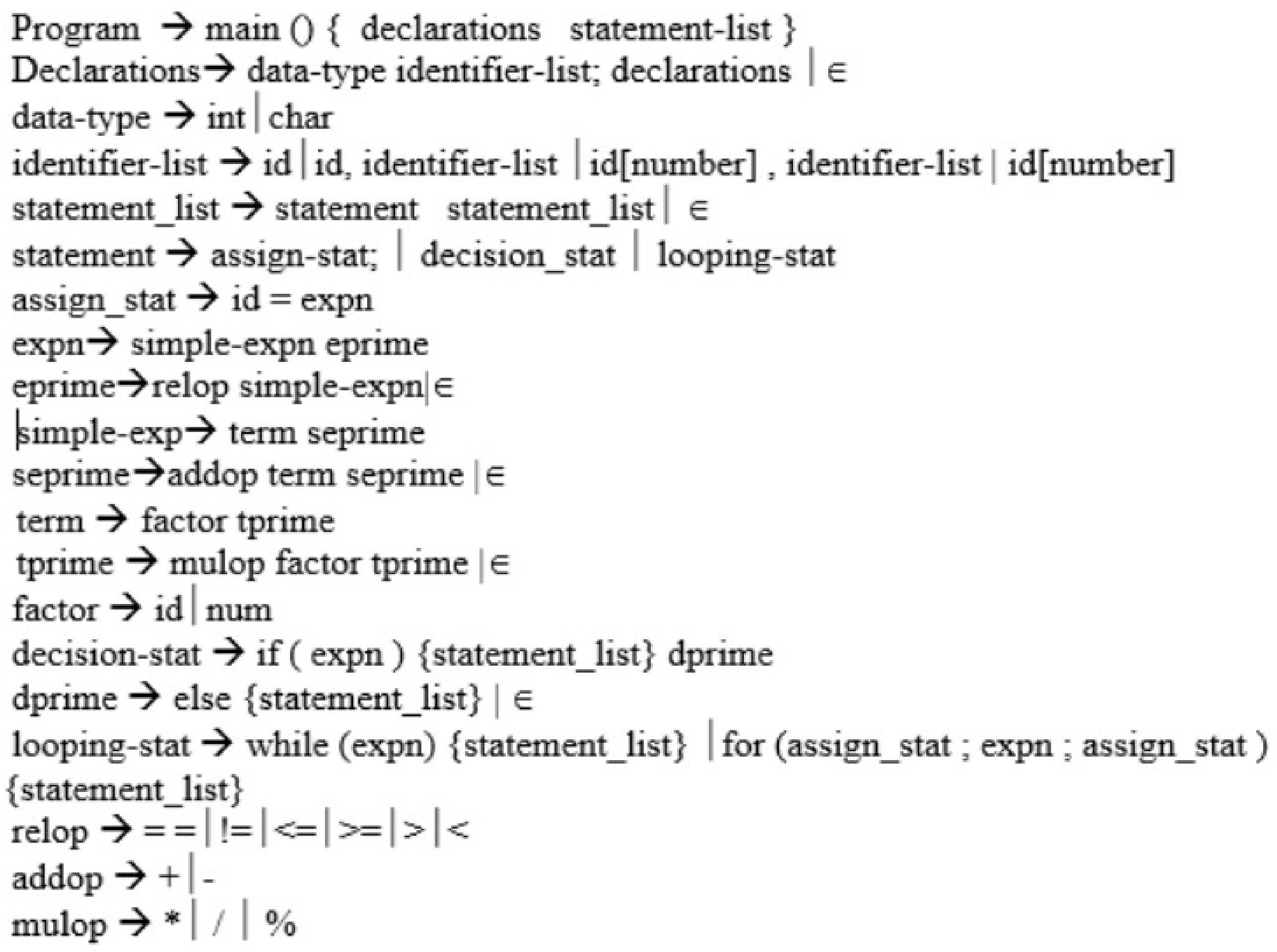
CD Lab
Rhea Adhikari
190905156
Lab8

To design a recursive descent parser for a C grammar. (Declaration statements, array declaration, looping, decision making statements, etc)



