST [ ]*[a-zA-Z][a-zA-Z0-9]*[ ]*(="")[ ]*{NUMBER};(.)
207
208 VAR_SPACE [ ]*
209
210 DATA_TYPE ("int"|"float"|"double")[ ]
211 FUNC_TYPE ("int"|"float"|"double"|"void")
212
213 IF "justInCase"[ ]*
214 ELSE "otherwise"
215 IF_BODY_START "{
216 IF_BODY_END [ ]**}"
217 IF_SPACE [ ]*
218
219 IGNORE_LEFT_BRACE "{"
220 IGNORE_RIGHT_BRACE "}"
221
222 FUNC_START "@"{VARIABLE}
223
224 LOOP_START "till"
225 LOOP_START_BRACE "{
226 LOOP_OTHERS [^{}\\n]*\\n
227 LOOP_END {IF_BODY_END}
228
229 NEW_LINE_AND_TAB [\\n\\t]*

```

```

230 {SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
231 "/*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
232
233
234
235 <COMMENT>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
236 <MAIN>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
237 <VAR_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
238 <IF_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
239 <ELSE_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
240 <IGNORE_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
241 <DISCARD_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
242 <LOOP_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
243 <LOOP_BODY_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
244 <OUT_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
245 <PROTO_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
246 <FUNC_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
247 <FUNC_SEC_VAR>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
248
249 <MAIN>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
250 <VAR_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
251 <IF_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
252 <ELSE_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
253 <IGNORE_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
254 <DISCARD_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
255 <LOOP_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
256 <LOOP_BODY_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
257 <OUT_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
258 <PROTO_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
259 <FUNC_SEC>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
260 <FUNC_SEC_VAR>"/**" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
261
262 <COMMENT>"/**" { appendTextToBuffer("/**"); }
263
264 <COMMENT>"/**" {
265     appendTextToBuffer("*/"); checkForEnd();
266     if(comment_depth == 0){ inc(2); BEGIN( popState() ); }
267 }
268
269 <COMMENT>{NEW_LINE_AND_TAB}* { appendTextToBuffer("\n"); }
270 <COMMENT>. { appendToBuffer(yytext[0]); }
271
272 {HEADER} { insertImport(yytext); inc(0); processHeader(); }
273
274 {FUNC_TYPE} {
275     //printf("m_type: -%s-\\n", yytext);
276     BEGIN( pushState(PROTO_SEC) );
277     char *type = removeRedundant(yytext);

```

```

277 yylval.name = strdup( type );
278 return FUNC_TYPE;
279 }
280
281 <PROTO_SEC>{FUNC_TYPE} {
282     //printf("type: %s\n",yytext);
283     char *type = removeRedundant(yytext);
284     yylval.name = strdup( type );
285     return FUNC_TYPE;
286 }
287 <PROTO_SEC>{VARIABLE} {
288     //printf("name: %s\n",yytext);
289     yylval.name = strdup(yytext);
290     return FUNC_NAME;
291 }
292
293 <PROTO_SEC>,"" { return *yytext; }
294 <PROTO_SEC> "(" { return *yytext; }
295 <PROTO_SEC>";" { BEGIN( popState() ); return *yytext; }
296
297 {MAIN_START} {
298     blockBalancer++;
299     //printf("main start %s\n",yytext);
300     initMain();
301     BEGIN( pushState(MAIN) );
302     return ENTRY_POINT;
303 }
304
305 <MAIN>{DATA_TYPE} {
306     //printf("main data type %s\n",yytext);
307     initVarSec(yytext);
308     char *type = removeRedundant(yytext);
309     yylval.name = strdup( type );
310     BEGIN( pushState(VAR_SEC) );
311     return DATA_TYPE;
312 }
313
314 <MAIN>{OUT_START} {
315     initOutBuffer();
316     BEGIN( pushState(OUT_SEC) );
317     return OUTPUT;
318 }
319
320 <MAIN>{IF} {
321     inc(7);
322     BEGIN( pushState(IF_SEC) );
323     return JUST_IN_CASE;
324 }
325 <MAIN>{ELSE} {
326     inc(10);
327     BEGIN( pushState(ELSE_SEC) );
328     return ELSE;
329 }
330
331 <MAIN>{LOOP_START} {
332     inc(8);
333     BEGIN( pushState(LOOP_SEC) );
334     return TILL;
335 }
336
337 <MAIN>{FUNC_START} {
338     //printf("----s----",yytext);
339     BEGIN( pushState(FUNC_SEC) );
340     yylval.name = strdup(yytext);
341     return FUNC_NAME;
342 }
343
344 <VAR_SEC>{FUNC_START} {
345     //printf("----s----",yytext);
346     BEGIN( pushState(FUNC_SEC_VAR) );
347     yylval.name = strdup(yytext);
348     return FUNC_NAME;
349 }
350
351 <FUNC_SEC_VAR>("|",")" { return *yytext; }
352 <FUNC_SEC_VAR>("(") { BEGIN( popState() ); return *yytext; }
353 <FUNC_SEC_VAR>{VAR_NUM} { yylval.name = strdup(yytext); return VAR_CON; }
354
355 <FUNC_SEC>{VAR_NUM} { yylval.name = strdup(yytext); return VAR_CON; }
356
357 <FUNC_SEC>("|",")" {
358     return *yytext;
359 }
360
361 <FUNC_SEC>";" {
362     BEGIN( popState() );
363     return *yytext;
364 }
365
366 <MAIN>{VARIABLE} {
367     yylval.name = strdup(yytext);
368     BEGIN( pushState(VAR_SEC) );
369     return VAR;
370 }

```

