

constant.h

```
1 //include<stdbool.h>
2 #ifndef CONSTANT_H
3 #define CONSTANT_H
4
5 //returns true if ch is an arithmetic operator
6 bool isArithOp(char *ch);
7
8 // true if op is a valid conditional operator
9 bool isCondOpValid(char* op);
10
11 // true if condition is valid after performing operation on given parameter
12 bool isConditionValid(double left, char* op, double right);
13
14 // perform arithmetic operations and returns result
15 double getValue(double left, double right, char *op);
16
17 #endif
```

constant.c

```
1 #include "constant.h"
2 #include<stdbool.h>
3 #include<string.h>
4 #include<stdio.h>
5 #include <stddef.h>
6 #include<stdlib.h>
7 #include <cctype.h>
8
9
10 int COND_OPERATORS_SIZE = 6;
11 char COND_OPERATORS[6][5] = {
12     "lt", "gt", "eq", "neq", "le", "ge"
13 };
14
15 int ARITHMATIC_OPERATORS_SIZE = 6;
16 char ARITHMATIC_OPERATORS[6][5] = {
17     "add", "sub", "mul", "div", "dif", "rem"
18 };
19
20 bool isArithOp(char *ch){
21     for(int i=0; i<ARITHMATIC_OPERATORS_SIZE; i++){
22         if( strcmp(ARITHMATIC_OPERATORS[i], ch) == 0 ) return true;
23     }
24     return false;
25 }
26
27 bool isCondOpValid(char* op){
28     //char arr[6][5] = { "lt", "gt", "eq", "neq", "le", "ge" };
29     for(int i=0; i<COND_OPERATORS_SIZE; i++){
30         if(strcmp(COND_OPERATORS[i],op) == 0) return true;
31     }
32     return false;
33 }
34
35 bool isConditionValid(double left, char* op, double right){
36     if( strncmp(op,"lt",2) == 0) return (left < right);
37     if( strncmp(op,"gt",2) == 0) return (left > right);
38     if( strncmp(op,"eq",2) == 0) return (left == right);
39
40     if( strncmp(op,"le",2) == 0) return (left <= right);
41     if( strncmp(op,"ge",2) == 0) return (left >= right);
42     if( strncmp(op,"neq",2) == 0) return (left != right); // 2 fine also
43 }
44
45
46 double getValue(double left, double right, char *op){
47     // "add", "sub"
48     if( strncmp(op,ARITHMATIC_OPERATORS[0],3) == 0 ) {
49         //printf("\nadd %lf %lf\n",left,right);
50         return left+right;
51     }
52     if( strncmp(op,ARITHMATIC_OPERATORS[1],3) == 0 ) return left-right;
53 }
```

```
54 // "mul", "div"
55 if( strncmp(op,ARITHMATIC_OPERATORS[2],3) == 0 ) return left*right;
56 if( strncmp(op,ARITHMATIC_OPERATORS[3],3) == 0 ) return left/( right == 0 ? 1 : right) ;
57
58 // "dif", "rem"
59 if( strncmp(op,ARITHMATIC_OPERATORS[4],3) == 0 ) return left>=right ? left-right : right-
left;
60 if( strncmp(op,ARITHMATIC_OPERATORS[5],3) == 0 ) return ((int)left) % ((int)right);
61
62 }
63 }
```

var_list.h

```
1 #include<stdbool.h>
2
3 #ifndef VAR_LIST_H
4 #define VAR_LIST_H
5
6 struct VARIABLE{
7     char name[30];
8     double value;
9     char type[10];
10    struct VARIABLE *prev;
11    struct VARIABLE *next;
12};
13
14 // create variable node using passed parameter
15 struct VARIABLE* createNode(const char *name, char *type, double value);
16
17 // insert variable into linked-list
18 void insertVariable(char *name, char *type, double val);
19
20 // return total number of variables
21 int getTotalVar();
22
23 // update variable with given val
24 void updateVariable(char *name, double val);
25
26 // delete variable from linked-list
27 void deleteVariable(char *name);
28
29 // return true if variable exists with passed name
30 bool doesVariableExists(char *name);
31
32 // return variable struct of given name or NULL if not found
33 struct VARIABLE* getVariable(char* name);
34
35 double getValueOrDefault(char* name);
36
37 char* getFormattedValueOrDefault(char *name);
38
39 // print all variable available when called
40 void printAll();
41
42 #endif
```

var_list.c

```
1 #include<stdio.h>
2 #include <stddef.h>
3 #include<string.h>
4 #include<stdlib.h>
5 #include "var_list.h"
6 #include <stdbool.h>
7
8 const int KEYS_SIZE = 25;
9 char KEYS[50][25] = {
10     "void", "int", "double", "float", "justInCase",
11     "println", "discard", "till", "import",
12     "static", "void", "entryPoint",
13     "lt", "gt", "eq", "neq", "le", "ge",
14     "add", "sub", "mul", "div", "dif", "rem"
15 };
16
17 struct VARIABLE *head = NULL;
18 struct VARIABLE *tail = NULL;
19
20 struct VARIABLE* createNode(const char *name, char *type, double value) {
21     struct VARIABLE *newNode = (struct VARIABLE*) malloc(sizeof(struct VARIABLE));
22     if(!newNode) {
23         printf("Memory allocation failed.\n");
24         return NULL;
25     }
26
27     strncpy(newNode->name, name, sizeof(newNode->name) - 1);
28     newNode->value = value;
29     strncpy(newNode->type, type, sizeof(newNode->type)-1);
30     newNode->prev = NULL;
31     newNode->next = NULL;
32     return newNode;
33 }
34
35 void insertVariable(char *name, char *type, double val) {
36
37     for(int i=0; i<KEYS_SIZE; i++){
38         if(strcmp(KEYS[i],name) == 0){
39             printf("Keyword '%s' can't be variable\n",name);
40             return;
41         }
42     }
43
44     struct VARIABLE *var = createNode(name,type,val);
45
46     if( strcmp("int",type,3) == 0 ){ // integer
47         val = (double)( (int)val ); // ignoring after decimal
48     }
49
50     if(tail == NULL) {
51         head = var;
52         tail = var;
53     }
```

```

54     else {
55         var->prev = tail;
56         tail->next = var;
57         tail = var;
58     }
59 }
60
61 int getTotalVar(){
62     struct VARIABLE *ptr;
63     int count=0;
64     ptr = head;
65     while(ptr != NULL){
66         count++;
67         ptr = ptr->next;
68     }
69     return count;
70 }
71
72 void updateVariable(char *name, double val){
73
74     struct VARIABLE *ptr;
75     ptr = head;
76     while(ptr != NULL){
77         if( strcmp(ptr->name,name) == 0 ){
78
79             if( strncmp("int",ptr->type,3) == 0 ) val = (int)val;
80
81             ptr->value = val;
82             break;
83         }
84         ptr = ptr->next;
85     }
86 }
87
88 void deleteVariable(char *name){
89     struct VARIABLE *ptr;
90     ptr = head;
91     while(ptr != NULL){
92         if( strcmp(ptr->name, name) == 0 ){
93             // first delete
94             if(ptr == head){
95                 //single node
96                 if(ptr == tail) tail = NULL;
97                 head = ptr->next;
98             }
99             else if(ptr == tail){ // last delete
100                 tail = tail->prev;
101                 tail->next = NULL;
102             }
103             else{
104                 ptr->prev->next = ptr->next;
105                 ptr->next->prev = ptr->prev;
106             }
107         }
108         ptr = ptr->next;
109     }