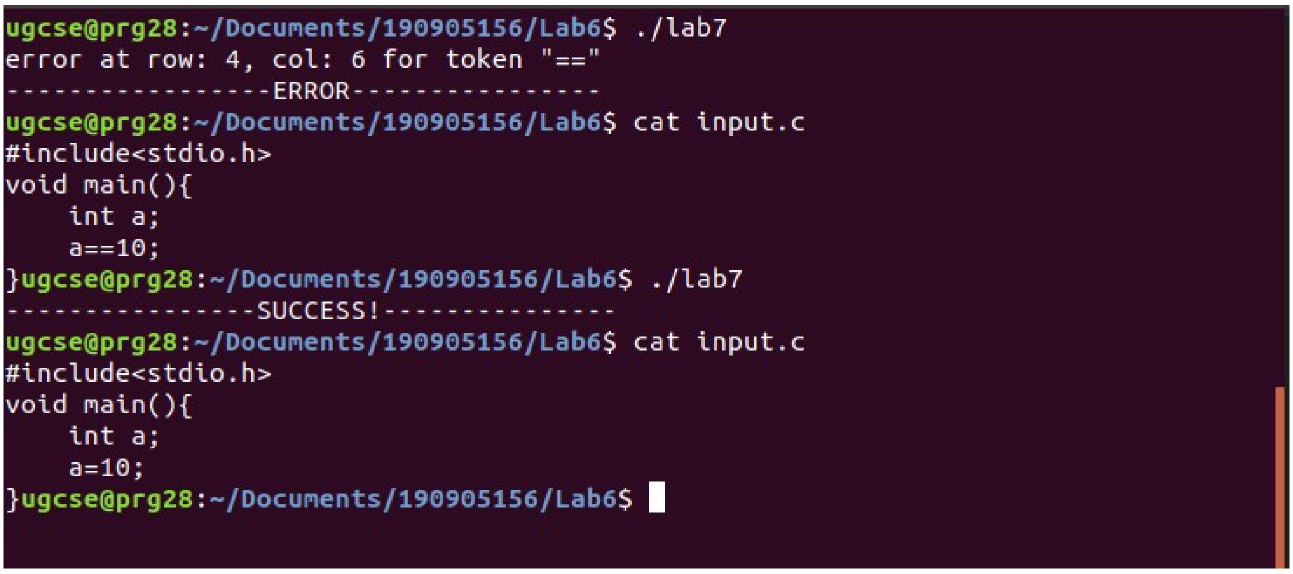
Program → main () { declarations statement-list }
Declarations→ data-type identifier-list; declarations | ∈
data-type → int | char
identifier-list → id | id, identifier-list | id[number] , identifier-list | id[number]
statement_list → statement statement_list | ∈
statement → assign-stat; | decision_stat | looping-stat
assign_stat → id = expn
expn→ simple-expn eprime
eprime→relop simple-expn| ∈
simple-exp→ term seprime
seprime→addop term seprime | ∈
term → factor tprime
tprime → mulop factor tprime | ∈
factor → id | num
decision-stat → if (expn) {statement_list} dprime
dprime → else {statement_list} | ∈
looping-stat → while (expn) {statement_list} | for (assign_stat ; expn ; assign_stat) {statement_list}
relop → = = | != | <= | >= | > | <
addop → + | -
mulop → * | / | %

