

final.1

```
1  %{\n2  #include<stdio.h>\n3  #include<string.h>\n4  #include <stdlib.h>\n5  #include <stdbool.h>\n6  #include <ctype.h>\n7  #include "final.tab.h"\n8\n9  // header, single, multi, var, print, dis, calc_val, if, till\n10 int tempCounter[10];\n11\n12 int blockBalancer = 0;\n13 int stack[100];\n14 int stackSize = 0;\n15\n16 const int MAXIMUM_VAR_SIZE = 10;\n17\n18 char error[200];\n19\n20 // extern\n21 char lastDataType[10];\n22 char *outBuffer = NULL;\n23 int outBufferSize = 0;\n24 bool isLastIfValid = false;\n25 //extern\n26\n27 int commentCounter = 0;\n28 int comment_depth = 0;\n29 char *comment_buffer = NULL;\n30\n31 bool canDeclareHeader = true;\n32 bool isMainBlockFound = false;\n33 bool isCommingFromMain = false;\n34 bool isCommingFromVar = false;\n35\n36 size_t buffer_length = 0;\n37\n38 int bracketCounter=0;\n39\n40 char *removeRedundant(char*);\n41\n42 void insertImport(char*);\n43\n44 // variable\n45 const int MAXIMUM_VARIABLE_LENGTH = 200;\n46\n47 int pushState(int id){\n48     stackSize++;
```

```
49     stack[stackSize] = id;\n50     return id;\n51 }\n52\n53 int popState(){\n54     if(stackSize > 0) stackSize--;\n55     return stack[stackSize];\n56 }\n57\n58 void inc(int index){\n59     tempCounter[index]++;\n60 }\n61\n62 void initSingleComment(){\n63     commentCounter++;\n64     printf("\\nSingle line comment: %s\\n",yytext);\n65 }\n66\n67 void appendToBuffer(char ch){\n68     comment_buffer = (char*) realloc(comment_buffer, buffer_length + 1);\n69     buffer_length += sprintf(comment_buffer + buffer_length, "%c", ch);\n70 }\n71\n72 void appendTextToBuffer(char* ch){\n73\n74     if( strcmp(ch,"/*",2) == 0) comment_depth++;\n75     if( strcmp(ch,"*/",2) == 0) comment_depth--;\n76     comment_buffer = (char*) realloc(comment_buffer, buffer_length + strlen(ch)\n77 );\n78\n79     buffer_length += sprintf(comment_buffer + buffer_length, ch);\n80 }\n81\n82 void initMultiComment(){\n83     comment_depth = 1;\n84     buffer_length = 2;\n85     comment_buffer = (char*) malloc(3);\n86\n87     memset(comment_buffer, 0, 3);\n88     strcat(comment_buffer, "/*");\n89\n90 void resetBuffer(){\n91     free(comment_buffer);\n92     comment_buffer = NULL;\n93     buffer_length = 0;\n94 }\n95\n96 void process_comment() {\n97     commentCounter++;\n98     printf("\\nCaptured comment: %s\\n\\n", comment_buffer);\n99     //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 = removedRedundant(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     strcpy(op,temp+1, strlen(temp) - 2 );
155     return op;
156 }
157
158 %}
159
160 %% COMMENT
161 %% MAIN
162 %% VAR_SEC
163 %% IF_SEC
164 %% IGNORE_SEC
165 %% DISCARD_SEC
166 %% LOOP_SEC
167 %% LOOP_BODY_SEC
168 %% OUT_SEC
169 %% PROTO_SEC
170 %% FUNC_SEC
171 %% FUNC_SEC_VAR
172
173 DQ \"
174 NUMBER [-]?([0-9]+(\".\"[0-9]+)|[-]?([0-9]+)|(\".\"[0-9]+)
175 VARIABLE [a-zA-Z][a-zA-Z0-9]*
176 VAR_NUM (<NUMBER>|{VARIABLE})
177 COND_OP (<lt>|\"<gt;\"|\"<eq>\"|\"<neq>\"|\"<le>\"|\"<ge>\"|
178 ARITH_OP (<add>|\"<sub>\"|\"<mul>\"|\"<div>\"|\"<dif>\"|\"<rem>\"|
179
180 HEADER \"import \"[a-zA-Z]+\";\"
181
182 OUT_START \"println(\"[ ]*
183 OUT_BODY_CONST {DQ}[^\\\"]*{DQ}
184 OUT_BODY_VAR {VAR_NUM}
185 OUT_SEP [ ]*(\"|\" \" \"[ ]*
186 OUT_END [ ]*\"\"[ ]*\"[ ]*
187
188 MAIN_START \"static void EntryPoint\"[ ]*(\"[ ]*\"[ ]*\"[ ]*\"{
189 MAIN_END \"}\"
190
191 SINGLE_LINE_COMMENT (\"/\"/\"/\".*(\\n)?
192
193 VARIABLE_ONLY [ ]*{VARIABLE}*\"[ ]*(,
194 VARIABLE_VALUE [ ]*{VARIABLE}\"[ ]*(\"=\")[ ]*{NUMBER}\"},
195 VARIABLE_VALUE_ASSIGN_CONST [ ]*{VARIABLE}\"[ ]*(\"=\")[ ]*{NUMBER}\";
196 VARIABLE_VALUE_ASSIGN_VAR [ ]*{VARIABLE}\"[ ]*(\"=\")[ ]*{VARIABLE}\";
197 VARIABLE_VALUE_ASSIGN_CALC [ ]*{VARIABLE}\"[ ]*(\"=\")[ ]*{VAR_NUM}\"[ ]*
198 {VAR_NUM}\"[ ]*,
199 VARIABLE_VALUE_ASSIGN_CALC_LAST [ ]*{VARIABLE}\"[ ]*(\"=\")[ ]*{VAR_NUM}\"[ ]*{ARITH_OP}\"[ ]*
200 {VAR_NUM}\"[ ]*;