```

```

110 }
111
112 bool doesVariableExists(char *name){
113     struct VARIABLE *ptr = head;
114
115     while (ptr != NULL)
116     {
117         if( strcmp(name,ptr->name) == 0 ) return true;
118         ptr = ptr->next;
119     }
120     return false;
121 }
122
123 struct VARIABLE* getVariable(char* name){
124     //printf("printing from inner\n");
125     //printAll();
126     struct VARIABLE *ptr = head;
127     while (ptr != NULL)
128     {
129         //printf("(%s %ld),",ptr->name,ptr->value);
130         if( strcmp(name,ptr->name) == 0 ){
131             //printf("\n value returning %s %lf\n",ptr->name,ptr->value);
132             return ptr;
133         }
134         ptr = ptr->next;
135     }
136     return NULL;
137 }
138
139 double getValueOrDefault(char* name){
140     struct VARIABLE *ptr = head;
141     while (ptr != NULL)
142     {
143         if( strcmp(name,ptr->name) == 0 ){
144             return ptr->value;
145         }
146         ptr = ptr->next;
147     }
148     return 0;
149 }
150
151 char* getFormattedValueOrDefault(char *name){
152     struct VARIABLE* var = getVariable(name);
153     if(var == NULL){
154         return "0";
155     }
156
157     char *arr = (char *) malloc(20);
158
159     if ( strcmp(var->type,"int") == 0){
160         int num = (int)(var->value);
161         sprintf(arr, "%d", num);
162         return arr;
163     }
164
165     double num = (var->value);

```

```
166     sprintf(arr, "%lf", num);
167     return arr;
168 }
169
170 void printAll(){
171     printf("\n");
172     if(head == NULL) return;
173
174
175     printf("-----Printing all variables-----\n");
176
177     struct VARIABLE *ptr;
178     ptr = head;
179     while(ptr != NULL){
180         if( strcmp(ptr->type,"int") == 0){
181             printf("%s(%s) %d -> ",ptr->name,ptr->type,(int)ptr->value);
182         }
183         else{
184             printf("%s(%s) %lf -> ",ptr->name,ptr->type,ptr->value);
185         }
186         ptr = ptr->next;
187     }
188     printf("\n\n");
189 }
190 }
```

prototype_list.h

```
1 #include<stdbool.h>
2
3 #ifndef PROTOTYPE_LIST_H
4 #define PROTOTYPE_LIST_H
5
6 struct PARAMETER{
7     char type[10];
8     double value;
9     struct PARAMETER *next;
10    struct PARAMETER *prev;
11
12 };
13
14 struct PROTOTYPE{
15     char funcType[10];
16     char funcName[30];
17     char libraryName[30];
18
19     struct PARAMETER *paramsHead;
20     struct PARAMETER *paramsTail;
21     struct PROTOTYPE *prev;
22     struct PROTOTYPE *next;
23 };
24
25 // insert import name from full import line
26 void insertImport(char fulImp[20]);
27
28 // returns true if imp is found
29 bool isImportImported(char *imp);
30
31 // prints all included imports
32 void printAllImports();
33
34 // create and returns PARAMETER after creating using type and value
35 struct PARAMETER* createParameter(const char *type,double value);
36
37 // inserts parameter to passed head and tail after creating using type and val
38 void insertParameter( struct PARAMETER **head, struct PARAMETER **tail, char *type, double val );
39
40 // creates proto-type and save it in the list
41 struct PROTOTYPE* createProto(char *type, char *name, char *libraryName, struct PARAMETER *paramsHead, struct PARAMETER *paramsTail );
42
43 // insert proto type to library proto-type list
44 void insertLibraryProto(struct PROTOTYPE* var);
45
46 // prints all library function
47 void printAllLibraryFunction();
48
49 // insert user defined proto-type to list
50 void insertProto(struct PROTOTYPE* var);
```

```
52 // returns actual prototype from function call by user, isLibrary true to check library
53 // function, false to check user-defined
54 struct PROTOTYPE* getOriginalProto(struct PROTOTYPE* proto, bool isLibrary);
55
56 // checks if passed proto-type exists
57 bool doesProtoExists(struct PROTOTYPE* proto, bool isLibrary);
58
59 // prints all user-defined proto-type
60 void printAllProto();
61
62 // prints proto-type in formatted way, reverse to indicate the parameter order
63 void printProto(struct PROTOTYPE *ptr, bool reverse);
64
65 // returns result after performing library function
66 double getLibrayFunctionResult(char* name, struct PARAMETER* params);
67
68 // for adding library function
69 void initializeLibraryFunction();
70 #endif
```

