**컴파일러 과제2**

1976393 최윤지

2076278 윤지윤

2076285 이강희

**1) .l 파일. 모든 .c 파일(lex.yy.c 제외), 모든 .h 파일**

|  |
| --- |
| tn.h |
| /\*  \* tn.h – token type  \* progrmmer – 최윤지, 윤지윤, 이강희  \* date – 04/27/2023  \*/  // Token 에 관한 정의  enum tokentypes {  TEOF, TCONST, TELSE, TIF, TINT, TRETURN, TVOID, TWHILE, TADD,TSUB,TMUL,TDIV,TMOD,TOP, TEQUAL, TNOTEQU, TLESSE,  TGREATE,TNOT, TAND, TOR, TINC, TDEC, TADDASSIGN, TSUBASSIGN, TMULASSIGN, TDIVASSIGN, TMODASSIGN, TSPACE,TTAB,TLINE,  TIDENT, TNUMBER,TBRASL,TBRASR,TBRAML,TBRAMR,TBRALL,TBRALR,TCOMMA,TSEMICOLON,TCMT,ILLSP,TOOLONG,ILLID  }; |
| glob.h |
| /\*  \* glob.h – global variable for the project  \* progrmmer – 최윤지, 윤지윤, 이강희  \* date – 04 / 27 / 2023  \*/  #define MaxIdentLen 10 // identifier의 최대 길이  extern char\* yytext; // 렉스 |

|  |
| --- |
| scanner.l |
| % {  /\* scanner.l - lexical analyzer for the MiniC  \* programmer – 최윤지, 윤지윤, 이강희  \* date – 04/27/2023  \*/  #include <stdio.h>  #include <stdlib.h>  #include "tn.h"  #include "glob.h"  extern reporterror(enum tnumber tn); /\*에러 출력하는 함수\*/  %}  A[a - zA - Z\_]  D[0 - 9]  % %  /\*  \* Keyword  \*/  "const" return(TCONST);  "else" return(TELSE);  "if" return(TIF);  "int" return(TINT);  "return" return(TRETURN);  "void" return(TVOID);  "while" return(TWHILE);  /\*  \* 주석  \*/  "//".\* return(TCMT);  /\*  \* Identifier MacIdentLen(10)보다 크면 에러  \*/  {A}({ A } | {D})\* {  if (yyleng > MaxIdentLen) {  reporterror(TOOLONG);  }  else {  SymbolTable();  }  }  /\*  \* 숫자로 시작하는 Identifier는 에러  \*/  {D}({ A } | {D})\* { A }({ A } | {D})\* {  reporterror(ILLID);  }  /\*  \* constant  \*/  {D} + return(TNUMBER);  /\*  \* operator  \*/  "+" return(TADD);  "-" return(TSUB);  "\*" return(TMUL);  "/" return(TDIV);  "%" return(TMOD);  "=" return(TOP);  "+=" return(TADDASSIGN);  "-=" return(TSUBASSIGN);  "\*=" return(TMULASSIGN);  "/=" return(TDIVASSIGN);  "%=" return(TMODASSIGN);  "!" return(TNOT);  "&&" return(TAND);  "||" return(TOR);  "==" return(TEQUAL);  "!=" return(TNOTEQU);  "<=" return(TLESSE);  ">=" return(TGREATE);  "<" return(TLESSE);  ">" return(TGREATE);  "++" return(TINC);  "--" return(TDEC);  /\*  \* Special Symbols  \*/  "(" return(TBRASL);  ")" return(TBRASR);  "{" return(TBRAML);  "}" return(TBRAMR);  "[" return(TBRALL);  "]" return(TBRALR);  "," return(TCOMMA);  ";" return(TSEMICOLON);  /\*  \* white spaces  \*/  "\t" return(TTAB);  " " return(TSPACE);  /\*  \* 줄바꿈 시 줄 갯수 1 증가 + '\n'은 white space 이므로 TLINE 리턴  \*/  "\n" {linenum++;  return(TLINE); }  /\*  \* 나머지 경우는 에러, 유효하지 않은 문자가 포함된  \*/  . reporterror(ILLSP);  %%  int yywrap()  {  return 1;  } |