Program

```
#include "la.h"

void program();
void declarations();
void datatype();
void idList();
void idListprime();
void idListprimePrime();
void stmtList();
void stmt();
void assignStat();
void expn();
void eprime();
void simpleExpn();
void seprime();
void term();
void tprime();
void factor();
void relop();
void addOp();
void mulOp();
void decStat();
void dPrime();
void loopStat();
void invalid();

struct token tkn;

FILE * file_ptr;

void invalid() {
  printf("Error at row: %d, col: %d for token \"%s\"\n", tkn.row, tkn.col, tkn.token_name);
  printf("---------ERROR---------\n ");
  exit(0);
}
void program() {
  tkn = getNextToken(file_ptr);
  if (strcmp("int", tkn.token_name) == 0 || strcmp("void", tkn.token_name) == 0) {
    tkn = getNextToken(file_ptr);
    if (strcmp(tkn.token_name, "main") == 0) {
      tkn = getNextToken(file_ptr);
      if (strcmp(tkn.token_name, "(") == 0) {
        tkn = getNextToken(file_ptr);
        if (strcmp(tkn.token_name, ")") == 0) {
          tkn = getNextToken(file_ptr);
          if (strcmp(tkn.token_name, "{") == 0) {
            tkn = getNextToken(file_ptr);
            declarations();
            stmtList();
            if (strcmp(tkn.token_name, "}") == 0) {
              printf("----------Successfully compiled------------ \n");
              return;
            } else {
              invalid();
            }
          } else {
            invalid();
          }
        } else {
          invalid();
        }
      } else {
        invalid();
      }
    } else {
      invalid();
    }
  } else {
    invalid();
  }
}
void declarations() {
  if (isdtype(tkn.token_name) == 0)
    return;
  datatype();
  idList();
  if (strcmp(tkn.token_name, ";") == 0) {
    tkn = getNextToken(file_ptr);
    printf("%s", tkn.token_name);
    declarations();
  } else {
    invalid();
  }
}
void datatype() {
  if (strcmp(tkn.token_name, "int") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, "char") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else {
    invalid();
  }
}
void assignStat() {
  if (strcmp(tkn.type, "Identifier") == 0) {
    tkn = getNextToken(file_ptr);
    if (strcmp(tkn.token_name, "=") == 0) {
      tkn = getNextToken(file_ptr);
      expn();
    } else {
      invalid();
    }
  } else {
    printf("here");
    invalid();
  }
}
void idList() {
  if (strcmp(tkn.type, "Identifier") == 0) {
    printf("%s", tkn.token_name);
    tkn = getNextToken(file_ptr);
    printf("%s", tkn.token_name);
    idListprime();
  } else {
    invalid();
  }
}
void idListprime() {
  if (strcmp(tkn.token_name, ",") == 0) {
    tkn = getNextToken(file_ptr);
    idList();
  } else if (strcmp(tkn.token_name, "=") == 0) {
    printf("Entered here");
    tkn = getNextToken(file_ptr);
    expn();
  } else if (strcmp(tkn.token_name, "[") == 0) {
    tkn = getNextToken(file_ptr);
    if (strcmp(tkn.type, "Number") == 0) {
      tkn = getNextToken(file_ptr);
      if (strcmp(tkn.token_name, "]") == 0) {
        tkn = getNextToken(file_ptr);
        idListprimePrime();
      } else {
        printf("here 7");
        invalid();
      }
    } else {
      printf("here 8");
      invalid();
    }
  }
}
void idListprimePrime() {
  if (strcmp(tkn.token_name, ",") == 0) {
    tkn = getNextToken(file_ptr);
    idList();
  } else
    return;
}
void stmtList() {
  if (strcmp(tkn.type, "Identifier") == 0 || strcmp(tkn.token_name, "if") == 0 ||
    strcmp(tkn.token_name, "for") == 0 || strcmp(tkn.token_name, "while") == 0) {
    stmt();
    stmtList();
  }
  return;
}
void stmt() {
  if (strcmp(tkn.type, "Identifier") == 0) {
    assignStat();
    if (strcmp(tkn.token_name, ";") == 0) {
      tkn = getNextToken(file_ptr);
      return;
    } else {
      invalid();
    }
  } else if (strcmp(tkn.token_name, "if") == 0)
    decStat();
  else if ((strcmp(tkn.token_name, "while") == 0) || (strcmp(tkn.token_name, "for") == 0))
    loopStat();
  else {
    printf("%d.%d : Expected \" statement \"\n", tkn.row, tkn.col);
    exit(0);
  }
}
void expn() {
  simpleExpn();
  eprime();
}

void eprime() {
  if (strcmp(tkn.type, "RelationalOperator") != 0)
    return;
  relop();
  simpleExpn();
}

void simpleExpn() {
  term();
  seprime();
}

void seprime() {
  if ((strcmp(tkn.token_name, "+") != 0) && (strcmp(tkn.token_name, "-") != 0))
    return;
  addOp();
  term();
  seprime();
}

void term() {
  factor();
  tprime();
}

void tprime() {
  if ((strcmp(tkn.token_name, "*") != 0) && (strcmp(tkn.token_name, "/") != 0) &&
    (strcmp(tkn.token_name, "%") != 0))
    return;
  mulOp();
  factor();
  tprime();
}

void factor() {
  if (strcmp(tkn.type, "Identifier") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.type, "Number") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else {
    invalid();
  }
}

void relop() {
  if (strcmp(tkn.token_name, "==") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, "!=") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, "<=") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, ">=") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, "<") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, ">") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else {
    printf("here 4");
    invalid();
  }
}

void addOp() {
  if (strcmp(tkn.token_name, "+") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, "-") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else {
    printf("here 5");
    invalid();
  }
}

void mulOp() {
  if (strcmp(tkn.token_name, "*") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, "/") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else if (strcmp(tkn.token_name, "*") == 0) {
    tkn = getNextToken(file_ptr);
    return;
  } else {
    printf("here 6");
    invalid();
  }
}

void decStat() {
  if (strcmp(tkn.token_name, "if") == 0) {
    tkn = getNextToken(file_ptr);
    if (strcmp(tkn.token_name, "(") == 0) {
      tkn = getNextToken(file_ptr);
      expn();
      if (strcmp(tkn.token_name, ")") == 0) {
        tkn = getNextToken(file_ptr);
        if (strcmp(tkn.token_name, "{") == 0) {
          tkn = getNextToken(file_ptr);
          stmtList();
          if (strcmp(tkn.token_name, "}") == 0) {
            tkn = getNextToken(file_ptr);
            dPrime();
            return;
          } else {}
          invalid();
        } else {
          invalid();
        }
      } else {
        invalid();
      }
    } else {
      invalid();
    }
  } else {
    invalid();
  }
}

void dPrime() {
  if (strcmp(tkn.token_name, "else") == 0) {
    tkn = getNextToken(file_ptr);
    if (strcmp(tkn.token_name, "{") == 0) {
      tkn = getNextToken(file_ptr);
      stmtList();
      if (strcmp(tkn.token_name, "}") == 0) {
        tkn = getNextToken(file_ptr);
        return;
      } else {
        invalid();
      }
    } else {
      invalid();
    }
  } else
    return;
}

void loopStat() {
  if (strcmp(tkn.token_name, "while") == 0) {
    tkn = getNextToken(file_ptr);
    if (strcmp(tkn.token_name, "(") == 0) {
      tkn = getNextToken(file_ptr);
      expn();
      if (strcmp(tkn.token_name, ")") == 0) {
        tkn = getNextToken(file_ptr);
        if (strcmp(tkn.token_name, "{") == 0) {
          tkn = getNextToken(file_ptr);
          stmtList();
          if (strcmp(tkn.token_name, "}") == 0) {
            tkn = getNextToken(file_ptr);
            return;
          } else {
            invalid();
          }
        } else {
          invalid();
        }
      } else {
        invalid();
      }
    } else {
      invalid();
    }
  } else if (strcmp(tkn.token_name, "for") == 0) {
    tkn = getNextToken(file_ptr);
    if (strcmp(tkn.token_name, "(") == 0) {
      tkn = getNextToken(file_ptr);
      assignStat();
      if (strcmp(tkn.token_name, ";") == 0) {
        tkn = getNextToken(file_ptr);
        expn();
        if (strcmp(tkn.token_name, ";") == 0) {
          tkn = getNextToken(file_ptr);
          assignStat();
          if (strcmp(tkn.token_name, ")") == 0) {
            tkn = getNextToken(file_ptr);
            if (strcmp(tkn.token_name, "{") == 0) {
              tkn = getNextToken(file_ptr);
              stmtList();
              if (strcmp(tkn.token_name, "}") == 0) {
```