```

371 }
372
373 <VAR_SEC>{NUMBER} {
374     sendNumber(yytext);
375     return NUMBER;
376 }
377 <VAR_SEC>{ARITH_OP} {
378     yylval.name = removeLpRp(yytext);
379     return ARITH_OPE;
380 }
381 <VAR_SEC>{VARIABLE} {
382     yylval.name = strdup(yytext);
383     return VAR;
384 }
385 <VAR_SEC>"," { return *strdup(yytext); }
386 <VAR_SEC>"=" { return *strdup(yytext); }
387 <VAR_SEC>";" {
388     BEGIN( popState() );
389     return *strdup(yytext);
390 }
391 <VAR_SEC>{VAR_SPACE} { }
392
393
394 <MAIN>{DISCARD_START} {
395     //printf("Discard section\n");
396     inc(5);
397     BEGIN( pushState(DISCARD_SEC) );
398     return DISCARD;
399 }
400
401 <DISCARD_SEC>{VARIABLE} {
402     //printf("\nFrom discard only: %s-\n", yytext);
403     yylval.name = strdup(yytext);
404     return VAR;
405 }
406 <DISCARD_SEC>"," { return *yytext; }
407 <DISCARD_SEC>";" { BEGIN( popState() ); return *yytext; }
408
409
410 <OUT_SEC>{OUT_BODY_CONST} {
411     yylval.name = strdup(yytext);
412     return OUTPUT_VC;
413 }
414 <OUT_SEC>{OUT_BODY_VAR} {
415     yylval.name = strdup(yytext);
416     return OUTPUT_VC;
417 }
418
419 <OUT_SEC>{OUT_SEP} {
420     return OUTPUT_SEP;
421 }
422 <OUT_SEC>{OUT_END} {
423     inc(4);
424     BEGIN( popState() );
425     return OUTPUT_END;
426 }
427 <ELSE_SEC>("("|"") { return *yytext; }
428 <ELSE_SEC>"}" { return END_POINT; }
429 <ELSE_SEC>{IF_BODY_START} {
430     //popState(); // popping else section and taking to main section arki
431     popState();
432
433     bool isLastIfValid = getCurrentValidity();
434
435     char *valid = !isLastIfValid ? "True" : "False";
436     printf("\nElse condition is %s\n", valid);
437     if(isLastIfValid){
438         bracketCounter=1;
439         BEGIN( pushState(IGNORE_SEC) );
440     }
441     else{
442         BEGIN(pushState(MAIN));
443     }
444     return *yytext;
445 }
446
447
448 <IF_SEC>("("|"") { return *yytext; }
449 <IF_SEC>"}" { return END_POINT; }
450
451 <IF_SEC>{COND_OP} { yylval.name = removeLpRp(yytext); return COND_OPE; }
452 <IF_SEC>"and" { return AND; }
453 <IF_SEC>"or" { return OR; }
454 <IF_SEC>{VAR_NUM} { yylval.name = strdup(yytext); return VAR_CON; }
455 <IF_SEC>{IF_BODY_START} {
456     //popState(); // popping if section and taking to main section arki
457     popState();
458
459     bool test = getCurrentValidity();
460     char *valid = test ? "True" : "False";
461     printf("\nIf condition is %s\n", valid);
462
463     if(!test){
464         bracketCounter=1;

```

```

511 }
512
513 <MAIN>{NEW_LINE_AND_TAB}* { }
514 {NEW_LINE_AND_TAB}* { }
515
516 %%
517
518 int yywrap(){
519     return 1;
520 }
521

```

```

465         BEGIN( pushState(IGNORE_SEC) );
466     }
467     else{
468         BEGIN(pushState(MAIN));
469     }
470     return *yytext;
471 }
472 <IF_SEC>{NEW_LINE_AND_TAB} {}
473
474 <IGNORE_SEC>{IGNORE_LEFT_BRACE} { bracketCounter++; }
475 <IGNORE_SEC>{IGNORE_RIGHT_BRACE} {
476     bracketCounter--;
477     if(bracketCounter == 0){
478         BEGIN( popState() );
479         return END_POINT;
480     }
481 }
482 <IGNORE_SEC>[^] {}
483
484
485 <LOOP_SEC>("("|")") { return *yytext; }
486 <LOOP_SEC>"}" { return END_POINT; }
487 <LOOP_SEC>{COND_OP} { yyval.name = removeLpRp(yytext); return COND_OPE; }
488 <LOOP_SEC>{VAR_NUM} { yyval.name = strdup(yytext); return VAR_CON; }
489
490 <LOOP_SEC>{LOOP_START_BRACE} { BEGIN( pushState(LOOP_BODY_SEC) ); return
    *yytext; }
491
492 <LOOP_BODY_SEC>{VARIABLE} {
493     yyval.name = strdup(yytext);
494     BEGIN( pushState(VAR_SEC) );
495     return VAR;
496 }
497 <LOOP_SEC>{NEW_LINE_AND_TAB} {}
498 <LOOP_BODY_SEC>{LOOP_END} {
499     popState();
500     BEGIN( popState() );
501     return END_POINT;
502 }
503
504 <LOOP_BODY_SEC>{NEW_LINE_AND_TAB} {}
505
506 <MAIN>{MAIN_END} {
507     blockBalancer--;
508     BEGIN( popState() );
509     //printf("\n-----main end--\n");
510     return END_POINT;

```