|  |
| --- |
| symtable.c |
| /\*  \* symtable.c – identifier들의 hashcode를 계산하고 st에 삽입  \* progrmmer – 최윤지, 윤지윤, 이강희  \* date – 04 / 27 / 2023  \*/  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include "tn.h"  extern int linenum; //라인 줄 수  extern int yyleng;  extern char\* yytext;  extern reporterror(enum tokentype tn);  #define STsize 1000 //string table 사이즈  #define HTsize 100 //hash table 사이즈  #define FALSE 0  #define TRUE 1  typedef struct HTentry\* HTpointer;  typedef struct HTentry {  int index; //ST에 있는 identifier의 인덱스  HTpointer next; //다음 identifier에 대한 포인터  }HTentry;  HTpointer HT[HTsize];  char ST[STsize];  int nextid = 0; //현재 identifier  int nextfree = 0; //ST에서 가능한 다음 index 번호the next available index of ST  int hashcode; //identifier의 hash code  int sameid; //identifier의 첫번째 index  int found; //identifier가 ST에 이미 존재할 경우  //identifier의 hash code들을 모두 더하여 modulo연산을 수행  void ComputeHS(int nid, int nfree) {  int code, i;  code = 0;  for (i = nid; i < nfree - 1; i++) {  code += (int)ST[i]; //hash code더하기  }  hashcode = code % HTsize; //modulo  }  //identifier가 이전에 삽입된 적 있는지 살펴봄 이미 존재한다면 ST에 이미 존재하는 시작 index를 불러옴  void LookupHS(int nid, int hscode) {  HTpointer here;  int i, j;  found = FALSE;  if (HT[hscode] != NULL) {  here = HT[hscode];  while (here != NULL && found == FALSE) {  found = TRUE;  i = here->index;  j = nid;  sameid = i;  while (ST[i] != '\0' && found == TRUE) {  if (ST[i] != ST[j]) {  found = FALSE;  }  else {  i++;  j++;  }  }  here = here->next;  }  }  }  //hash table에 새로운 identifier를 추가  //null이었으면 바로 추가하고, 이미 존재한다면 linked list로 연결  void ADDHT(int hscode) {  HTpointer ptr;  ptr = (HTpointer)malloc(sizeof(ptr));  ptr->index = nextid;  ptr->next = HT[hscode];  HT[hscode] = ptr;  }  //메인 함수  int SymbolTable()  {  nextid = nextfree;  // yytext 한글자씩 ST에 넣음  for (int i = 0; i < yyleng; i++) {  ST[nextfree++] = yytext[i];  }    // identifier를 다 넣은 후  ST[nextfree++] = '\0';  // 각 identifer에 대해 hastable에 이미 있는지를 확인  ComputeHS(nextid, nextfree);  LookupHS(nextid, hashcode);    //처음 등장한 identifier일 경우  if (!found) {  // line number, Token-type, ST-index(identifier인 경우), token 순으로 출력  printf("%-20d %-30s %-20d %-30s\n", linenum, "Identifier", nextid, yytext);  ADDHT(hashcode); // HT 삽입  }  //identifier가 이미 존재하는 경우  else {  // line number, Token-type, ST-index(identifier인 경우), token 순으로 출력  printf("%-20d %-30s %-20d %-30s\n", linenum, "Identifier", sameid, yytext);  nextfree = nextid; //이번에 ST에 읽은 identifier를 다음에 덮어씀  }  } |

|  |
| --- |
| reporterror.c |
| /\*  \* reporterror.c – error에 대한 report를 따로 처리  \* progrmmer – 최윤지, 윤지윤, 이강희  \* date – 04 / 27 / 2023  \*/  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include "tn.h"  extern int linenum;  extern char\* yytext;  extern int cErrors; //에러 개수  //token type별로 에러를 나누고 해당 에러문 출력  void reporterror(enum tokentype tn) {  switch (tn) {  //ILLSP : 유효하지 않은 문자가 섞인 경우  case ILLSP:  printf("%-20d %-30s %-20s %s is illegal IDENT\n", linenum, "\*\*\*Error\*\*\*", " ", yytext);  cErrors++;  break;  //ILLID : 숫자로 시작하는 경우  case ILLID:  printf("%-20d %-30s %-20s %-1s %-30s\n", linenum, "\*\*\*Error\*\*\*", " ", yytext, " starts with digit");  cErrors++;  break;  //TOOLONG : 10자 이상인 경우  case TOOLONG:  printf("%-20d %-30s %-20s %-30s %-30s\n", linenum, "\*\*\*Error\*\*\*", " ", yytext, "too long IDENT");  cErrors++;  break;  }  } |

