

1. code_generator

The screenshot shows a code editor window with the following details:

- Title Bar:** code_generator.c — code_generator
- Toolbar:** Includes standard icons for file operations.
- Left Sidebar:** Shows the "OPEN FILES" list, which includes:
 - code_generator.c
 - FOLDERS
 - code_generator
 - * bubbleSort.ast
 - * bubbleSort.lst
 - * bubbleSort.mc
 - * bubbleSort.uco
 - code_generator.h
 - main.c
 - parser.c
 - parser.h
 - perfect.ast
 - perfect.lst
 - perfect.mc
 - perfect.uco
 - prime.ast
 - prime.lst
 - prime.mc
 - prime.uco
- Code Editor:** The main area displays the code for `code_generator.c`. The code is a C program that processes declarations and simple variables. It includes various #include directives, function definitions like `processDeclaration` and `processSimpleVariable`, and logic for handling different token types and variable qualifiers.
- Right Sidebar:** Shows a vertical stack of other code files from the project, including `scanner.c`, `parser.c`, `perfect.c`, `prime.c`, and `main.c`.
- Bottom Status Bar:** Shows "Line 1, Column 1" on the left and "Tab Size: 4" on the right.

code_generator.c — code_generator

UNREGISTERED

OPEN FILES

code_generator.c

```
93 }  
94 }  
95  
96 void processArrayVariable(Node *ptr, int typeSpecifier, int typeQualifier) {  
97     Node *p = ptr -> son;  
98     int stIndex, size;  
99  
100    if(ptr -> token.number != ARRAY_VAR) {  
101        printf("error in ARRAY_VAR\n");  
102        return;  
103    }  
104  
105    if(p -> brother == NULL) {  
106        printf("array size must be typeSpecified\n");  
107        return;  
108    }  
109    else  
110        size = p -> brother -> token.value.num;  
111  
112    stIndex = insert(p -> token.value.id, typeSpecifier, typeQualifier, base, offset, size, 0);  
113  
114    offset += size;  
115  
116 }  
117  
118 void processOperator(Node *ptr) {  
119     int stIndex;  
120     switch(ptr -> token.number) {  
121         // assignment operator  
122         case ASSIGN_OP:  
123         {  
124             Node *lhs = ptr -> son;  
125             Node *rhs = ptr -> son -> brother;  
126  
127             if(lhs -> token.number == ASSIGN_OP && rhs -> token.number == ASSIGN_OP)  
128                 flag = 1;  
129  
130             // step 1 : generate instructions for left-hand side if INDEX node.  
131             if(lhs -> noderep == nonterm) {  
132                 lvalue = 1;  
133                 processOperator(lhs);  
134                 lvalue = 0;  
135             }  
136             // step 2 : generate instructions for right-hand side  
137             if(rhs -> noderep == nonterm) {  
138                 rvalue = 1;  
139                 processOperator(rhs);  
140                 rvalue = 0;  
141             }  
142  
143             //step 3 : generate a store instruction  
144             if(lhs -> noderep == terminal) {  
145                 stIndex = lookup(lhs -> token.value.id);  
146                 if(stIndex == -1) {  
147                     printf("undefined variable: %s\n", lhs -> token.value.id);  
148                     return;  
149                 }  
150                 emit2("str", symbolTable[stIndex].base, symbolTable[stIndex].offset);  
151             }  
152             else  
153                 emit0("sti");  
154             break;  
155         }  
156         // complex assignment operators  
157         case ADD_ASSIGN: case SUB_ASSIGN: case MUL_ASSIGN:  
158         case DIV_ASSIGN: case MOD_ASSIGN:  
159         {  
160             Node *lhs = ptr -> son;  
161             Node *rhs = ptr -> son -> brother;  
162  
163             int nodeNumber = ptr -> token.number;  
164  
165             ptr -> token.number = ASSIGN_OP;  
166  
167             // step 1 : code generation for left hand side  
168             if(lhs -> noderep == nonterm) {  
169                 lvalue = 1;  
170                 processOperator(lhs);  
171                 lvalue = 0;  
172             }  
173  
174             ptr -> token.number = nodeNumber;  
175  
176             // step 2 : code generation for repeating part  
177             if(lhs -> noderep == nonterm)  
178                 processOperator(lhs);  
179             else  
180                 rv_emit(lhs);  
181  
182             // step 3 : code generation for right hand side  
183             if(rhs -> noderep == nonterm)  
184                 processOperator(rhs);  
185             else  
186                 rv_emit(rhs);  
187  
188             // step 4 : code generation for final part  
189             if(flag == 1)  
190                 processOperator(rhs);  
191             else  
192                 rv_emit(rhs);  
193         }  
194     }  
195 }
```

FOLDERS

code_generator

```
* bubbleSort.ast  
* bubbleSort.lst  
* bubbleSort.mc  
* bubbleSort.uco  
code_generator.c  
code_generator.h  
* main.c  
* MiniC.tbl  
* parser.c  
* parser.h  
* perfect.ast  
* perfect.lst  
* perfect.mc  
* perfect.uco  
scanner.c
```

8 lines, 141 characters selected

Tab Size: 4

code_generator.c — code_generator

OPEN FILES

code_generator.c

FOLDERS

code_generator

- * bubbleSort.ast
- * bubbleSort.lst
- * bubbleSort.mc
- * bubbleSort.uco

code_generator.c

- </>
- * main
- </>
- main.c
- * MiniC.tbl
- * parser
- </>
- parser.c
- </>
- parser.h
- * perfect.ast
- * perfect.lst
- * perfect.mc
- * perfect.uco
- * prime.ast
- * prime.lst
- * prime.mc
- * prime.uco

code_generator.c

- </>
- * scanner.c
- FOLDERS
- code_generator

 - * bubbleSort.ast
 - * bubbleSort.lst
 - * bubbleSort.mc
 - * bubbleSort.uco

code_generator.c

 - </>
 - * main
 - </>
 - main.c
 - * MiniC.tbl
 - * parser
 - </>
 - parser.c
 - </>
 - parser.h
 - * perfect.ast
 - * perfect.lst
 - * perfect.mc
 - * perfect.uco
 - * prime.ast
 - * prime.lst
 - * prime.mc
 - * prime.uco