```

```

199 DISCARD_START "discard "
200
201 VARIABLE_ONLY_LAST [ ]*[a-zA-Z][a-zA-Z0-9]*[ ]*(;);
202 VARIABLE_VALUE_LAST [ ]*[a-zA-Z][a-zA-Z0-9]*[ ]*( "=" [ ]*{NUMBER} );
203
204
205 VAR_SPACE [ ]*
206
207 DATA_TYPE ("int"|"float"|"double")[ ]
208 FUNC_TYPE ("int"|"float"|"double"|"void")
209
210 IF "justInCase"[ ]*
211 IF_BODY_START "{"
212 IF_BODY_END [ ]*"}"
213 IF_SPACE [ ]*
214
215 IGNORE_LEFT_BRACE "{"
216 IGNORE_RIGHT_BRACE "}"
217
218 FUNC_START "@"{VARIABLE}
219
220 LOOP_START "till"
221 LOOP_START_BRACE "{"
222 LOOP_OTHERS ["^{\n}*\n
223 LOOP_END {IF_BODY_END}
224
225 NEW_LINE_AND_TAB [\n\t]*
226
227 %%
228
229 {SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
230 "/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
231
232
233 <COMMENT>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
234 <MAIN>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
235 <VAR_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
236 <IF_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
237 <IGNORE_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
238 <DISCARD_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
239 <LOOP_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
240 <LOOP_BODY_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
241 <OUT_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
242 <PROTO_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
243 <FUNC_SEC>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
244 <FUNC_SEC_VAR>{SINGLE_LINE_COMMENT} { inc(1); initSingleComment(); }
245
246 <MAIN>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
247 <VAR_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
248 <IF_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
249 <IGNORE_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }

```

```

250 <DISCARD_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
251 <LOOP_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
252 <LOOP_BODY_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
253 <OUT_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
254 <PROTO_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
255 <FUNC_SEC>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
256 <FUNC_SEC_VAR>/"* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
257
258 <COMMENT>/"* { appendTextToBuffer("/"); }
259
260 <COMMENT>/"* {
261     appendTextToBuffer("*/"); checkForEnd();
262     if(comment_depth == 0){ inc(2); BEGIN( popState() ); }
263 }
264 <COMMENT>{NEW_LINE_AND_TAB}* { appendTextToBuffer("\n"); }
265 <COMMENT>. { appendToBuffer(yytext[0]); }
266
267 {HEADER} { insertImport(yytext); inc(0); processHeader();}
268
269 {FUNC_TYPE} {
270     //printf("m_type: %s-\n",yytext);
271     BEGIN( pushState(PROTO_SEC) );
272     char *type = removeRedundant(yytext);
273     yylval.name = strdup( type );
274     return FUNC_TYPE;
275 }
276
277 <PROTO_SEC>{FUNC_TYPE} {
278     //printf("type: %s-\n",yytext);
279     char *type = removeRedundant(yytext);
280     yylval.name = strdup( type );
281     return FUNC_TYPE;
282 }
283 <PROTO_SEC>{VARIABLE} {
284     //printf("name: %s-\n",yytext);
285     yylval.name = strdup(yytext);
286     return FUNC_NAME;
287 }
288
289 <PROTO_SEC>," { return *yytext; }
290 <PROTO_SEC>{" { return *yytext; }
291 <PROTO_SEC>";" { BEGIN( popState() ); return *yytext; }
292
293 {MAIN_START} {
294     blockBalancer++;
295     //printf("main start %s\n",yytext);
296     initMain();
297     BEGIN( pushState(MAIN) );
298     return ENTRY_POINT;
299 }
300

```

```

301 <MAIN>{DATA_TYPE} {
302     //printf("main data type %s\n",yytext);
303     initVarSec(yytext);
304     char *type = removeRedundant(yytext);
305     yylval.name = strdup( type );
306     BEGIN( pushState(VAR_SEC) );
307     return DATA_TYPE;
308 }
309
310 <MAIN>{OUT_START} {
311     initOutBuffer();
312     BEGIN( pushState(OUT_SEC) );
313     return OUTPUT;
314 }
315
316 <MAIN>{IF} {
317     inc(7);
318     BEGIN( pushState(IF_SEC) );
319     return JUST_IN_CASE;
320 }
321
322 <MAIN>{LOOP_START} {
323     inc(8);
324     BEGIN( pushState(LOOP_SEC) );
325     return TILL;
326 }
327
328 <MAIN>{FUNC_START} {
329     //printf("----%s----",yytext);
330     BEGIN( pushState(FUNC_SEC) );
331     yylval.name = strdup(yytext);
332     return FUNC_NAME;
333 }
334
335 <VAR_SEC>{FUNC_START} {
336     //printf("----%s----",yytext);
337     BEGIN( pushState(FUNC_SEC_VAR) );
338     yylval.name = strdup(yytext);
339     return FUNC_NAME;
340 }
341
342 <FUNC_SEC_VAR>{"|",","} { return *yytext; }
343 <FUNC_SEC_VAR>{"(") { BEGIN( popState() ); return *yytext; }
344 <FUNC_SEC_VAR>{VAR_NUM}{ yylval.name = strdup(yytext); return VAR_CON; }
345
346
347 <FUNC_SEC>{VAR_NUM} { yylval.name = strdup(yytext); return VAR_CON; }
348
349 <FUNC_SEC>{"(|"|"") {
350     return *yytext;
351 }