prototype_list.c

```
45 void printAllImports(){
46     printf("\nPrinting all imports:\n");
47     for(int i=0; i<totalImport; i++){
48         printf("%s\n",imports[i]);
49     }
50 }
51 }
52
53 struct PARAMETER* createParameter(const char *type,double value){
54     struct PARAMETER *node = (struct PARAMETER*) malloc(sizeof(struct
55 PARAMETER));
56
57     if(!node) { printf("Memory allocation failed.\n"); return NULL; }
58
59     strncpy(node->type, type, sizeof(node->type) - 1);
60     node->value = value;
61
62     node->prev = NULL;
63     node->next = NULL;
64     return node;
65 }
66
67 void insertParameter( struct PARAMETER **head, struct PARAMETER **tail, char
68 *type, double val){
69
70     struct PARAMETER *var = createParameter(type,val);
71
72     if(*tail == NULL) {
73         *head = var;
74         *tail = var;
75     }
76     else {
77         var->prev = (*tail);
78         (*tail)->next = var;
79         (*tail) = var;
80     }
81 }
82
83 struct PROTOTYPE* createProto(char *type, char *name, char *libraryName,
84 struct PARAMETER *paramsHead, struct PARAMETER *paramsTail ) {
85
86     struct PROTOTYPE *newNode = (struct PROTOTYPE*) malloc(sizeof(struct
87 PROTOTYPE));
88
89     if(!newNode) {
90
91         bool isImportImported(char *imp){
92             for(int i=0; i<totalImport; i++){
93                 if( strcmp(imports[i], imp) == 0) return true;
94             }
95             return false;
96         }
97
98     }
99 }
```

```

89     printf("Memory allocation failed.\n");
90     return NULL;
91 }
92
93 //printf("Inside-3 %s \n",type);
94 strcpy(newNode->funcType, type, sizeof(newNode->funcType) - 1);
95
96 //printf("Inside-4\n");
97 strcpy(newNode->funcName, name , sizeof(newNode->funcName) -1);
98
99 strcpy(newNode->libraryName, libraryName , sizeof(newNode->libraryName)
-1);
100 //printf("Inside-5\n");
101 newNode->paramsHead = paramsHead;
102 newNode->paramsTail = paramsTail;
103
104 newNode->prev = NULL;
105 newNode->next = NULL;
106
107 //printf("Inside - 6\n");
108
109 // printf("inside create: ");
110 // printProto(newNode);
111
112 return newNode;
113
114 }
115
116 void insertLibraryProto(struct PROTOTYPE* var) {
117
118 if(libraryProtoTail == NULL) {
119     libraryProtoHead = var;
120     libraryProtoTail = var;
121 }
122 else {
123     var->prev = libraryProtoTail;
124     libraryProtoTail->next = var;
125     libraryProtoTail = var;
126 }
127
128
129 void printAllLibraryFunction(){
130     printf("\nAll library functions are:\n");
131     struct PROTOTYPE *ptr = libraryProtoHead;
132
133     while( ptr != NULL ){
134         printProto(ptr, false);
135     }
136 }
137 }
138
139 void insertProto(struct PROTOTYPE* var) {
140
141     if(protoTail == NULL) {
142         protoHead = var;
143         protoTail = var;
144     }
145     else {
146         var->prev = protoTail;
147         protoTail->next = var;
148         protoTail = var;
149     }
150 }
151
152 struct PROTOTYPE* getOriginalProto(struct PROTOTYPE* proto, bool isLibrary){
153
154     struct PROTOTYPE *ptr = isLibrary ? libraryProtoHead : protoHead;
155
156     while (ptr != NULL)
157     {
158         if(isNameSame = strcmp( proto->funcName ,ptr->funcName ) == 0 ?
159             true : false;
160
161         if(isNameSame){
162             struct PARAMETER *param1 = proto->paramsHead;
163             struct PARAMETER *param2 = ptr->paramsTail;
164
165             while ( param1 != NULL && param2 != NULL )
166             {
167                 if( strcmp(param2->type, "any") == 0 ) {}
168                 else if( strcmp(param1->type, param2->type) != 0 ){
169                     break;
170                 }
171             }
172             param1 = param1->next;
173             param2 = param2->next;
174         }
175
176     if(param1 == NULL && param2 == NULL){
177         return ptr;
178     }
179 }
180

```

```

181     }
182     ptr = ptr->next;
183     return NULL;
184 }
185 }
186 bool doesProtoExists(struct PROTOTYPE* proto, bool isLibrary){
187     return getOriginalProto(proto,isLibrary) != NULL;
188 }
189 }
190 void printAllProto(){
191     struct PROTOTYPE *ptr = protoHead;
192     while (ptr != NULL)
193     {
194         printProto(ptr,true);
195         ptr = ptr->next;
196     }
197 }
198 }
199 }
200 }
201 // prints data also if reverse is false
202 void printProto(struct PROTOTYPE *ptr, bool reverse){
203     printf("%s %s(%s", ptr->funcType, ptr->funcName);
204     struct PARAMETER* param = reverse ? ptr->paramsTail : ptr->paramsHead;
205     while(param != NULL){
206         printf("%s",param->type);
207         param = reverse ? param->prev : param->next;
208         printf("\n");
209     }
210     printf("%s",param->value);
211     //if(!reverse) printf("'%lf'",param->value);
212     if( (reverse ? param->prev : param->next) != NULL){ printf(","); }
213     param = reverse ? param->prev : param->next;
214     printf("\n");
215     printf("%s",param->value);
216     printf("%s",param->value);
217     printf("\n");
218     printf("%s",param->value);
219 }
220 }
221 double getLibraryFunctionResult(char* name, struct PARAMETER* params){
222     struct PARAMETER *head = NULL, *tail = NULL;
223     if( strcmp(name, "max") == 0 ){
224         return fmax(params->value, params->next->value);
225     }
226 }
227 void initializeLibraryFunction(){
228     if( strcmp(name, "sqrt") == 0 ){
229         if(params->value < 0){
230             printf("\nStay in real world\n");
231             return 0;
232         }
233         return sqrt(params->value);
234     }
235     if( strcmp(name, "scanInt") == 0 ){
236         printf("\nTaking int input 0\n");
237         return 0;
238     }
239     if( strcmp(name, "scan") == 0 ){
240         printf("\nTaking input 0.0\n");
241         return 0;
242     }
243     if( strcmp(name, "toInt") == 0 ){
244         return (int)(params->value);
245     }
246     if( strcmp(name, "toFloat") == 0 ){
247         return (float)(params->value);
248     }
249     if( strcmp(name, "toDouble") == 0 ){
250         return (double)(params->value);
251     }
252     if( strcmp(name, "show") == 0 ){
253         printf("From show function: %lf \n",params->value);
254     }
255     if( strcmp(name, "libraryFunction") == 0 ){
256         return 0;
257     }
258     if( strcmp(name, "library") == 0 ){
259         printf("From library function: %lf \n",params->value);
260         return 0;
261     }
262 }
263 return 0;
264 }
265 }
266 void initializeLibraryFunction();
267
268 struct PARAMETER *head = NULL, *tail = NULL;
269
270 //scanfInt stdio
271 {
272     head = NULL; tail = NULL;
273 }
274

```

```

275     struct PROTOTYPE* scanInt = createProto("int", "scanf", "stdio", head,
276     insertLibraryProto(scanInt);
277   }
278
279   //scan stdio
280   {
281     head = NULL; tail = NULL;
282     struct PROTOTYPE* scan = createProto("float", "scanf", "stdio", head, tail);
283     insertLibraryProto(scan);
284   }
285
286   // show stdio
287   {
288     head = NULL; tail = NULL;
289     insertParameter(&head, &tail, "any", -1);
290     struct PROTOTYPE* show = createProto("void", "show", "stdio", head, tail);
291     insertLibraryProto(show);
292   }
293
294   // max math
295   {
296     head = NULL; tail = NULL;
297     insertParameter(&head, &tail, "any", -1);
298     insertParameter(&head, &tail, "any", -1);
299
300     struct PROTOTYPE* max = createProto("float", "max", "math", head, tail);
301     insertLibraryProto(max);
302   }
303
304   // sqrt math
305   {
306     head = NULL; tail = NULL;
307     insertParameter(&head, &tail, "any", -1);
308     struct PROTOTYPE* sqrt = createProto("double", "sqrt", "math", head, tail);
309     insertLibraryProto(sqrt);
310   }
311
312   //toInt converter
313   {
314     head = NULL; tail = NULL;
315     insertParameter(&head, &tail, "any", -1);
316     struct PROTOTYPE* toInt = createProto("int", "toInt", "converter", head,
317     tail);
318     insertLibraryProto(toInt);
319   }