scanner.c

code_generator.c

```
// step 4 : emit the corresponding operation code
switch(ptr -> token.number) {
    case ADD_ASSIGN: emit0("add"); break;
    case SUB_ASSIGN: emit0("sub"); break;
    case MUL_ASSIGN: emit0("mult"); break;
    case DIV_ASSIGN: emit0("div"); break;
    case MOD_ASSIGN: emit0("mod"); break;
}

// step 5 : code generation for store code
if(lhs -> noderep == terminal) {
    stIndex = lookup(lhs -> token.value.id);
    if(stIndex == -1) return;

    emit2("str", symbolTable[stIndex].base, symbolTable[stIndex].offset);
}
else
    emit0("sti");
break;

// binary(arithmetic / relational / logical) operations
case ADD: case SUB: case MUL: case DIV: case MOD:
case EQ: case NE: case GT: case LT: case GE:
case LE: case LOGICAL_AND: case LOGICAL_OR:
{
    Node *lhs = ptr -> son;
    Node *rhs = ptr -> son -> brother;

    // step 1 : visit left operand
    if(lhs -> noderep == nonterm) processOperator(lhs);
    else rv_emit(lhs);
    // step 2 : visit right operand
    if(rhs -> noderep == nonterm) processOperator(rhs);
    else rv_emit(rhs);

    // step 3 : visit root
    switch(ptr -> token.number) {
        case ADD: emit0("add"); break;
        case SUB: emit0("sub"); break;
        case MUL: emit0("mult"); break;
        case DIV: emit0("div"); break;
        case MOD: emit0("mod"); break;
        case EQ: emit0("eq"); break;
        case NE: emit0("ne"); break;
        case GT: emit0("gt"); break;
        case GE: emit0("ge"); break;
        case LE: emit0("le"); break;
        case LOGICAL_AND: emit0("and"); break;
        case LOGICAL_OR: emit0("or"); break;
    }
    if(flag == 1){
        emit0("dup");
        flag = 0;
    }
    break;
}

case UNARY_MINUS: case LOGICAL_NOT:
{
    Node *p = ptr -> son;

    if(p -> noderep == nonterm) processOperator(p);
    else rv_emit(p);

    switch(ptr -> token.number) {
        case UNARY_MINUS: emit0("neg"); break;
        case LOGICAL_NOT: emit0("not"); break;
    }
    break;
}

case INDEX:
{
    Node *indexExp = ptr -> son -> brother;

    if(indexExp -> noderep == nonterm) processOperator(indexExp);
    else rv_emit(indexExp);

    stIndex = lookup(ptr -> son -> token.value.id);

    if(stIndex == -1) {
        printf("undefined variable: %s\n", ptr -> son -> token.value.id);
        return;
    }

    emit2("lda", symbolTable[stIndex].base, symbolTable[stIndex].offset);
    emit0("add");

    if(!lvalue) emit0("ldi");
    break;
}
```

8 lines, 141 characters selected

UNREGISTERED

Tab Size: 4

```
// increment / decrement operations
case PRE_INC: case PRE_DEC: case POST_INC: case POST_DEC:
{
    Node *p = ptr -> son;
    Node *q;
    int stIndex; int amount = 1;

    if(p -> noderep == nonterm) processOperator(p);
    else rv_emit(p);

    q = p;

    while(q -> noderep != terminal)
        q = q -> son;

    if(!q || (q -> token.number != tident)) {
        printf("increment/decrement operators can not be applied in expression\n");
        return;
    }

    stIndex = lookup(q -> token.value.id);
    if(stIndex == -1) return;

    switch(ptr -> token.number) {
        case PRE_INC: emit0("inc"); break;
        case PRE_DEC: emit0("dec"); break;
        case POST_INC: emit0("inc"); break;
        case POST_DEC: emit0("dec"); break;
    }

    if(p -> noderep == terminal) {
        stIndex = lookup(p -> token.value.id);
        if(stIndex == -1) return;
        emit2("str", symbolTable[stIndex].base, symbolTable[stIndex].offset);
    } else if(p -> token.number == INDEX) {
        lvalue = 1;
        processOperator(p);
        lvalue = 0;
        emit0("sti");
    }
    else printf("error increase/decrease operators\n");
    break;
}

case CALL:
{
    Node *p = ptr -> son;
    char *functionName;
    int stIndex;
    int noArguments;

    if(checkPredefined(p)) break;

    functionName = p -> token.value.id;
    stIndex = lookup(functionName);

    if(stIndex == -1) break;

    noArguments = symbolTable[stIndex].width;

    emit0("ldp");
    p = p -> brother;

    while(p) {
        if(p -> noderep == nonterm) processOperator(p);
        else rv_emit(p);

        noArguments--;
        p = p -> brother;
    }

    if(noArguments > 0)
        printf("%s: too few actual arguments\n", functionName);
    if(noArguments < 0)
        printf("%s: too many actual arguments\n", functionName);
    emitJump("call", ptr -> son -> token.value.id);
    break;
}
```