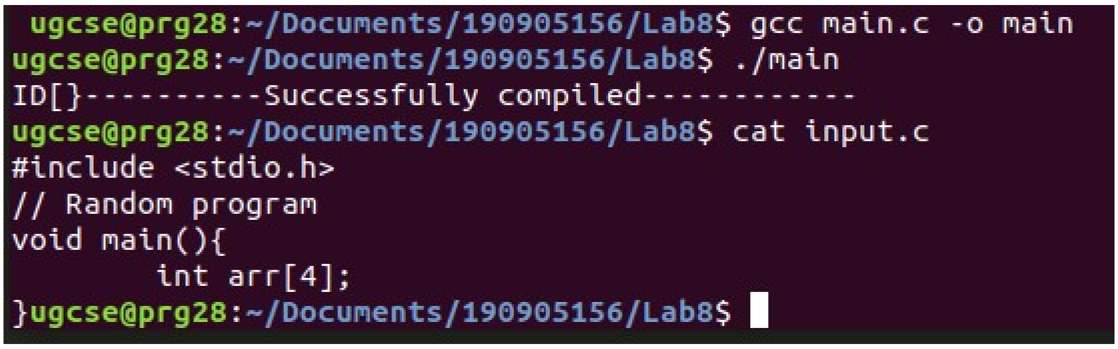
```
            tkn = getNextToken(file_ptr);
            return;
          } else {
            invalid();
          }
        } else {
          invalid();
        }
      } else {
        invalid();
      }
    } else {
      invalid();
    }
  } else {
    invalid();
  }
  } else {
    invalid();
  }
 }
}

int main() {
  file_ptr = fopen("input.c", "r");

  if (!file_ptr) {
    printf("-----------File does not open-----------\n");
    return 0;
  }

  program();
  fclose(file_ptr);
}
```