|  |
| --- |
| main.c |
| /\*  \* main.c - 각 token에 대한 출력  \* programmer – 최윤지, 윤지윤, 이강희  \* date – 04/27/2023  \*/  #include <stdio.h>  #include <stdlib.h>  #include "tn.h"  extern yylex();  extern char\* yytext;  extern int linenum;  extern int cErrors;  /\*  \* printToken() -- token type 별로 출력  \* switch문에 tn을 넣어 token을 구분하고 출력  \* line number, token type, ST index (ident인 경우만), Token을 프린트  \*/  void printToken(enum tokentypes tn) {  switch (tn) {  // 주석일 때  case TCMT:  printf("%-20d %-30s %-20s %-30s\n", linenum, "comment", " ", yytext);  break;  // keyword일 때  case TCONST: // CONST  printf("%-20d %-30s %-20s %-30s\n", linenum, "keyword", " ", yytext);  break;  case TELSE: // ELSE  printf("%-20d %-30s %-20s %-30s\n", linenum, "keyword", " ", yytext);  break;  case TIF: // IF  printf("%-20d %-30s %-20s %-30s\n", linenum, "keyword", " ", yytext);  break;  case TINT: // INT  printf("%-20d %-30s %-20s %-30s\n", linenum, "keyword", " ", yytext);  break;  case TRETURN: // RETURN  printf("%-20d %-30s %-20s %-30s\n", linenum, "keyword", " ", yytext);  break;  case TVOID: // VOID  printf("%-20d %-30s %-20s %-30s\n", linenum, "keyword", " ", yytext);  break;  case TWHILE: // WHILE  printf("%-20d %-30s %-20s %-30s\n", linenum, "keyword", " ", yytext);  break;  // operator일 때  case TADD: // +  printf("%-20d %-30s %-20s %-30s\n", linenum, "addition", " ", yytext);  break;  case TSUB: // -  printf("%-20d %-30s %-20s %-30s\n", linenum, "subtraction", " ", yytext);  break;  case TMUL: // \*  printf("%-20d %-30s %-20s %-30s\n", linenum, "multiplication", " ", yytext);  break;  case TDIV: // /  printf("%-20d %-30s %-20s %-30s\n", linenum, "division", " ", yytext);  break;  case TMOD: // %  printf("%-20d %-30s %-20s %-30s\n", linenum, "modulo", " ", yytext);  break;  case TOP: // =  printf("%-20d %-30s %-20s %-30s\n", linenum, "assign", " ", yytext);  break;  case TEQUAL: // ==  printf("%-20d %-30s %-20s %-30s\n", linenum, "equal", " ", yytext);  break;  case TNOTEQU: // !=  printf("%-20d %-30s %-20s %-30s\n", linenum, "not equal", " ", yytext);  break;  case TLESSE: // <=  printf("%-20d %-30s %-20s %-30s\n", linenum, "less", " ", yytext);  break;  case TGREATE: // >=  printf("%-20d %-30s %-20s %-30s\n", linenum, "grater", " ", yytext);  break;  case TNOT: // !  printf("%-20d %-30s %-20s %-30s\n", linenum, "not", " ", yytext);  break;  case TAND: // &&  printf("%-20d %-30s %-20s %-30s\n", linenum, "and", " ", yytext);  break;  case TOR: // ||  printf("%-20d %-30s %-20s %-30s\n", linenum, "or", " ", yytext);  break;  case TINC: // ++  printf("%-20d %-30s %-20s %-30s\n", linenum, "increase one", " ", yytext);  break;  case TDEC: // --  printf("%-20d %-30s %-20s %-30s\n", linenum, "decrease one", " ", yytext);  break;  case TADDASSIGN: // +=  printf("%-20d %-30s %-20s %-30s\n", linenum, "add assign", " ", yytext);  break;  case TSUBASSIGN: // -=  printf("%-20d %-30s %-20s %-30s\n", linenum, "sub assign", " ", yytext);  break;  case TMULASSIGN: // \*=  printf("%-20d %-30s %-20s %-30s\n", linenum, "mul assign", " ", yytext);  break;  case TDIVASSIGN: // /=  printf("%-20d %-30s %-20s %-30s\n", linenum, "div assign", " ", yytext);  break;  case TMODASSIGN: // %=  printf("%-20d %-30s %-20s %-30s\n", linenum, "mod assign", " ", yytext);  break;  // number일 때  case TNUMBER:  printf("%-20d %-30s %-20s %-30s\n", linenum, "number", " ", yytext);  break;  // 괄호일 때  case TBRASL: // (  printf("%-20d %-30s %-20s %-30s\n", linenum, "open small bracket", " ", yytext);  break;  case TBRASR: // )  printf("%-20d %-30s %-20s %-30s\n", linenum, "close small bracket", " ", yytext);  break;  case TBRAML: // {  printf("%-20d %-30s %-20s %-30s\n", linenum, "open middle bracket", " ", yytext);  break;  case TBRAMR: // }  printf("%-20d %-30s %-20s %-30s\n", linenum, "close middle bracket", " ", yytext);  break;  case TBRALL: // [  printf("%-20d %-30s %-20s %-30s\n", linenum, "open large bracket", " ", yytext);  break;  case TBRALR: // ]  printf("%-20d %-30s %-20s %-30s\n", linenum, "close large bracket", " ", yytext);  break;  // 콤마일 때  case TCOMMA: // ,  printf("%-20d %-30s %-20s %-30s\n", linenum, "comma", " ", yytext);  break;  // 세미콜론일 때  case TSEMICOLON: // ;  printf("%-20d %-30s %-20s %-30s\n", linenum, "semicolon", " ", yytext);  break;  // White spaces  case TLINE: // 줄바꿈  printf("%-20d %-30s %-20s %-30s\n", linenum, "white space-line", " ", "\\n");  break;  case TTAB: // tab  printf("%-20d %-30s %-20s %-30s\n", linenum, "white space-tab", " ", "\\t");  break;  case TSPACE: // 공백 문자  printf("%-20d %-30s %-20s %c %c\n", linenum, "white space-space", " ", '\"', '\"');  break;  }  }  /\*  \* main() -- yylex()로 토큰을 받아와 출력 함수를 호출, Error 개수를 세어 출력문 구성  \* switch문에 tn을 넣어 token을 구분하고 출력  \* line number, token type, ST index (ident인 경우만), Token을 프린트  \*/  void main()  {  // token type을 저장할 변수 선언  enum tokentypes tn;  // print header  printf("========================================================================================================================\n");  printf("%-20s %-30s %-20s %-30s\n", "Line number", "Token type", "ST-index", "Token");  printf("========================================================================================================================\n");    // token을 하나씩 받아와서 출력 처리  while((tn=yylex()) != TEOF){  // line number, token type, ST index (ident인 경우만), Token을 프린트  printToken(tn);  }  // Error 개수 출력  if (cErrors == 0) { // Error가 0개일 때  printf("========================================================================================================================\n");  printf("No errors detected\n");  printf("========================================================================================================================\n");  }  else { // Error가 1개 이상일 때, Error 개수 출력  printf("========================================================================================================================\n");  printf("%d errors detected\n", cErrors);  printf("========================================================================================================================\n");  }  // Programmer  printf("\n1976393 최윤지\n2076278 윤지윤\n2076285 이강희");  } |