code_generator.c — code_generator

UNREGISTERED

OPEN FILES

code_generator.c

FOLDERS

code_generator

- * bubbleSort.ast
- * bubbleSort.lst
- * bubbleSort.mc
- * bubbleSort.uco

code_generator.c

- 371 void processStatement(Node *ptr) {
- 372 Node *p, *q;
- 373 char *startLabel, *endLabel;
- 374
- 375 switch(ptr -> token.number) {
- 376 case COMPOUND_ST:
- 377 p = ptr -> son -> brother;
- 378 p = p -> son;
- 379 while (p) {
- 380 processStatement(p);
- 381 p = p -> brother;
- 382 }
- 383 break;
- 384
- 385 case EXP_ST:
- 386 if(ptr -> son != NULL) processOperator(ptr -> son);
- 387 break;
- 388
- 389 case RETURN_ST:
- 390 if (ptr -> son != NULL) {
- 391 p = ptr -> son;
- 392 if(p -> noderep == nonterm)
- 393 processOperator(p);
- 394 else
- 395 rv_emit(p);
- 396 emit0("retv");
- 397 else
- 398 emit0("ret");
- 399 break;
- 400
- 401 case IF_ST:
- 402 {
- 403 char label[LABEL_SIZE];
- 404
- 405 genLabel(label);
- 406 processCondition(ptr -> son);
- 407 emitJump("fjp", label);
- 408 processStatement(ptr -> son -> brother);
- 409 emitLabel(label);
- 410 }
- 411 break;
- 412
- 413 case IF_ELSE_ST:
- 414 {
- 415 char label1[LABEL_SIZE], label2[LABEL_SIZE];
- 416
- 417 genLabel(label1); genLabel(label2);
- 418 processCondition(ptr -> son);
- 419
- 420 emitJump("fjp", label1);
- 421 processStatement(ptr -> son -> brother);
- 422 emitJump("ujp", label2);
- 423
- 424 emitLabel(label1);
- 425 processStatement(ptr -> son -> brother -> brother);
- 426 emitLabel(label2);
- 427 }
- 428 break;
- 429
- 430 case WHILE_ST:
- 431 {
- 432 char label1[LABEL_SIZE], label2[LABEL_SIZE];
- 433
- 434 genLabel(label1); genLabel(label2);
- 435 emitLabel(label1);
- 436 processCondition(ptr -> son);
- 437
- 438 emitJump("fjp", label2);
- 439 processStatement(ptr -> son -> brother);
- 440 emitJump("ujp", label1);
- 441 emitLabel(label2);
- 442 }
- 443 break;
- 444
- 445 }
- 446 }
- 447 void processCondition(Node *ptr) {
- 448 if(ptr -> noderep == nonterm) processOperator(ptr);
- 449 else rv_emit(ptr);
- 450 }
- 451
- 452 void processFuncHeader(Node *ptr) {
- 453 int noArguments, returnType;
- 454 int stIndex;
- 455 Node *p = ptr;
- 456
- 457 printf("processFunctionHeader\n\n");
- 458
- 459 if(ptr -> token.number != FUNC_HEAD){
- 460 printf("error in processFunctionHeader!\n");
- 461 printTree(ptr,0);
- 462 exit(1);
- 463 }
- 464 }

```
  * bubbleSort.ast      465 // step 1 : process the function return type
  * bubbleSort.mc      466 p = ptr -> son -> son;
  * bubbleSort.uco     467 while(p){
  *                   468     if(p -> token.number == INT_NODE)
  *                   469         returnType = INT_TYPE;
  *                   else if(p -> token.number == VOID_NODE)
  *                   470                     returnType = VOID_TYPE;
  *                   else printf("invalid function return type!\n");
  * 
  *                   p = p -> brother;
  * }
  * 
  * // step 2 : count the number of formal parameters
  * main.c             477 p = ptr -> son -> brother -> brother; // FORMAL PARA
  * MiniC.tbl          478 p = p -> son; // PARAM_DCL
  * 
  * parser             480 noArguments = 0;
  * 
  * parser.c           482 while(p){
  *     noArguments++;
  *     p = p -> brother;
  * }
  * 
  * // step 3 : insert the function name
  * perfect.ast        487 stIndex = insert(ptr -> son -> brother -> token.value.id, returnType, FUNC_TYPE, 1/*base*/, 0/*offset*/, noArgument);
  * 
  * perfect.lst        489 }
  * 
  * perfect.mc          491
  * 
  * perfect.uco         493 void codeGen(Node *ptr) {
  * Node *p;
  * int globalSize;
  * 
  * initSymbolTable();
  * // step 1 : process the declaration part
  * for(p = ptr -> son; p; p = p -> brother){
  *     if(p -> token.number == DCL){
  *         printf("\n\n\n");
  *         processDeclaration(p -> son);
  *     }
  *     else if(p -> token.number == FUNC_DEF) processFuncHeader(p -> son);
  * }
  * 
  * // dumpSymbolTable();
  * globalSize = offset - 1;
  * // printf("size of global variables = %d\n", globalSize);
  * 
  * genSym(base);
  * 
  * // step 2 : process the function part
  * for(p = ptr -> son; p; p = p -> brother)
  *     if(p -> token.number == FUNC_DEF) processFunction(p);
  * //if(!mainExist) warningmsg("main does not exist");
  * 
  * // step 3 : generate codes for starting routine
  * emit1("bgn", globalSize);
  * emit0("ldp");
  * emitJump("call", "main");
  * emit0("end");
  * }
  * 
  * int checkPredefined(Node *ptr) {
  * Node *p = ptr;
  * char *functionName;
  * int noArguments;
  * int stIndex;
  * functionName = p -> token.value.id;
  * 
  * if(strcmp(functionName, "read") == 0) {
  *     noArguments = 1;
  *     emit0("ldp");
  *     p = p -> brother;
  *     while(p) {
  *         if(p->nodeType == nonterm)
  *             processOperator(p);
  *         else {
  *             stIndex = lookup(p -> token.value.id);
  * 
  *             if(stIndex == -1)
  *                 return 0;
  * 
  *             emit2("lda", symbolTable[stIndex].base, symbolTable[stIndex].offset);
  *         }
  *         noArguments--;
  *         p = p -> brother;
  *     }
  * 
  *     if(noArguments > 0) printf("%s: too few actual arguments\n", functionName);
  *     else if(noArguments < 0) printf("%s: too many actual arguments\n", functionName);
  * 
  *     emitJump("call", functionName);
  *     return 1;
  * }
  * 
  * return 0;
  * }
  * 
  * Line 437, Column 42
  * code_generator.c    508
  * FOLDERS
  * code_generator
  * 
  * code_generator.c    512
  * 
  * * bubbleSort.ast      514 // step 2 : process the function part
  * * bubbleSort.lst      515 for(p = ptr -> son; p; p = p -> brother)
  * *                   516     if(p -> token.number == FUNC_DEF) processFunction(p);
  * * //if(!mainExist) warningmsg("main does not exist");
  * * 
  * * // step 3 : generate codes for starting routine
  * * emit1("bgn", globalSize);
  * * emit0("ldp");
  * * emitJump("call", "main");
  * * emit0("end");
  * * }
  * * 
  * * int checkPredefined(Node *ptr) {
  * * Node *p = ptr;
  * * char *functionName;
  * * int noArguments;
  * * int stIndex;
  * * functionName = p -> token.value.id;
  * * 
  * * if(strcmp(functionName, "read") == 0) {
  * *     noArguments = 1;
  * *     emit0("ldp");
  * *     p = p -> brother;
  * *     while(p) {
  * *         if(p->nodeType == nonterm)
  * *             processOperator(p);
  * *         else {
  * *             stIndex = lookup(p -> token.value.id);
  * * 
  * *             if(stIndex == -1)
  * *                 return 0;
  * * 
  * *             emit2("lda", symbolTable[stIndex].base, symbolTable[stIndex].offset);
  * *         }
  * *         noArguments--;
  * *         p = p -> brother;
  * *     }
  * * 
  * *     if(noArguments > 0) printf("%s: too few actual arguments\n", functionName);
  * *     else if(noArguments < 0) printf("%s: too many actual arguments\n", functionName);
  * * 
  * *     emitJump("call", functionName);
  * *     return 1;
  * * }
  * * 
  * * return 0;
  * * }
  * * 
  * * Line 437, Column 42
  * * code_generator.c    524
  * * 
  * * code_generator.c    528
  * * 
  * * * main.c             530
  * * * MiniC.tbl          531
  * * * 
  * * * parser             532
  * * * parser.c           534
  * * * parser.h           536
  * * * 
  * * * perfect.ast        538
  * * * perfect.lst        540
  * * * perfect.mc          542
  * * * perfect.uco         544
  * * * 
  * * * prime.ast          546
  * * * prime.lst          548
  * * * prime.mc          550
  * * * prime.uco          552
  * * * 
  * * * scanner.c          554
  * * * 
  * * * Line 437, Column 42
  * * * code_generator.c    556
  * * * 
  * * * code_generator.c    560
  * * * 
  * * * * main.c             562
  * * * * MiniC.tbl          563
  * * * * 
  * * * * parser             565
  * * * * parser.c           567
  * * * * parser.h           569
  * * * * 
  * * * * perfect.ast        571
  * * * * perfect.lst        573
  * * * * perfect.mc          575
  * * * * perfect.uco         577
  * * * * 
  * * * * prime.ast          579
  * * * * prime.lst          581
  * * * * prime.mc          583
  * * * * prime.uco          585
  * * * * 
  * * * * scanner.c          587
  * * * * 
  * * * * Line 437, Column 42
  * * * * code_generator.c    589
  * * * * 
  * * * * code_generator.c    593
  * * * * 
  * * * * * main.c             595
  * * * * * MiniC.tbl          596
  * * * * * 
  * * * * * parser             598
  * * * * * parser.c           600
  * * * * * parser.h           602
  * * * * * 
  * * * * * perfect.ast        604
  * * * * * perfect.lst        606
  * * * * * perfect.mc          608
  * * * * * perfect.uco         610
  * * * * * 
  * * * * * prime.ast          612
  * * * * * prime.lst          614
  * * * * * prime.mc          616
  * * * * * prime.uco          618
  * * * * * 
  * * * * * scanner.c          620
  * * * * * 
  * * * * * Line 437, Column 42
  * * * * * code_generator.c    622
  * * * * * 
  * * * * * code_generator.c    626
  * * * * * 
  * * * * * * main.c             628
  * * * * * * MiniC.tbl          629
  * * * * * * 
  * * * * * * parser             631
  * * * * * * parser.c           633
  * * * * * * parser.h           635
  * * * * * * 
  * * * * * * perfect.ast        637
  * * * * * * perfect.lst        639
  * * * * * * perfect.mc          641
  * * * * * * perfect.uco         643
  * * * * * * 
  * * * * * * prime.ast          645
  * * * * * * prime.lst          647