```

final.1

```
49     stack[stackSize] = id;
50     return id;
51 }
52
53 #include<stdio.h>
54 #include<string.h>
55 #include <stdlib.h>
56 #include <stdbool.h>
57 #include <ctype.h>
58 #include "final.tab.h"
59
60 // header, single, multi, var, print, dis, calc_val, if, till
61 int tempCounter[10];
62
63 int blockBalancer = 0;
64 int stack[100];
65 int stackSize = 0;
66
67 const int MAXIMUM_VAR_SIZE = 10;
68 char error[200];
69
70 // extern
71 char lastDataType[10];
72 char *outBuffer = NULL;
73 int outBufferSize = 0;
74 bool isLastIfValid = false;
75 //extern
76
77 int commentCounter = 0;
78 int comment_depth = 0;
79 char *comment_buffer = NULL;
80
81 bool canDeclareHeader = true;
82 bool isMainBlockFound = false;
83 bool isCommingFromMain = false;
84 bool isCommingFromVar = false;
85
86 size_t buffer_length = 0;
87
88 int bracketCounter=0;
89
90 char *removeRedundant(char* );
91 void insertImport(char* );
92
93
94 // variable
95 const int MAXIMUM_VARIABLE_LENGTH = 200;
96
97 int pushState(int id){
98     stackSize++;
```

```

99
}
100 void stopProgram(char *error){
101     print("%s\n", error);
102     free(comment_buffer);
103     exit(1);
104 }
105
106 void checkForEnd(){
107     if (comment_depth == 0) {
108         BEGIN_INITIAL;
109         process_comment();
110         resetBuffer();
111     }
112     else if (comment_depth < 0){
113         resetBuffer();
114         stopProgram("Invalid comment found");
115     }
116 }
117
118 void initMain(){
119     canDeclareHeader = false;
120     //printf("execution started\n");
121 }
122
123 void processHeader(){
124     if(!canDeclareHeader){
125         stopProgram("Header must be at top");
126     }
127     return;
128 }
129
130
131 void initVarSec(char *temp){
132     char *type = removeRedundant(temp);
133     strcpy(lastDataType, type);
134     canDeclareHeader = false;
135 }
136
137 void sendNumber(char *text){
138     double num = 0;
139     sscanf(text, "%lf", &num);
140     yyval.num = num;
141     //printf("===== %f =====", num);
142 }
143
144 void initOutBuffer(){
145     outBuffer = (char*) malloc(1);
146     memset(outBuffer, 0, 1);
147     outBufferSize = 1;
148 }
149
150 char* removeOp(char *val){
151     char *op = malloc(14*sizeof(char));
152     char *temp = strdup(val);
153     strcpy(op, temp+1, strlen(temp) - 2 );
154     return op;
155 }
156 }
157
158 %}
159 % COMMENT
160 % MAIN
161 % VAR_SEC
162 % IF_SEC
163 % IGNORE_SEC
164 % DISCARD_SEC
165 % LOOP_SEC
166 % LOOP_BODY_SEC
167 % OUT_SEC
168 % PROTO_SEC
169 % FUNC_SEC
170 % FUNC_SEC_VAR
171
172 DQ \
173 NUMBER [-]?([0-9]+[.][0-9]+)[[-]?([0-9]+)|([.][0-9]+)
174 VARIABLE [a-zA-Z][a-zA-Z0-9]*
175 VAR_NUM {NUMBER} {VARIABLE}
176 COND_OP ("<lt>"|"<gt>"|"<eq>"|"<neq>"|"<le>"|"<ge>")
177 ARITH_OP ("<add>"|"<sub>"|"<mul>"|"<div>"|"<dif>"|"<rem>")
178 HEADER "import "[a-zA-Z]+";
179
180 OUT_START "println("[ "
181 OUT_BODY_CONST "[DQ][^"]*[DQ]"
182 OUT_BODY_VAR "{VAR_NUM}"
183 OUT_SEP "[ ]*[ \"|\" ][ ]*"
184 OUT_END "[ ]*")"[ ]*[ ]*
185
186 MAIN_START "static void entryPoint"[ ]*("[" ][ ]*{"[ "]*"
187 MAIN_END "]"
188 SINGLE_LINE_COMMENT("//").*(\n)?
189
190 VARIABLE_ONLY [ ]*{VARIABLE}*[*{NUMBER}, ]
191 VARIABLE_VALUE_ASSIGN_CONST [ ]*{VARIABLE}*[*{NUMBER}, ]
192 VARIABLE_VALUE_ASSIGN_VAR [ ]*{VARIABLE}*[*{NUMBER}, ]
193 VARIABLE_VALUE_ASSIGN_CALC [ ]*{VARIABLE}*[*{NUMBER}, ]
194 VARIABLE_VALUE_ASSIGN [ ]*{VARIABLE}*[*{NUMBER}, ]
195 VARIABLE_VALUE_ASSIGN_EQ [ ]*{VARIABLE}*[*{NUMBER}, ]
196 VARIABLE_VALUE_ASSIGN_EQ_EQ [ ]*{VARIABLE}*[*{NUMBER}, ]
197 VARIABLE_VALUE_ASSIGN_EQ_EQ_EQ [ ]*{VARIABLE}*[*{NUMBER}, ]
198 VARIABLE_VALUE_ASSIGN_EQ_EQ_EQ_EQ [ ]*{VARIABLE}*[*{NUMBER}, ]