```

```

352 <FUNC_SEC>";" {
353     BEGIN( popState() );
354     return *yytext;
355 }
356
357 <MAIN>{VARIABLE} {
358     yylval.name = strdup(yytext);
359     BEGIN( pushState(VAR_SEC) );
360     return VAR;
361 }
362
363 <VAR_SEC>{NUMBER} {
364     sendNumber(yytext);
365     return NUMBER;
366 }
367
368 <VAR_SEC>{ARITH_OP} {
369     yylval.name = removeLpRp(yytext);
370     return ARITH_OPE;
371 }
372
373 <VAR_SEC>{VARIABLE} {
374     yylval.name = strdup(yytext);
375     return VAR;
376 }
377
378 <VAR_SEC>";" { return *strdup(yytext); }
379 <VAR_SEC> "=" { return *strdup(yytext); }
380 <VAR_SEC>";" {
381     BEGIN( popState() );
382     return *strdup(yytext);
383 }
384
385 <MAIN>{DISCARD_START} {
386     //printf("Discard section\n");
387     inc(5);
388     BEGIN( pushState(DISCARD_SEC) );
389     return DISCARD;
390 }
391
392 <DISCARD_SEC>{VARIABLE} {
393     //printf("\nFrom discard only: -%s-\n",yytext);
394     yylval.name = strdup(yytext);
395     return VAR;
396 }
397 <DISCARD_SEC>";" { return *yytext; }
398 <DISCARD_SEC>";" { BEGIN( popState() ); return *yytext; }
399
400 <OUT_SEC>{OUT_BODY_CONST} {
401     yylval.name = strdup(yytext);
402

```

```

403         return OUTPUT_VC;
404     }
405     <OUT_SEC>{OUT_BODY_VAR} {
406         yyval.name = strdup(yytext);
407         return OUTPUT_VC;
408     }
409     <OUT_SEC>{OUT_SEP} {
410         return OUTPUT_SEP;
411     }
412     <OUT_SEC>{OUT_END} {
413         inc(4);
414         BEGIN( popState() );
415         return OUTPUT_END;
416     }
417
418     <IF_SEC>{"(|")" } return *yytext; }
419     <IF_SEC>"}" { return END_POINT; }
420
421     <IF_SEC>{COND_OP} { yyval.name = removeLpRp(yytext); return COND_OPE; }
422     <IF_SEC>"and" { return AND; }
423     <IF_SEC>"or" { return OR; }
424     <IF_SEC>{VAR_NUM} { yyval.name = strdup(yytext); return VAR_CON; }
425     <IF_SEC>{IF_BODY_START} {
426         //popping if section and taking to main section arki
427         popState();
428
429         char *valid = isLastIfValid ? "True" : "False";
430         printf("\nIf condition is %s\n",valid);
431         if(!isLastIfValid){
432             bracketCounter=1;
433             BEGIN( pushState(IGNORE_SEC) );
434         }
435         else{
436             BEGIN(pushState(MAIN));
437         }
438         return *yytext;
439     }
440     <IF_SEC>{NEW_LINE_AND_TAB} {}
441
442     <IGNORE_SEC>{IGNORE_LEFT_BRACE} { bracketCounter++; }
443     <IGNORE_SEC>{IGNORE_RIGHT_BRACE} {
444         bracketCounter--;
445         if(bracketCounter == 0){
446             BEGIN( popState() );
447             return END_POINT;
448         }
449     }
450     <IGNORE_SEC>[^] {}
451
452     <LOOP_SEC>{"(|")" } return *yytext; }
453
454     <LOOP_SEC>"}" { return END_POINT; }
455     <LOOP_SEC>{COND_OP} { yyval.name = removeLpRp(yytext); return COND_OPE; }
456     <LOOP_SEC>{VAR_NUM} { yyval.name = strdup(yytext); return VAR_CON; }
457
458     <LOOP_SEC>{LOOP_START_BRACE} { BEGIN( pushState(LOOP_BODY_SEC) ); return *yytext; }
459
460     <LOOP_BODY_SEC>{VARIABLE} {
461         yyval.name = strdup(yytext);
462         BEGIN( pushState(VAR_SEC) );
463         return VAR;
464     }
465     <LOOP_SEC>{NEW_LINE_AND_TAB} {}
466     <LOOP_BODY_SEC>{LOOP_END} {
467         popState();
468         BEGIN( popState() );
469         return END_POINT;
470     }
471
472     <LOOP_BODY_SEC>{NEW_LINE_AND_TAB} {}
473
474     <MAIN>{MAIN_END} {
475         blockBalancer--;
476         BEGIN( popState() );
477         return END_POINT;
478     }
479
480     <MAIN>{NEW_LINE_AND_TAB}* {}
481     {NEW_LINE_AND_TAB}* {}
482
483     %%
484
485     int yywrap() {
486         return 1;
487     }
488

```