  * * * * * * prime.mc          649
  * * * * * * prime.uco          651
  * * * * * * 
  * * * * * * scanner.c          653
  * * * * * * 
  * * * * * * Line 437, Column 42
  * * * * * * code_generator.c    655
  * * * * * * 
  * * * * * * code_generator.c    659
  * * * * * * 
  * * * * * * * main.c             661
  * * * * * * * MiniC.tbl          662
  * * * * * * * 
  * * * * * * * parser             664
  * * * * * * * parser.c           666
  * * * * * * * parser.h           668
  * * * * * * * 
  * * * * * * * perfect.ast        670
  * * * * * * * perfect.lst        672
  * * * * * * * perfect.mc          674
  * * * * * * * perfect.uco         676
  * * * * * * * 
  * * * * * * * prime.ast          678
  * * * * * * * prime.lst          680
  * * * * * * * prime.mc          682
  * * * * * * * prime.uco          684
  * * * * * * * 
  * * * * * * * scanner.c          686
  * * * * * * * 
  * * * * * * * Line 437, Column 42
  * * * * * * * code_generator.c    688
  * * * * * * * 
  * * * * * * * code_generator.c    692
  * * * * * * * 
  * * * * * * * * main.c             694
  * * * * * * * * MiniC.tbl          695
  * * * * * * * * 
  * * * * * * * * parser             697
  * * * * * * * * parser.c           699
  * * * * * * * * parser.h           701
  * * * * * * * * 
  * * * * * * * * perfect.ast        703
  * * * * * * * * perfect.lst        705
  * * * * * * * * perfect.mc          707
  * * * * * * * * perfect.uco         709
  * * * * * * * * 
  * * * * * * * * prime.ast          711
  * * * * * * * * prime.lst          713
  * * * * * * * * prime.mc          715
  * * * * * * * * prime.uco          717
  * * * * * * * * 
  * * * * * * * * scanner.c          719
  * * * * * * * * 
  * * * * * * * * Line 437, Column 42
  * * * * * * * * code_generator.c    721
  * * * * * * * * 
  * * * * * * * * code_generator.c    725
  * * * * * * * * 
  * * * * * * * * * main.c             727
  * * * * * * * * * MiniC.tbl          728
  * * * * * * * * * 
  * * * * * * * * * parser             730
  * * * * * * * * * parser.c           732
  * * * * * * * * * parser.h           734
  * * * * * * * * * 
  * * * * * * * * * perfect.ast        736
  * * * * * * * * * perfect.lst        738
  * * * * * * * * * perfect.mc          740
  * * * * * * * * * perfect.uco         742
  * * * * * * * * * 
  * * * * * * * * * prime.ast          744
  * * * * * * * * * prime.lst          746
  * * * * * * * * * prime.mc          748
  * * * * * * * * * prime.uco          750
  * * * * * * * * * 
  * * * * * * * * * scanner.c          752
  * * * * * * * * * 
  * * * * * * * * * Line 437, Column 42
  * * * * * * * * * code_generator.c    754
  * * * * * * * * * 
  * * * * * * * * * code_generator.c    758
  * * * * * * * * * 
  * * * * * * * * * * main.c             760
  * * * * * * * * * * MiniC.tbl          761
  * * * * * * * * * * 
  * * * * * * * * * * parser             763
  * * * * * * * * * * parser.c           765
  * * * * * * * * * * parser.h           767
  * * * * * * * * * * 
  * * * * * * * * * * perfect.ast        769
  * * * * * * * * * * perfect.lst        771
  * * * * * * * * * * perfect.mc          773
  * * * * * * * * * * perfect.uco         775
  * * * * * * * * * * 
  * * * * * * * * * * prime.ast          777
  * * * * * * * * * * prime.lst          779
  * * * * * * * * * * prime.mc          781
  * * * * * * * * * * prime.uco          783
  * * * * * * * * * * 
  * * * * * * * * * * scanner.c          785
  * * * * * * * * * * 
  * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * code_generator.c    787
  * * * * * * * * * * 
  * * * * * * * * * * code_generator.c    791
  * * * * * * * * * * 
  * * * * * * * * * * * main.c             793
  * * * * * * * * * * * MiniC.tbl          794
  * * * * * * * * * * * 
  * * * * * * * * * * * parser             796
  * * * * * * * * * * * parser.c           798
  * * * * * * * * * * * parser.h           800
  * * * * * * * * * * * 
  * * * * * * * * * * * perfect.ast        802
  * * * * * * * * * * * perfect.lst        804
  * * * * * * * * * * * perfect.mc          806
  * * * * * * * * * * * perfect.uco         808
  * * * * * * * * * * * 
  * * * * * * * * * * * prime.ast          810
  * * * * * * * * * * * prime.lst          812
  * * * * * * * * * * * prime.mc          814
  * * * * * * * * * * * prime.uco          816
  * * * * * * * * * * * 
  * * * * * * * * * * * scanner.c          818
  * * * * * * * * * * * 
  * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * code_generator.c    820
  * * * * * * * * * * * 
  * * * * * * * * * * * code_generator.c    824
  * * * * * * * * * * * 
  * * * * * * * * * * * * main.c             826
  * * * * * * * * * * * * MiniC.tbl          827
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             829
  * * * * * * * * * * * * parser.c           831
  * * * * * * * * * * * * parser.h           833
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        835
  * * * * * * * * * * * * perfect.lst        837
  * * * * * * * * * * * * perfect.mc          839
  * * * * * * * * * * * * perfect.uco         841
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          843
  * * * * * * * * * * * * prime.lst          845
  * * * * * * * * * * * * prime.mc          847
  * * * * * * * * * * * * prime.uco          849
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          851
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    853
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    857
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             859
  * * * * * * * * * * * * MiniC.tbl          860
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             862
  * * * * * * * * * * * * parser.c           864
  * * * * * * * * * * * * parser.h           866
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        868
  * * * * * * * * * * * * perfect.lst        870
  * * * * * * * * * * * * perfect.mc          872
  * * * * * * * * * * * * perfect.uco         874
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          876
  * * * * * * * * * * * * prime.lst          878
  * * * * * * * * * * * * prime.mc          880
  * * * * * * * * * * * * prime.uco          882
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          884
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    886
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    890
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             892
  * * * * * * * * * * * * MiniC.tbl          893
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             895
  * * * * * * * * * * * * parser.c           897
  * * * * * * * * * * * * parser.h           899
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        901
  * * * * * * * * * * * * perfect.lst        903
  * * * * * * * * * * * * perfect.mc          905
  * * * * * * * * * * * * perfect.uco         907
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          909
  * * * * * * * * * * * * prime.lst          911
  * * * * * * * * * * * * prime.mc          913
  * * * * * * * * * * * * prime.uco          915
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          917
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    919
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    923
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             925
  * * * * * * * * * * * * MiniC.tbl          926
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             928
  * * * * * * * * * * * * parser.c           930
  * * * * * * * * * * * * parser.h           932
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        934
  * * * * * * * * * * * * perfect.lst        936
  * * * * * * * * * * * * perfect.mc          938
  * * * * * * * * * * * * perfect.uco         940
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          942
  * * * * * * * * * * * * prime.lst          944
  * * * * * * * * * * * * prime.mc          946
  * * * * * * * * * * * * prime.uco          948
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          950
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    952
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    956
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             958
  * * * * * * * * * * * * MiniC.tbl          959
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             961
  * * * * * * * * * * * * parser.c           963
  * * * * * * * * * * * * parser.h           965
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        967
  * * * * * * * * * * * * perfect.lst        969
  * * * * * * * * * * * * perfect.mc          971
  * * * * * * * * * * * * perfect.uco         973
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          975
  * * * * * * * * * * * * prime.lst          977
  * * * * * * * * * * * * prime.mc          979
  * * * * * * * * * * * * prime.uco          981
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          983
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    985
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    989
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             991
  * * * * * * * * * * * * MiniC.tbl          992
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             994
  * * * * * * * * * * * * parser.c           996
  * * * * * * * * * * * * parser.h           998
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1000