```

```

199 DISCARD_START "discard"
200
201 <DTSCARD_SEC> /* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
202 <LOOP_SEC> /* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
203 <LOOP_BODY_SEC> /* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
204 <OUT_SEC> /* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
205 <PROTO_SEC> /* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
206 <FUNC_SECS> /* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
207 <FUNC_SEC_VARS> /* { initMultiComment(); BEGIN( pushState(COMMENT) ); }
208 <COMMENT> /* { appendTextToBuffer("//"); }
209 <COMMENT> /* { appendTextToBuffer("//"); }
210 <IF> "justInCase" [ ]
211 IF_BODY_START "{"
212 IF_BODY_END [ ]*;""
213 IF_SPACE [ ]*
214 IGNORE_LEFT_BRACE "{"
215 IGNORE_RIGHT_BRACE "}"
216
217 FUNC_START "@'{VARIABLE}'
218
219 LOOP_START "till"
220 LOOP_START_BRACE "{"
221 LOOP_OTHERS [^{}]\n*"\n
222 LOOP_END {IF_BODY_END}
223
224 NEW_LINE_AND_TAB [\n\t]*
225 /**
226 {*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
227 %%}
228 {*" { inc(1); initSingleComment(); }
229 {*" { inc(1); initSingleComment(); }
230 {*" { initMultiComment(); BEGIN( pushState(COMMENT) ); }
231
232 {*" { inc(1); initSingleComment(); }
233 {*" { inc(1); initSingleComment(); }
234 {*" { inc(1); initSingleComment(); }
235 {*" { inc(1); initSingleComment(); }
236 {*" { inc(1); initSingleComment(); }
237 {*" { inc(1); initSingleComment(); }
238 {*" { inc(1); initSingleComment(); }
239 {*" { inc(1); initSingleComment(); }
240 {*" { inc(1); initSingleComment(); }
241 {*" { inc(1); initSingleComment(); }
242 {*" { inc(1); initSingleComment(); }
243 {*" { inc(1); initSingleComment(); }
244 {*" { inc(1); initSingleComment(); }
245 {*" { inc(1); initMultiComment(); BEGIN( pushState(COMMENT) );
246 {*" { inc(1); initMultiComment(); BEGIN( pushState(COMMENT) );
247 {*" { inc(1); initMultiComment(); BEGIN( pushState(COMMENT) );
248 {*" { inc(1); initMultiComment(); BEGIN( pushState(COMMENT) );
249 {*" { inc(1); initMultiComment(); BEGIN( pushState(COMMENT) );

```

```

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

352 <FUNC_SEC>; { {
353     BEGIN( popState() );
354     return *yytext;
355 }
356 }
357 <MAIN>{VARIABLE} {
358     yyval.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     yyval.name = removeOp(yytext);
370     return ARITH_OPE;
371 }
372 <VAR_SEC>{VARIABLE} {
373     yyval.name = strdup(yytext);
374     return VAR;
375 }
376 <VAR_SEC>; { {
377     yyval.name = strdup(yytext);
378 }
379 <MAIN>{DISCARD_START} {
380     BEGIN( popState() );
381     return *strdup(yytext);
382 }
383 <VAR_SEC>{VAR_SPACE} {}
384 }
385 <MAIN>{DISCARD_START} {
386     //printf("Discard section\n");
387     inc(5);
388     BEGIN( pushState(DISCARD_SEC) );
389     return DISCARD;
390 }
391 <DISCARD_SEC>{VARIABLE} {
392     //printf("\nFrom discard only:-%s-\n",yytext);
393     yyval.name = strdup(yytext);
394     return VAR;
395 }
396 <DISCARD_SEC>; { {
397     yyval.name = strdup(yytext);
398     <DISCARD_SEC>; { {
399     BEGIN( popState() );
400     <OUT_SEC>{OUT_BODY_CONST} {
401         yyval.name = strdup(yytext);
402     }
403 }

```

```

403     }  
404     return OUTPUT_VC;  
405     }  
406     <OUT_SEC>{OUT_BODY_VAR} {  
407       yyval.name = strdup(yytext);  
408       return OUTPUT_VC;  
409     }  
410     <OUT_SEC>{OUT_SEP} {  
411       return OUTPUT_SEP;  
412     }  
413     <OUT_SEC>{OUT_END} {  
414       inc(4);  
415       BEGIN( popState() );  
416       return OUTPUT_END;  
417     }  
418     <IF_SEC>{("["}) { return *yytext; }  
419     <IF_SEC>{"]"} { return END_POINT; }  
420  
421     <IF_SEC>{COND_OP} { yyval.name = removeRp(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       //popState(); / 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++;  
433         BEGIN( pushState(IFIGNORE_SEC) );  
434       }  
435       else{  
436         BEGIN(pushState(MAIN));  
437       }  
438       return *yytext;  
439     }  
440     <IF_SEC>{NEW_LINE_AND_TAB} {  
441       IGNORE_SEC>{IGNORE_LEFT_BRACE} { bracketCounter++; }  
442       IGNORE_SEC>{IGNORE_RIGHT_BRACE} {  
443         bracketCounter--;  
444       }  
445       if(bracketCounter == 0){  
446         BEGIN( popState() );  
447         return END_POINT;  
448       }  
449     }  
450     IGNORE_SEC>["{}"] {  
451   }  
452     <LOOP_SEC>("(|") { return *yytext; }  
453