|  |
| --- |
| testdata1.dat |
|  |

**2) 채점용 테스트데이터 5개에 대한 output 캡처본 => 총 5개**

|  |
| --- |
| testdata2.dat – (1) |
|  |

|  |
| --- |
| testdata2.dat – (2) |
|  |

|  |
| --- |
| testdata3.dat |
|  |

|  |
| --- |
| testdata4.dat – (1) |
|  |

|  |
| --- |
| testdata4.dat – (2) |
|  |

|  |
| --- |
| testdata5.dat – (1) |
|  |

|  |
| --- |
| testdata5.dat – (2) |
|  |

|  |
| --- |
| 직접 만든 테스트데이터 1 (Error) |
|  |

**3) 직접 만든 에러 있는 테스트데이터 3개에 대한 input, output 캡쳐본**

|  |
| --- |
| 직접 만든 테스트데이터 2 (Error) |
|  |

|  |
| --- |
| 직접 만든 테스트데이터 3 (Error) |
|  |

|  |
| --- |
| 직접 만든 테스트데이터 1 (noError) |
|  |

**4) 직접 만든 에러 없는 테스트데이터 3개에 대한 input, output 캡쳐본**

|  |
| --- |
| 직접 만든 테스트데이터 2 (noError) |
|  |

|  |
| --- |
| 직접 만든 테스트데이터 3 (noError) |
|  |

**5) 과제에 대한 설명**

|  |
| --- |
| 1. Identifier인 경우 Symbol Table에 넣어 줌.  2. 각 토큰이 인식되었을 때 line number, token type, ST index (identifier인 경우만), Token 출력  3. 정의되어 있지 않은 token이 인식되었을 경우 error 출력(ILLSP)  4. 숫자로 시작하는 identifier의 경우 error 출력 (ILLID)  5. 10자 초과하는 identifier의 경우 error 출력 (TOOLONG)  6. Token을 공백의 경우 " ", 탭의 경우"\t", 줄 바꿈의 경우"\n" 로 출력하도록 설정하여 각 Token이 무엇인지 더욱 명확히 구분하도록 함. |