  * * * * * * * * * * * * perfect.lst        1002
  * * * * * * * * * * * * perfect.mc          1004
  * * * * * * * * * * * * perfect.uco         1006
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1008
  * * * * * * * * * * * * prime.lst          1010
  * * * * * * * * * * * * prime.mc          1012
  * * * * * * * * * * * * prime.uco          1014
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1016
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1018
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1022
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1024
  * * * * * * * * * * * * MiniC.tbl          1025
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1027
  * * * * * * * * * * * * parser.c           1029
  * * * * * * * * * * * * parser.h           1031
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1033
  * * * * * * * * * * * * perfect.lst        1035
  * * * * * * * * * * * * perfect.mc          1037
  * * * * * * * * * * * * perfect.uco         1039
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1041
  * * * * * * * * * * * * prime.lst          1043
  * * * * * * * * * * * * prime.mc          1045
  * * * * * * * * * * * * prime.uco          1047
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1049
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1051
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1055
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1057
  * * * * * * * * * * * * MiniC.tbl          1058
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1060
  * * * * * * * * * * * * parser.c           1062
  * * * * * * * * * * * * parser.h           1064
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1066
  * * * * * * * * * * * * perfect.lst        1068
  * * * * * * * * * * * * perfect.mc          1070
  * * * * * * * * * * * * perfect.uco         1072
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1074
  * * * * * * * * * * * * prime.lst          1076
  * * * * * * * * * * * * prime.mc          1078
  * * * * * * * * * * * * prime.uco          1080
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1082
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1084
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1088
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1090
  * * * * * * * * * * * * MiniC.tbl          1091
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1093
  * * * * * * * * * * * * parser.c           1095
  * * * * * * * * * * * * parser.h           1097
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1099
  * * * * * * * * * * * * perfect.lst        1101
  * * * * * * * * * * * * perfect.mc          1103
  * * * * * * * * * * * * perfect.uco         1105
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1107
  * * * * * * * * * * * * prime.lst          1109
  * * * * * * * * * * * * prime.mc          1111
  * * * * * * * * * * * * prime.uco          1113
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1115
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1117
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1121
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1123
  * * * * * * * * * * * * MiniC.tbl          1124
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1126
  * * * * * * * * * * * * parser.c           1128
  * * * * * * * * * * * * parser.h           1130
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1132
  * * * * * * * * * * * * perfect.lst        1134
  * * * * * * * * * * * * perfect.mc          1136
  * * * * * * * * * * * * perfect.uco         1138
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1140
  * * * * * * * * * * * * prime.lst          1142
  * * * * * * * * * * * * prime.mc          1144
  * * * * * * * * * * * * prime.uco          1146
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1148
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1150
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1154
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1156
  * * * * * * * * * * * * MiniC.tbl          1157
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1159
  * * * * * * * * * * * * parser.c           1161
  * * * * * * * * * * * * parser.h           1163
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1165
  * * * * * * * * * * * * perfect.lst        1167
  * * * * * * * * * * * * perfect.mc          1169
  * * * * * * * * * * * * perfect.uco         1171
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1173
  * * * * * * * * * * * * prime.lst          1175
  * * * * * * * * * * * * prime.mc          1177
  * * * * * * * * * * * * prime.uco          1179
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1181
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1183
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1187
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1189
  * * * * * * * * * * * * MiniC.tbl          1190
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1192
  * * * * * * * * * * * * parser.c           1194
  * * * * * * * * * * * * parser.h           1196
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1198
  * * * * * * * * * * * * perfect.lst        1200
  * * * * * * * * * * * * perfect.mc          1202
  * * * * * * * * * * * * perfect.uco         1204
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1206
  * * * * * * * * * * * * prime.lst          1208
  * * * * * * * * * * * * prime.mc          1210
  * * * * * * * * * * * * prime.uco          1212
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1214
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1216
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1220
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1222
  * * * * * * * * * * * * MiniC.tbl          1223
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1225
  * * * * * * * * * * * * parser.c           1227
  * * * * * * * * * * * * parser.h           1229
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1231
  * * * * * * * * * * * * perfect.lst        1233
  * * * * * * * * * * * * perfect.mc          1235
  * * * * * * * * * * * * perfect.uco         1237
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1239
  * * * * * * * * * * * * prime.lst          1241
  * * * * * * * * * * * * prime.mc          1243
  * * * * * * * * * * * * prime.uco          1245
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1247
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1249
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1253
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1255
  * * * * * * * * * * * * MiniC.tbl          1256
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1258
  * * * * * * * * * * * * parser.c           1260
  * * * * * * * * * * * * parser.h           1262
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1264
  * * * * * * * * * * * * perfect.lst        1266
  * * * * * * * * * * * * perfect.mc          1268
  * * * * * * * * * * * * perfect.uco         1270
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1272
  * * * * * * * * * * * * prime.lst          1274
  * * * * * * * * * * * * prime.mc          1276
  * * * * * * * * * * * * prime.uco          1278
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1280
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1282
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1286
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1288
  * * * * * * * * * * * * MiniC.tbl          1289
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1291
  * * * * * * * * * * * * parser.c           1293
  * * * * * * * * * * * * parser.h           1295
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1297
  * * * * * * * * * * * * perfect.lst        1299
  * * * * * * * * * * * * perfect.mc          1301
  * * * * * * * * * * * * perfect.uco         1303
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1305
  * * * * * * * * * * * * prime.lst          1307
  * * * * * * * * * * * * prime.mc          1309
  * * * * * * * * * * * * prime.uco          1311
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1313
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1315
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1319
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1321
  * * * * * * * * * * * * MiniC.tbl          1322
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1324
  * * * * * * * * * * * * parser.c           1326
  * * * * * * * * * * * * parser.h           1328
  * * * * * * * * * * * * 
  * * * * * * * * * * * * perfect.ast        1330
  * * * * * * * * * * * * perfect.lst        1332
  * * * * * * * * * * * * perfect.mc          1334
  * * * * * * * * * * * * perfect.uco         1336
  * * * * * * * * * * * * 
  * * * * * * * * * * * * prime.ast          1338
  * * * * * * * * * * * * prime.lst          1340
  * * * * * * * * * * * * prime.mc          1342
  * * * * * * * * * * * * prime.uco          1344
  * * * * * * * * * * * * 
  * * * * * * * * * * * * scanner.c          1346
  * * * * * * * * * * * * 
  * * * * * * * * * * * * Line 437, Column 42
  * * * * * * * * * * * * code_generator.c    1348
  * * * * * * * * * * * * 
  * * * * * * * * * * * * code_generator.c    1352
  * * * * * * * * * * * * 
  * * * * * * * * * * * * * main.c             1354
  * * * * * * * * * * * * MiniC.tbl          1355
  * * * * * * * * * * * * 
  * * * * * * * * * * * * parser             1357
  * * * * * *
```