```

final.y

```
45   for(int i=0; i<len; i++){  
46     /printf("%d, ", source[i]);  
47     if( source[i] == ' ' ||  
48       source[i] == ',' ||  
49       source[i] == ';' ||  
50       source[i] == '\t' ||  
51       source[i] == '\n' ||  
52       source[i] == 32 )  
53     )continue;  
54   temp[j] = source[i];  
55   j++;  
56 }  
57  
58 temp[j] = '\0';  
59 char *dest;  
60 dest = malloc(j*sizeof(char));  
61 strcpy(dest,temp);  
62 //printf("\n%sn",dest);  
63 return dest;  
64 }  
65  
66 void printAndClearOutBuffer(){  
67   printf("\n%sn", outBuffer);  
68   free(outBuffer);  
69   outBufferSize = 0;  
70 }  
71  
72  
73 void continueOutBuffer(char *vc){  
74   int i=0, len = strlen(vc) - 1;  
75   while( i < len && vc[i] == ' ') i++; // leading space  
76   while( len>0 && vc[len] == ' ') len--; // trailing space after '  
77   while( len>0 && vc[len] == '\n') len--; // space  
78   char *res;  
79  
80   if(vc[i] == '\n'){ // constant  
81     i++; // trailing quotaion bad dilam  
82   }  
83   int size = len-i+1;  
84   res = (char *)malloc(size+1);  
85   strcpy(res, vc + i, size);  
86   res[size] = '\0';  
87 }  
88 else{ // variable  
89   vc = removeRedundant(vc);  
90   STRUCT VARIABLE *var = getVariable(vc);  
91 }
```

```

93     if( !doesVariableExists(vc) ) {
94         printf("\nVariable %s doesn't exist\n",vc);
95     }
96     char* val = getFormattedValueOrDefault(vc);
97     res = (char *) malloc(20);
98     sprintf(res, "%s ", val);
99 }
100
101 outBufferSize += strlen(res);
102 outBuffer = (char *)realloc(outBuffer,outBufferSize);
103
104 strcat(outBuffer,res);
105 }
106
107 void initProto(char *paramType){
108     //printf("type found: %s\n",paramType);
109     // -1 default value, not needed for proto-type
110     //insertParameter(&paramHead, &paramTail, paramType,-1);
111 }
112
113 void processProto(char *funcType, char *funcName){
114     //printf("Inside\n");
115
116     struct PROTOTYPE* proto = createProto(funcType, funcName, "user-defined",
117     paramHead,paramTail);
118     //printf("Inside-2\n");
119     paramHead = NULL; paramTail = NULL;
120
121     if( doesProtoExists(proto,red) ){
122         printf("\nDuplicate proto-type found\n");
123         return;
124     }
125
126     insertProto(proto);
127     printf("\nPrototype inserted: ");
128     printProto(proto,true);
129 }
130
131
132
133
134 double doArithOperation(double val1, double val2, char* op){
135     //printf("%lf %lf %s -----%kuttar baccha-\n",val1,val2,op);
136     return getValue(val1, val2, op);
137 }
138
139     bool addVariable(char* name,char *type){
140         if( doesVariableExists(name) ){
141             printf("\nVariable %s is already defined\n",name);
142             return false;
143         }
144         insertVariable(name,type,0);
145         return true;
146     }
147
148     void declareVariable(char *name, char *type, double val){
149         if( addVariable(name, type ) ){
150             updateVariable(name,val);
151         }
152     }
153
154     void assignIfPossible(char *name, double val){
155         if( !doesVariableExists(name) ){
156             printf("\nVariable %s doesn't exist\n",name);
157             return;
158         }
159     }
160
161     updateVariable(name,val);
162 }
163
164     void discardVariable(char *name){
165         if( !doesVariableExists(name) ){
166             printf("\nCan't discard %s since not found.\n",name);
167             return;
168         }
169     }
170     printf("\nDiscarded variable %s\n",name);
171     deleteVariable(name);
172 }
173
174     bool isNumber(char* num){
175         bool count = 0;
176         int i = num[0] == '-' ? 1 : 0;
177
178         for(; i<strlen(num); i++){
179             if( num[i] == '.' && count == 0 ) { count=1; continue; }
180             if( !isdigit(num[i]) ) return false;
181         }
182         return true;
183     }
184 }
```

```

185     bool evaluateCondition(char* leftChar, char* op, char *rightChar){
186         int counter = 0;
187         printf("\n");
188         while( evaluateCondition( lastCond.left, lastCond.op, lastCond.right
189             ) ){
230             printf("Iterating - %d", counter);
231             printf("\n");
232         }
233         printf("Iterating - %d", counter);
234         if( !isNumber(lastCond.left) ){
235             if( !isNumber(lastCond.left) ){
236                 printf("%s", value);
237                 getFormattedValueOrDefault(lastCond.left);
238             }
239             printf("\n");
240             double val1 = 0, val2 = 0;
241             if( isNumber(lastAssign.left) ){
242                 sscanf(lastAssign.left, "%lf", &val1);
243             }
244             else{
245                 val1 = getValueOrDefault(lastAssign.left);
246             }
247         }
248         if( isNumber(lastAssign.right) ){
249             sscanf(lastAssign.right, "%lf", &val2);
250         }
251         else{
252             val2 = getValueOrDefault(lastAssign.right);
253         }
254         if( !doesVariableExists(rightChar) ){
255             if( !doesVariableExists(rightChar) ){
256                 printf("%s doesn't exist. Using default value(0)
257 \n", rightChar);
258             }
259             double res = doArithOperation(val1, val2, lastAssign.op);
260             double res = doArithOperation(val1, val2, lastAssign.op);
261             updateVariable( lastAssign.var, res );
262             updateVariable( lastAssign.var, res );
263             counter++;
264             Sleep(200);
265         }
266     }
267     void addTypeToCall(char *name){
268         //printf("Type call %s\n", name);
269         char *type = "double";
270         return type;
271     }
272     double val = 0;
273 }
```

```

274 // check library function
275 if( doesProtoExists(temp, true) ){
276     // returns from library function
277     struct PROTOTYPE* orig = getOriginalProto(temp, true);
278 }
279 else{
280     printf("\nCalling function ");
281     printProto( orig, true);
282
283     if( doesVariableExists(name) ){
284         struct VARIABLE* var = getVariable(name);
285         type = var->type;
286         val = var->value;
287     }
288     else{
289         printf("\nvariable %s doesn't exists\n",name);
290     }
291     insertParameter(&callParamHead, &callParamTail, type, val);
292
293 }
294
295 double getDefaultResultFromFunction(
296     char* name, struct PARAMETER* head,
297     struct PROTOTYPE* functoCall )
298 {
299
300     return getLibraryFunctionResult(name, head);
301
302
303
304     void processCall(char* funcName){
305         char realName[ strlen(funcName) ];
306
307         // Copy the string starting from the second character
308         strcpy(realName, funcName + 1);
309
310         struct PROTOTYPE* temp = createProto("void",realName, "NULL",
311                                         callParamHead,NULL);
312
313         strcpy( lastFuncRes.type, "null" );
314         lastFuncRes.res = 0;
315
316     if( !doesProtoExists(temp, false) ){
317
318
319
320     // check library function
321     if( doesProtoExists(temp, true) ){
322         // returns from library function
323         struct PROTOTYPE* orig = getOriginalProto(temp, true);
324
325         printf("\nCalling function ");
326         printProto( orig, true);
327
328         if( !isImported(orig->libraryName) ){
329             printf("Warning - library %s is not imported\n", orig->
330                   libraryName);
331
332             strcpy( lastFuncRes.type, orig->funcType );
333             lastFuncRes.res = getOrDefaultResultFromFunction(realName,
334
335             callParamHead,orig);
336             printf("\nFunction not found\n");
337         }
338
339     }
340     else{
341         printf("\nCalling function ");
342         struct PROTOTYPE* orig = getOriginalProto(temp, false);
343         strcpy( lastFuncRes.type, orig->funcType);
344         printProto( orig, true);
345     }
346     callParamHead = NULL;
347     callParamTail = NULL;
348
349     void assignFromFunction(char* name, char* varType, bool update){
350         double val = 0;
351         if( strcmp("void",lastFuncRes.type,4) == 0 ){
352             printf("\nInvalid assignment from void to %s\n",lastDataType);
353             val = 0;
354         }
355         else if( strcmp(varType, lastFuncRes.type) == 0 ){
356             val = lastFuncRes.res;
357         }
358         else if( strcmp("int", lastDataType, 3) != 0 ){
359             val = lastFuncRes.res;
360
361         else if( strcmp("int", lastDataType, 3) == 0 ){
362             val = (int)lastFuncRes.res;
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
8010
8011
8012
8013
8014
8015
8016
8017
8018
8019
8020
8021
8022
8023
8024
8025
8026
8027
8028
8029
8030
8031
8032
8033
8034
8035
8036
8037
8038
8039
8040
8041
8042
8043
8044
8045
8046
8047
8048
8049
8040
8051
8052
8053
8054
8055
8056
8057
8058
8059
8050
8061
8062
8063
8064
8065
8066
8067
8068
8069
8060
8071
8072
8073
8074
8075
8076
8077
8078
8079
8070
8081
8082
8083
8084
8085
8086
8087
8088
8089
8080
8091
8092
8093
8094
8095
8096
8097
8098
8099
8090
80100
80101
80102
80103
80104
80105
80106
80107
80108
80109
80100
80111
80112
80113
80114
80115
80116
80117
80118
80119
80110
80121
80122
80123
80124
80125
80126
80127
80128
80129
80110
80131
80132
80133
80134
80135
80136
80137
80138
80139
80110
80140
80141
80142
80143
80144
80145
80146
80147
80148
80110
80149
80150
80151
80152
80153
80154
80155
80156
80157
80110
80158
80159
80160
80161
80162
80163
80164
80165
80166
80110
80167
80168
80169
80170
80171
80172
80173
80174
80110
80175
80176
80177
80178
80179
80180
80181
80182
80110
80183
80184
80185
80186
80187
80188
80189
80110
80190
80191
80192
80193
80194
80195
80196
80197
80110
80198
80199
80120
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80120
80199
80121
80122
80123
80124
80125
80126
80127
80128
80129
80120
80130
80131
80132
80133
80134
80135
80136
80137
80138
80120
80139
80140
80141
80142
80143
80144
80145
80146
80147
80148
80120
80149
80150
80151
80152
80153
80154
80155
80156
80157
80158
80120
80159
80160
80161
80162
80163
80164
80165
80166
80167
80168
80120
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80120
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80120
80189
80190
80191
80192
80193
80194
80195
80196
80197
80
```

```

365     printf("\nIncompatible assignment. Using default value...\n");
366 }
367
368     if(update){
369         assignIfPossible(name, val);
370     } else{
371         declareVariable(name, lastDataType, val);
372     }
373
374     void continueToAssignFromFunction(char *name, bool update){
375
376         if(doesVariableExists(name)){
377             struct VARIABLE* var = getVariable(name);
378             assignFromFunction(name,var->type,update);
379         } else{
380             printf("\nVariable %s doesn't exists");
381         }
382     }
383
384     if(endsWith(name)){
385         printf("\nVariable %s ends with %c");
386     }
387
388     if(var->type == VAR_TYPE_FUNC){
389         if(var->type == VAR_TYPE_FUNC){
390             %union{
391                 double num;
392                 char *name;
393                 bool myBool;
394             }
395         }
396     }
397
398     %error-verbose
399     %token ENTRY_POINT END_POINT
400     %token DATA_TYPE FUNC_TYPE VAR NUMBER ARITH_OPE DISCARD
401     %token FUNC_NAME
402
403     %token OUTPUT_OUTPUT_VC OUTPUT_SEP OUTPUT_END
404     %token JUST_IN_CASE_COND_OPE OR AND VAR_CON TILL
405
406     // defining token type
407     //%%type<name> DATA_TYPE VAR FUNC_TYPE FUNC_NAME ARITH_OPE DISCARD
408
409     %%type<name> DATA_TYPE VAR FUNC_TYPE FUNC_NAME ARITH_OPE DISCARD
410     %%type<name> OUTPUT_VC OUTPUT_SEP
411     %%type<name> VAR_CON COND_OPE
412
413     %type<name> NUMBER arith_exp
414     %type<myBool> many_logic_cond logic_cond
415
416     %%program: many_proto_type ENTRY_POINT block END_POINT
417     %%executed");
418
419     ;
420
421     many_proto_type:
422     | many_proto_type FUNC_TYPE FUNC_NAME `(` many_type `)` {
423         processProto($2,$3);
424     }
425
426     many_type:
427     | FUNC_TYPE many_type { initProto($1); }
428     | FUNC_TYPE `.` many_type { initProto($1); }
429     ;
430
431     block: {}
432     | JUST_IN_CASE_if_sec_block {}
433     | TILL_loop_sec_block {}
434     | func_call `.` block {}
435     | var_declare block {}
436     | var_assign block {}
437     | DISCARD_discard_var block {}
438     | OUTPUT_out_sec_block {}
439
440     ;
441
442     func_call: FUNC_NAME `(` many_var_con `)` {
443         processcall($1); $$ = lastFuncRes.type;
444     }
445
446     many_var_con:
447     | many_var_con VAR_CON { addTypeToCall($2); }
448     | many_var_con `.` VAR_CON { addTypeToCall($3); }
449     ;
450
451     var_declare: DATA_TYPE others`;
452     ;
453     others: VAR {
454         declareVariable($1, lastDataType, 0);
455     }
456     | VAR `.` others {
457         declareVariable($1, lastDataType, 0);