Week6

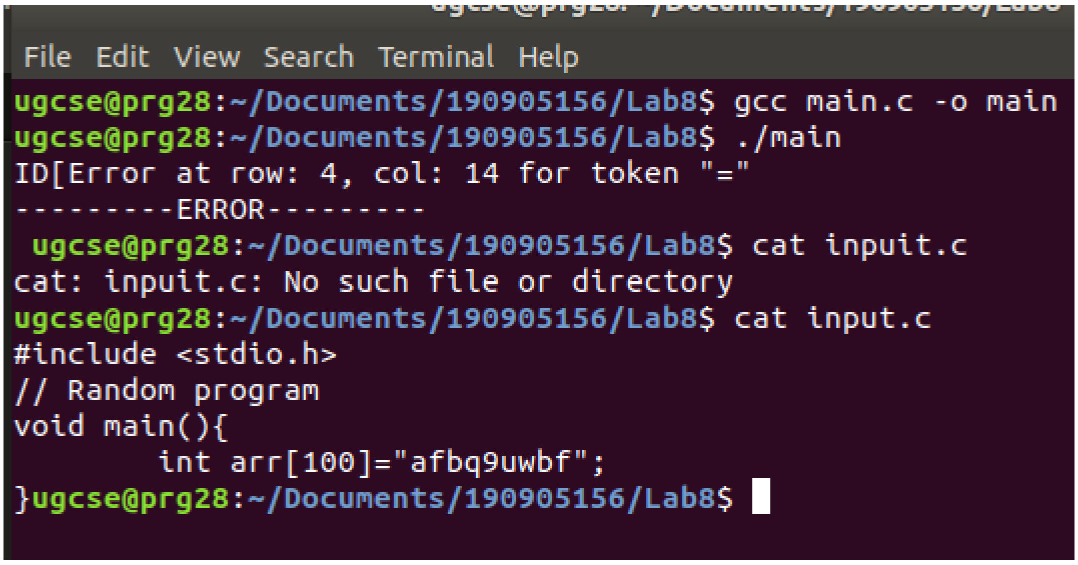


```
ugcse@prg28:~/Documents/190905156/Lab6$ ./lab7
error at row: 4, col: 6 for token "=="
-----------------ERROR----------------
ugcse@prg28:~/Documents/190905156/Lab6$ cat input.c
#include<stdio.h>
void main(){
    int a;
    a==10;
}ugcse@prg28:~/Documents/190905156/Lab6$ ./lab7
----------------SUCCESS!---------------
ugcse@prg28:~/Documents/190905156/Lab6$ cat input.c
#include<stdio.h>
void main(){
    int a;
    a=10;
}ugcse@prg28:~/Documents/190905156/Lab6$ 
```