The screenshot shows a code editor interface with the following details:

- Project:** code_generator
- File:** code_generator.c
- Status:** UNREGISTERED
- Open Files:** code_generator.c
- Folders:** code_generator (containing bubbleSort.ast, bubbleSort.lst, bubbleSort.mc, bubbleSort.uco), code_generator.c (containing main.c, MiniC.tbl, parser.c, parser.h, perfect.ast, perfect.lst, perfect.mc, perfect.uco, prime.ast, prime.lst, prime.mc, prime.uco, scanner.c)
- Code Content:** The code is a C program named code_generator.c. It includes several functions and loops. Key parts include:
 - A function `processFunction` that processes a node pointer.
 - Comments indicating steps: "step 1 : process formal parameters", "step 2 : process the declaration part in function body", "step 3 : emit the function start code", "step 4 : process the statement part in function body", "step 5 : check if return check if return type and return value", and "step 6 : generate the ending codes".
 - Specific tokens like PARAM_DCL, DCL, COMPOUND_ST, and RET are checked and processed.
 - Code generation for emitting functions starts at offset -1.
- Editor UI:** Shows tabs for Line 437, Column 42, Tab Size: 4, and a character icon.

2. main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "parser.c"
4 #include "scanner.c"
5 #include "code_generator.c"
6
7 FILE *sourceFile;
8 FILE *astFile;
9 FILE *ucodeFile;
10
11 void main (int argc, char *argv[]){
12     char fileName[30];
13     Node *root;
14     printf(" *** start of Mini C Compiler\n");
15     if(argc != 2){
16         icg_error(1);
17         exit(1);
18     }
19
20     strcpy(fileName, argv[1]);
21
22     printf(" * source file name : %s\n", fileName);
23
24     if((sourceFile = fopen(fileName , "r")) == NULL){
25         icg_error(2);
26         exit(1);
27     }
28     astFile = fopen(strcat(strtok(fileName, ".") , ".ast") , "w");
29     ucodeFile = fopen(strcat(strtok(fileName, ".") , ".uco") , "w");
30
31     printf(" === start of Parser\n");
32     root = parser();
33     printf(" === start of ICG\n");
34     codeGen(root);
35     printf(" *** end of Mini C Compiler\n");
36 } //end of main
37
```

3. bubble.mc

```
1 void main ()  
2 {  
3     int list[100];  
4     int element;  
5     int total, i, top;  
6     int temp;  
7     i = 1;  
8     read(element);  
9  
10    while(element != 0){  
11        list[i] = element;  
12        ++i;  
13        read(element);  
14    }  
15  
16    top = total = i - 1;  
17  
18    while (top > 1) {  
19        i = 1;  
20        while (i < top) {  
21            if(list[i] > list[i + 1]) {  
22                temp = list[i];  
23                list[i] = list[i + 1];  
24                list[i + 1] = temp;  
25            }  
26            ++i;  
27        }  
28        top--;  
29    }  
30  
31    i = 1;  
32  
33    while(i <= total) {  
34        write(list[i]);  
35        ++i;  
36    }  
37}  
38}
```

4. perfect.mc

```
1 const int max = 500;
2
3 void main()
4 {
5     int i, j, k;
6     int rem, sum;
7
8     i = 2;
9     while(i <= max){
10         sum = 0;
11         k = i / 2;
12         j = 1;
13         while(j <= k){
14             rem = i % j;
15             if(rem == 0) sum += j;
16             ++j;
17         }
18         if (i == sum) write(i);
19         ++i;
20     }
21 }
```

5. bubbleSort.uco