```

```

458     }
459     | VAR '=' NUMBER {
460       declareVariable($1, lastDataType, $3);
461     }
462     | VAR '=' VAR {
463       double val = getValueOrDefault($3);
464       declareVariable($1, lastDataType, val);
465     }
466     | VAR '=' func_call {
467       assignFromFunction($1, lastDataType, false);
468     }
469     | VAR '=' func_call ',' others {
470       assignFromFunction($1, lastDataType, false);
471     }
472     | VAR '=' NUMBER ',' others {
473       declareVariable($1, lastDataType, $3);
474     }
475     | VAR '=' VAR ',' others {
476       double val = getValueOrDefault($3);
477       declareVariable($1, lastDataType, val);
478     }
479     | VAR '=' arith_exp {
480       declareVariable($1, lastDataType, $3);
481     }
482     | VAR '=' arith_exp ',' others {
483       declareVariable($1, lastDataType, $3);
484     }
485   }
486   var_assign: VAR '=' NUMBER ',' {
487     assignIfPossible($1,$3);
488   }
489   | VAR '=' VAR ',' {
490     if( !doesVariableExists($3) ){
491       printf("\nVariable %s doesn't exist\n", $3);
492     }
493   }
494   else{
495     double val = getValueOrDefault($3);
496     assignIfPossible($1, val);
497   }
498   | VAR '=' func_call ',' {
499     continueToAssignFromFunction($1,true);
500   }
501   | VAR '=' arith_exp ',' {
502     assignIfPossible($1, $3);
503   }
504 }

505   ;
506   discard_var: VAR ';' {
507     discardVariable($1);
508   }
509   | VAR ',' discard_var {
510     discardVariable($1);
511   }
512 }

513 out_sec: out_vc OUTPUT_END { printAndClearOutBuffer(); }
514 out_sec: out_vc OUTPUT_END { printAndClearOutBuffer(); }
515 ;
516 out_vc: OUTPUT_VC { continueOutBuffer($1); }
517   | out_vc OUTPUT_SEP OUTPUT_VC { continueOutBuffer($3); }
518 ;

519 if_sec: '(' many_logic_cond ')' '{' block END_POINT { printf("\nif
520 processed\n");
521   ;
522 loop_sec: '(' logic_cond ')' '{' loop_updater END_POINT {
523   printf("\nLoop matched\n");
524   startLoop();
525   ;
526 }

527 ;
528 loop_updater: VAR '=' arith_exp ';' {
529   lastAssign.var = $1;
530   //printf("== %s %s ==", lastCond.left, lastCond.op,
531   lastCond.right);
532   //printf("== %s %s %s ==", lastAssign.var, lastAssign.left,
533   );
534   ;
535 many_logic_cond: logic_cond { $$ = $1; isLastIfValid=$$; }
536   | many_logic_cond_OR logic_cond { $$ = $1 || $3; isLastIfValid=$$; }
537   | many_logic_cond_AND logic_cond { $$ = $1 && $3; isLastIfValid=$$; }
538   ;
539 }

540 logic_cond: VAR_CON COND_OPE VAR_CON {
541   $$ = evaluateCondition($1, $2, $3);
542   //printf("-----%s %s %d\n", $1, $2, $3, $$);
543   ;
544 }

545 arithmetic_exp: VAR_ARITH_OPE VAR {
546   double val1 = getDefaultValueOrDefault($1);
547   arithmetic_val = VAR_ARITH_OPE_VAR {
548