Week 7 output

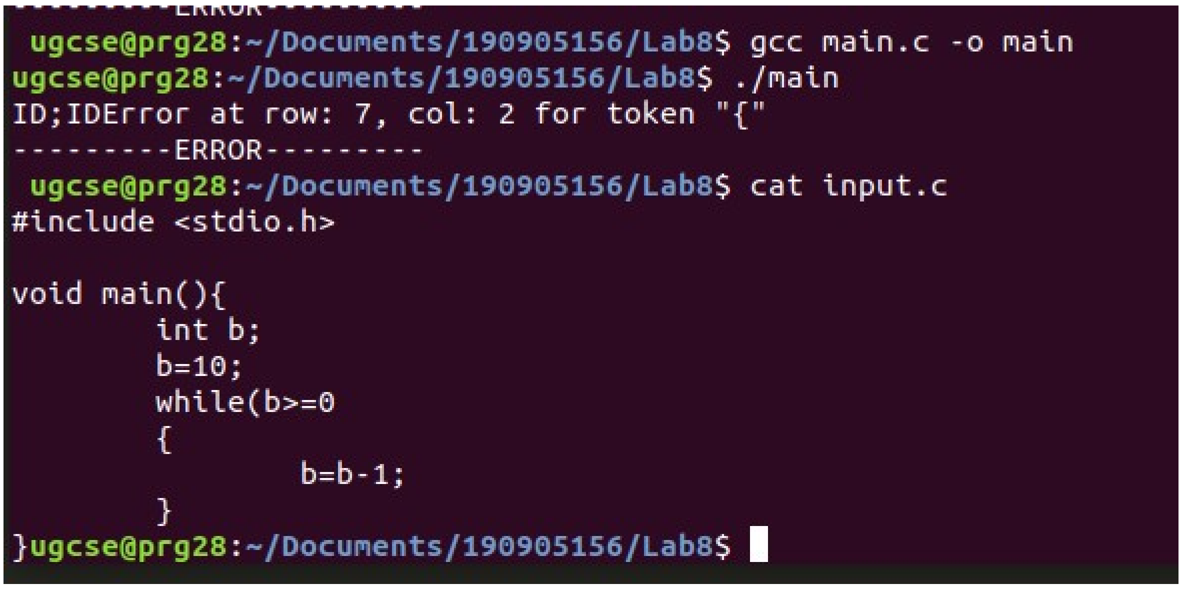


```
ugcse@prg28:~/Documents/190905156/Lab8$ gcc main.c -o main
ugcse@prg28:~/Documents/190905156/Lab8$ ./main
ID[}---------Successfully compiled-----------
ugcse@prg28:~/Documents/190905156/Lab8$ cat input.c
#include <stdio.h>
// Random program
void main(){
        int arr[4];
}ugcse@prg28:~/Documents/190905156/Lab8$
```