```

1 |     fun 105 1 2      44    $$4      nop      88    $$5      nop
2 |     sym 1 1 100      45      lod 1 103   89      lod 1 104   nop
3 |     sym 1 101 1      46      lod 1 104   90      dec      91      str 1 104   lod 1 104
4 |     sym 1 102 1      47      lt       91      str 1 104   dec
5 |     sym 1 103 1      48      fjp $$5    92      ujp $$2    str 1 104
6 |     sym 1 104 1      49      lod 1 103   93      $$3      ujp $$2
7 |     sym 1 105 1      50      lda 1 1    94      nop      95      nop
8 |     ldc 1             51      add      95      ldc 1    96      $$7      ldc 1
9 |     str 1 103        52      ldi      96      str 1 103   str 1 103
10 |    ldp               53      lod 1 103  97      lod 1 103   str 1 103
11 |    lda 1 101        54      ldc 1    98      lod 1 102   lod 1 102
12 |    call read         55      add      99      le      100      fjp $$8
13 |    $$0               56      lda 1 1    100      fjp $$8
14 |    nop               57      add      101      ldp      101      ldp
15 |    lod 1 101        58      ldi      102      lod 1 103   lod 1 103
16 |    ldc 0             59      gt       103      lda 1 1    102      ldp
17 |    ne                60      fjp $$6    104      add      103      lod 1 103
18 |    fjp $$1            61      lod 1 103  105      ldi      104      lda 1 1
19 |    lod 1 103        62      lda 1 1    106      call write  add
20 |    lda 1 1             63      add      107      lod 1 103   ldi
21 |    add               64      ldi      108      inc      108      call write
22 |    lod 1 101        65      str 1 105  109      str 1 103   lod 1 103
23 |    sti               66      lod 1 103  110      ujp $$7   ujp $$7
24 |    lod 1 103        67      lda 1 1    111      $$8      nop
25 |    inc                68      add      112      ret      111      $$8
26 |    str 1 103        69      lod 1 103  113      end      112      end
27 |    ldp               70      ldc 1    114      bgn 0    113      end
28 |    lda 1 101        71      add      115      ldp      114      bgn 0
29 |    call read         72      lda 1 1    116      call main  ldp
30 |    ujp $$0            73      add      117      end      116      call main
31 |    $$1               74      add      117      end
32 |    nop               75      ldi      118
33 |    lod 1 103        76      sti      119
34 |    ldc 1             77      lod 1 103  120
35 |    sub               78      ldc 1    121
36 |    dup                79      add      122
37 |    str 1 102        80      lda 1 1    123
38 |    str 1 104        81      add      124
39 |    $$2               82      lod 1 105  125
40 |    lod 1 104        83      sti      126
41 |    ldc 1             84      nop      127
42 |    gt                85      lod 1 103  128
43 |    fjp $$3            86      inc      129
44 |    ldc 1             87      str 1 103  130
45 |    str 1 103        88      ujp $$4    131

```

6. perfect.uco

```
1      fun 5 1 2    34      fjp $$4
2      sym 1 1 1   35      lod 1 5
3      sym 1 2 1   36      lod 1 2
4      sym 1 3 1   37      add
5      sym 1 4 1   38      str 1 5
6      sym 1 5 1   39      $$4
7      ldc 2       40      nop
8      str 1 1     41      lod 1 2
9      nop         42      inc
10     lod 1 1     43      str 1 2
11     ldc 500     44      ujp $$2
12     le          45      nop
13     fjp $$1     46      lod 1 1
14     ldc 0       47      lod 1 5
15     str 1 5     48      eq
16     lod 1 1     49      fjp $$5
17     ldc 2       50      ldp
18     div         51      lod 1 1
19     str 1 3     52      call write
20     ldc 1       53      nop
21     str 1 2     54      lod 1 1
22     $$2         55      inc
23     nop         56      str 1 1
24     lod 1 2     57      ujp $$0
25     lod 1 3     58      nop
26     le          59      ret
27     fjp $$3     60      end
28     lod 1 1     61      bgn 0
29     lod 1 2     62      ldp
30     mod         63      call main
31     str 1 4     64      end
32     lod 1 4
33     ldc 0
34     eq
35     fjp $$4
36     lod 1 5
37     lod 1 2
38     add
39     str 1 5
40     $$4         nop
41     lod 1 2
42     inc
43     str 1 2
44     ujp $$2
45     $$3         nop
46     lod 1 1
47     lod 1 5
48     eq
49     fjp $$5
50     ldp
51     lod 1 1
52     call write
```

7. 실행결과

```
== start of ICG
processFunctionHeader

processFunction

Nonterminal : FUNC_DEF
    Nonterminal : FUNC_HEAD
        Nonterminal : DCL_SPEC
            Nonterminal : VOID_NODE
                Terminal : main
                Nonterminal : FORMAL PARA
Nonterminal : COMPOUND_ST
    Nonterminal : DCL_LIST
        Nonterminal : DCL
            Nonterminal : DCL_SPEC
                Nonterminal : INT_NODE
                Nonterminal : DCL_ITEM
                    Nonterminal : ARRAY_VAR
                        Terminal : list
                        Terminal : 100
                Nonterminal : DCL
                    Nonterminal : DCL_SPEC
                        Nonterminal : INT_NODE
                    Nonterminal : DCL_ITEM
                        Nonterminal : SIMPLE_VAR
                            Terminal : element
                Nonterminal : DCL
                    Nonterminal : DCL_SPEC
                        Nonterminal : INT_NODE
                    Nonterminal : DCL_ITEM
                        Nonterminal : SIMPLE_VAR
                            Terminal : total
                Nonterminal : DCL_ITEM
                    Nonterminal : SIMPLE_VAR
                        Terminal : i
                Nonterminal : DCL_ITEM
                    Nonterminal : SIMPLE_VAR
                        Terminal : top
        Nonterminal : DCL
            Nonterminal : DCL_SPEC
                Nonterminal : INT_NODE
            Nonterminal : DCL_ITEM
                Nonterminal : SIMPLE_VAR
                    Terminal : temp
Nonterminal : STAT_LIST
    Nonterminal : EXP_ST
        Nonterminal : ASSIGN_OP
            Terminal : i
            Terminal : 1
    Nonterminal : EXP_ST
        Nonterminal : CALL
            Terminal : read
            Terminal : element
Nonterminal : WHILE_ST
    Nonterminal : NE
        Terminal : element
Terminal : 1
Nonterminal : WHILE_ST
    Nonterminal : GT
        Terminal : top
        Terminal : i
Nonterminal : COMPOUND_ST
    Nonterminal : DCL_LIST
        Nonterminal : STAT_LIST
            Nonterminal : EXP_ST
                Nonterminal : ASSIGN_OP
                    Terminal : i
                    Terminal : 1
                Nonterminal : WHILE_ST
                    Nonterminal : LT
                        Terminal : i
                        Terminal : top
                Nonterminal : COMPOUND_ST
                    Nonterminal : DCL_LIST
                        Nonterminal : STAT_LIST
                            Nonterminal : IF_ST
                                Nonterminal : GT
                                    Nonterminal : INDEX
                                        Terminal : list
                                        Terminal : i
                                    Nonterminal : INDEX
                                        Terminal : list
                                        Terminal : i
                                    Nonterminal : ADD
                                        Terminal : i
                                        Terminal : 1
                                Nonterminal : COMPOUND_ST
                                    Nonterminal : DCL_LIST
                                        Nonterminal : STAT_LIST
                                            Nonterminal : EXP_ST
                                                Nonterminal : ASSIGN_OP
                                                    Terminal : temp
                                                    Nonterminal : INDEX
                                                        Terminal : list
                                                        Terminal : i
                                                Nonterminal : EXP_ST
                                                    Nonterminal : ASSIGN_OP
                                                        Nonterminal : INDEX
                                                            Terminal : list
                                                            Terminal : i
                                                        Nonterminal : INDEX
                                                            Terminal : list
                                                            Terminal : i
                                                        Nonterminal : ADD
                                                            Terminal : i
                                                            Terminal : 1
                                                Nonterminal : EXP_ST
                                                    Nonterminal : ASSIGN_OP
                                                        Nonterminal : INDEX
                                                            Terminal : list
                                                            Terminal : i
                                                        Nonterminal : ADD
                                                            Terminal : i
                                                            Terminal : 1
                                                Nonterminal : EXP_ST
                                                    Nonterminal : PRE_INC
                                                        Terminal : i
                                                        Nonterminal : POST_DEC
                                                            Terminal : top
                                                        Nonterminal : EXP_ST
                                                            Nonterminal : ASSIGN_OP
                                                                Terminal : i
                                                                Terminal : 1
                                                        Nonterminal : WHILE_ST
                                                            Nonterminal : LE
                                                                Terminal : i
                                                                Terminal : total
                                                        Nonterminal : COMPOUND_ST
                                                            Nonterminal : DCL_LIST
                                                                Nonterminal : STAT_LIST
                                                                    Nonterminal : EXP_ST
                                                                        Nonterminal : CALL
                                                                            Terminal : write
                                                                            Nonterminal : INDEX
                                                                                Terminal : list
                                                                                Terminal : i
                                                                        Nonterminal : EXP_ST
                                                                            Nonterminal : PRE_INC
                                                                                Terminal : i
Nonterminal : code_generator -- bash -- 149x56
```