```

596 |

```
549     double val2 = getValueOrDefault($3);
550     updateLast($1, $2, $3);
551     $$ = doArithOperation(val1, val2, $2);
552 }
553 | VAR ARITH_OPE NUMBER {
554     double val1 = getValueOrDefault($1);
555     updateLast($1, $2, dtoc($3) );
556     $$ = doArithOperation(val1, $3, $2);
557
558 }
559 | NUMBER ARITH_OPE VAR {
560     double val2 = getValueOrDefault($3);
561
562     updateLast( dtoc($1), $2, $3);
563     $$ = doArithOperation($1, val2, $2);
564
565 | NUMBER ARITH_OPE NUMBER {
566     updateLast( dtoc($1), $2, dtoc($3) );
567     $$ = doArithOperation($1, $3, $2);
568
569 }
570 ;
571
572
573
574 %%
575
576 void yyerror(char *s)
577 {
578     fprintf(stderr, "\n%s", s);
579 }
580
581 int main(){
582     initializeLibraryFunction();
583
584     freopen("input.txt", "r", stdin);
585     freopen("output.txt", "w", stdout); // output in file
586
587     yyparse();
588     printAll();
589     printf("\nPrinting all prototype\n");
590     printAllProto();
591     printAllLibraryFunction();
592     printAllImports();
593
594     return 0;
595 }
```

output.txt

Captured comment: /* condition */

If condition is True
Actually executed since 10.000000 < 100.000000 < - - - - -
If condition is True
Also executed < - - - - -
if processed
If condition is False
if processed
if processed
----- separator ----- < - - - - -
If condition is True
5 < 10 and 10.000000 < 15 < - - - - -
if processed
If condition is False
if processed
Loop matched
Iterating - 0, value: 0
Iterating - 1, value: 15
Iterating - 2, value: 30
Iterating - 3, value: 45
Iterating - 4, value: 60
Iterating - 5, value: 75
Iterating - 6, value: 90

Captured comment: /* library function */

Calling function int scanInt()
Taking int input 0
0 < - - - - -
Calling function int scan()
Value after scan is: 0.000000 < - - - - -
Discarded variable f
Discarded variable n
Calling function void show(any)
From show function: 24.000000
Calling function float max(any,any)
Calling function void show(any)
From show function: 5.000000
Calling function double sqrt(any)
Square root of 42 is :6.480741 (float) < - - - - -
Calling function double sqrt(any)
Stay in real world
Square root of -42 is :0 (int) < - - - - -

```
Calling function int toInt(any)
Warning - library converter is not imported

6.480741 becomes 6 after toInt < -----
Calling function float toFloat(any)
Warning - library converter is not imported

6 becomes 6.000000 after toFloat < ----

Calling function double toDouble(any)
Warning - library converter is not imported

6.000000 becomes 6.000000 after toDouble < ----

Single line comment: // user defined function

Calling function int add(int,int,float)

0 < ----

Calling function void add(int,float,double)

Calling function void add(int,float,double)

Invalid assignment from void to int

Function not found
program executed
-----Printing all variables-----
a(float) 10.000000 -> b(float) 100.000000 -> c(int) 15 -> i(int) 105 -> n(int) 6 -> m(int) 6 -> result(int) 24 -> mx(dc

Printing all prototype
int add(int,int,float)
int scan()
void add(int,float,double)

All library functions are:
int scanInt()
float scan()
void show(any)
float max(any,any)
double sqrt(any)
int toInt(any)
float toFloat(any)
double toDouble(any)

Printing all imports:
math
stdio
```

input.txt

```
/* importing header */
import math;
import stdio;
import math;

// declaring function prototype
int add(int,int, float);
int scan();
void add(int, float, double);

/*
    multi-line comment
    /* support nested also */
*/

static void entryPoint(){

    // variable declaration
    int a;
    float b,c;

    println(a b c);
    // deleting variables
    discard a,b,c;

    /* variable initialization */
    int a = 10, b;
    int c = a <add> b, d = 10 <sub> a;
    println(a,b,c,d);

    double e = @max(2,3), f;
    println(e,f);

    discard c,d,e, /* valid comment */ f;

    // variable assignment
    a = 100;
    println("a: " a);

    b = a <rem> 11;
    println("b: " b);

    b = @max(a,101);
    println("b: " b);
    discard a,b;

    /* condition */
    float a = 10;
    float b = 100;

    justInCase(a <lt> b){
        println("Actually executed since " a "< " b);

        justInCase(10 <lt> 100){
            println("Also executed");
        }

        justInCase(10 <gt> b){
            println("Not executed since false");
        }
    }
}
```

```

        }

}

println("----- separator -----");

a = 10; int c = 15;
justInCase(5 <lt> 10 and a <lt> c){
    println("5 < 10 and " a " < " c);
}

justInCase(5 <lt> 10 and a <lt> b and 10 <gt> 10 ){
    println("Condition is false");
}

int i = 0;
till( i <lt> 100){
    i = i <add> 15;
}

/* library function */

int n = @scanInt();
println(n);

float f = @scan();
println("Value after scan is: " f);

discard n,f;

int m = 6, n = 4;
int result = m <mul> n;
@show(result);

double mx = @max(2.5,5);
@show(mx);

float rootf = @sqrt(42);
println("Square root of 42 is :" rootf "(float)");

int toFind = -42;
n = @sqrt(-42);
println("Square root of " toFind " is :" n "(int"));

n = @toInt(rootf);
println(rootf " becomes " n " after toInt" );

rootf = @toFloat(n);
println(n " becomes " rootf " after toFloat" );

rootf = @toDouble(rootf);
println(rootf " becomes " rootf " after toDouble" );

// user defined function
int n1, n2;
float f1;

int addRes = @add(n1,n1,f1);
println(addRes);

@add(m,f1,100);

int dum = @add(m,f1,100);

@test();
}

```