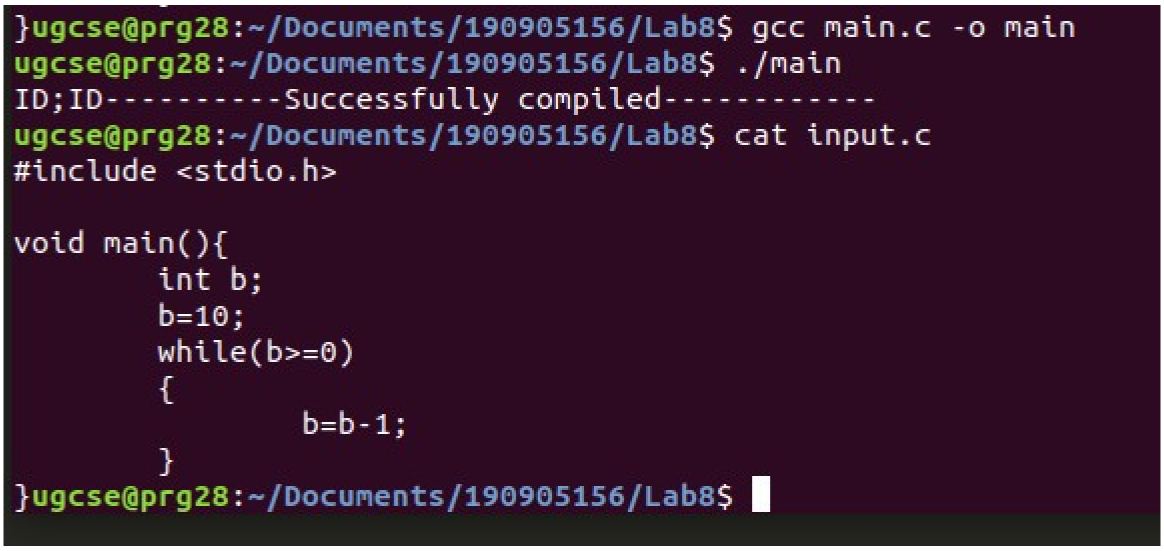


```
ugcse@prg28:~/Documents/190905156/Lab8$ gcc main.c -o main
ugcse@prg28:~/Documents/190905156/Lab8$ ./main
ID[Error at row: 4, col: 14 for token "="
---------ERROR---------
ugcse@prg28:~/Documents/190905156/Lab8$ cat inpuit.c
cat: inpuit.c: No such file or directory
ugcse@prg28:~/Documents/190905156/Lab8$ cat input.c
#include <stdio.h>
// Random program
void main(){
        int arr[100]="afbq9uwbf";
}ugcse@prg28:~/Documents/190905156/Lab8$
```

Week 8 output



```
ugcse@prg28:~/Documents/190905156/Lab8$ gcc main.c -o main
ugcse@prg28:~/Documents/190905156/Lab8$ ./main
ID;IDError at row: 7, col: 2 for token "{"
---------ERROR---------
ugcse@prg28:~/Documents/190905156/Lab8$ cat input.c
#include <stdio.h>

void main(){
        int b;
        b=10;
        while(b>=0
        {
                b=b-1;
        }
}ugcse@prg28:~/Documents/190905156/Lab8$
```



```
}ugcse@prg28:~/Documents/190905156/Lab8$ gcc main.c -o main
ugcse@prg28:~/Documents/190905156/Lab8$ ./main
ID;ID----------Successfully compiled------------
ugcse@prg28:~/Documents/190905156/Lab8$ cat input.c
#include <stdio.h>

void main(){
        int b;
        b=10;
        while(b>=0)
        {
                b=b-1;
        }
}ugcse@prg28:~/Documents/190905156/Lab8$
```

la.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define SIZEhash 10

struct token {
  char token_name[30];
  int row, col;
  char type[30];
};

struct symbTable {
  char lex_name[30];
  char dtype[30];
  unsigned int size;
}symVal[SIZEhash];

int symInd = 0;

static int row = 1, col = 1;

char buf[1024];