```
code_generator — bash — 149x56

$4
str 1 103
nop
lod 1 103
lod 1 104
lt
fjp $$5
lod 1 103
lda 1 1
add
ldi
lod 1 103
ldc 1
add
lda 1 1
add
ldi
gt
fjp $$6
lod 1 103
lda 1 1
add
ldi
str 1 105
lod 1 103
lda 1 1
add
lod 1 103
ldc 1
add
lda 1 1
add
ldi
sti
lod 1 103
ldc 1
add
lda 1 1
add
lod 1 105
sti
nop
lod 1 103
inc
str 1 103
ujp $$4
$$5
nop
lod 1 104
dec
str 1 104
ujp $$2
$$3
nop
ldc 1
str 1 103
$$7
nop
lod 1 103
lod 1 102
$$3
nop
ldc 1
str 1 103
$$7
nop
lod 1 103
lod 1 102
le
fjp $$8
ldp
lod 1 103
lda 1 1
add
ldi
call write
lod 1 103
inc
str 1 103
ujp $$7
$$8
nop
ret
end
bgn 0
ldp
call main
end
*** end of Mini C Compiler
```

code

```
==== start of ICG

processDeclaration!
processFunctionHeader

processFunction

Nonterminal : FUNC_DEF
    Nonterminal : FUNC_HEAD
        Nonterminal : DCL_SPEC
            Nonterminal : VOID_NODE
                Terminal : main
                Nonterminal : FORMAL PARA
Nonterminal : COMPOUND_ST
    Nonterminal : DCL_LIST
        Nonterminal : DCL
            Nonterminal : DCL_SPEC
                Nonterminal : INT_NODE
                    Nonterminal : DCL_ITEM
                        Nonterminal : SIMPLE_VAR
                            Terminal : i
                            Nonterminal : DCL_ITEM
                                Nonterminal : SIMPLE_VAR
                                    Terminal : j
                            Nonterminal : DCL_ITEM
                                Nonterminal : SIMPLE_VAR
                                    Terminal : k
            Nonterminal : DCL
                Nonterminal : DCL_SPEC
                    Nonterminal : INT_NODE
                        Nonterminal : DCL_ITEM
                            Nonterminal : SIMPLE_VAR
                                Terminal : rem
                            Nonterminal : DCL_ITEM
                                Nonterminal : SIMPLE_VAR
                                    Terminal : sum
Nonterminal : STAT_LIST
    Nonterminal : EXP_ST
        Nonterminal : ASSIGN_OP
            Terminal : i
            Terminal : 2
    Nonterminal : WHILE_ST
        Nonterminal : LE
            Terminal : i
            Terminal : max
    Nonterminal : COMPOUND_ST
        Nonterminal : DCL_LIST
            Nonterminal : STAT_LIST
                Nonterminal : EXP_ST
                    Nonterminal : ASSIGN_OP
                        Terminal : sum
                        Terminal : 0
                Nonterminal : EXP_ST
                    Nonterminal : ASSIGN_OP
                        Terminal : j
                        Terminal : k
            Nonterminal : COMPOUND_ST
                Nonterminal : DCL_LIST
                    Nonterminal : STAT_LIST
                        Nonterminal : EXP_ST
                            Nonterminal : ASSIGN_OP
                                Terminal : rem
                                Nonterminal : MOD
                                    Terminal : i
                                    Terminal : j
            Nonterminal : IF_ST
                Nonterminal : EQ
                    Terminal : rem
                    Terminal : 0
                Nonterminal : EXP_ST
                    Nonterminal : ADD_ASSIGN
                        Terminal : sum
                        Terminal : j
                Nonterminal : EXP_ST
                    Nonterminal : PRE_INC
                        Terminal : j
            Nonterminal : IF_ST
                Nonterminal : EQ
                    Terminal : i
                    Terminal : sum
                Nonterminal : EXP_ST
                    Nonterminal : CALL
                        Terminal : write
                        Terminal : i
            Nonterminal : EXP_ST
                Nonterminal : PRE_INC
                    Terminal : i
```

```
code_generator — bash — 149x56
ldc 500
le
fjp $$1
ldc 0
str 1 5
lod 1 1
ldc 2
div
str 1 3
ldc 1
str 1 2
$$2
nop
lod 1 2
lod 1 3
le
fjp $$3
lod 1 1
lod 1 2
mod
str 1 4
lod 1 4
ldc 0
eq
fjp $$4
lod 1 5
lod 1 2
add
str 1 5
$$4
nop
lod 1 2
inc
str 1 2
ujp $$2
$$3
nop
lod 1 1
lod 1 5
eq
fjp $$5
ldp
lod 1 1
call write
$$5
nop
lod 1 1
inc
str 1 1
ujp $$0
$$1
nop
ret
end
bgn 0
ldp
call main
end
*** end of Mini C Compiler
ihaghyeon-ui-MacBook-Pro:code_generator leehakhyun$
ihaghyeon-ui-MacBook-Pro:code_generator leehakhyun$
```

```
z:\Users\leehakhyun\Documents\Compiler\Compiler\code_generator>uicodei.exe bubbleSort.uco bubbleSort.lst
== Assembling ...
== Executing ...
== Result
2 7 8 5 8 5 8 4 4 5 3 45 3 35 4 3 21 54 7 0
2 3 3 3 4 4 4 5 5 5 7 7 8 8 8 21 35 45 54

z:\Users\leehakhyun\Documents\Compiler\Compiler\code_generator>uicodei.exe perfect.uco perfect.lst
== Assembling ...
== Executing ...
== Result
6 28 496
```

윈도우 실행파일과 unix용 실행파일 모두 첨부하였습니다!