const char specialsymbols[] = {
  '?',
  ';',
  ':',
  ',',
  '(',
  ')',
  '{',
  '}',
  '[',
  ']'
};

const char * Keywords[] = {
  "main",
  "void",
  "const",
  "char",
  "int",
  "return",
  "for",
  "while",
  "do",
  "switch",
  "if",
  "else",
  "unsigned",
  "case",
  "break",
  "double",
  "float"
};

const char arithmeticsymbols[] = {
  '*'
};

char dtype[100];

const char * dataType[] = {
  "int",
  "float",
  "double",
  "char",
  "void"
};

int isdtype(char * w)
{
  for (int i = 0; i < sizeof(dataType) / sizeof(char * ); i++)
    if (strcmp(w, dataType[i]) == 0)
      return 1;
  return 0;
}

int isKeyword(const char * str) {
  for (int i = 0; i < sizeof(Keywords) / sizeof(char * ); i++) {
    if (strcmp(str, Keywords[i]) == 0)
      return 1;
  }
  return 0;
```

```
}

int charBelongsTo(int c,
  const char * arr) {
  int len;
  if (arr == specialsymbols)
    len = sizeof(specialsymbols) / sizeof(char);
  else if (arr == arithmeticsymbols)
    len = sizeof(arithmeticsymbols) / sizeof(char);
  for (int i = 0; i < len; i++) {
    if (c == arr[i])
      return 1;
  }
  return 0;
}

int dtypeSize(char dtype[30]) {
  int i;
  for (i = 0; i < sizeof(dataType) / sizeof(char * ); ++i) {
    if (strcmp(dtype, dataType[i]) == 0) {
      break;
    }
  }
  switch (i) {
  case 0:
    return sizeof(int);
    break;
  case 1:
    return sizeof(float);
    break;
  case 2:
    return sizeof(double);
  case 3:
    return sizeof(char);
  case 4:
    return sizeof(void);
  default:
    return -1;
    break;
  }
}

void fillToken(struct token * tkn, char c, int row, int col, char * type) {
  tkn -> row = row;
  tkn -> col = col;
  strcpy(tkn -> type, type);
  tkn -> token_name[0] = c;
  tkn -> token_name[1] = '\0';
}

int searchTable(char lex[30]) {
  for (int i = 0; i < symInd; ++i) {
    if (strcmp(lex, symVal[i].lex_name) == 0) {
      return 1;
    }
  }
  return 0;
}

void insertTable(char lex[30], char dtype[30], int flag) {
  int searchVal = searchTable(lex);
  if (searchVal == 1) {
    return;
  }
  strcpy(symVal[symInd].lex_name, lex);
  if (flag == 0) {
    symVal[symInd].size = dtypeSize(dtype);
    strcpy(symVal[symInd].dtype, dtype);
  } else {
    symVal[symInd].size = -1;
    strcpy(symVal[symInd].dtype, "function");
  }
  ++symInd;
}

void newLine() {
  ++row;
  col = 1;
}

struct token getNextToken(FILE * fin) {
  int c;
  struct token tkn = {
    .row = -1
  };
  int gotToken = 0;
  while (!gotToken && (c = fgetc(fin)) != EOF) {
    //special symbol
    if (charBelongsTo(c, specialsymbols)) {
      fillToken( & tkn, c, row, col, "Special Symbol");
      gotToken = 1;
      ++col;
    }
    //arithmetic operator
    else if (charBelongsTo(c, arithmeticsymbols)) {
      fillToken( & tkn, c, row, col, "ArithmeticOperator");
      gotToken = 1;
      ++col;
    }
    //check if + or ++
    else if (c == '+') {
      int d = fgetc(fin);
      if (d != '+') {
        fillToken( & tkn, c, row, col, "ArithmeticOperator");
        gotToken = 1;
        ++col;
        fseek(fin, -1, SEEK_CUR);
      } else {
        fillToken( & tkn, c, row, col, "UnaryOperator");
        strcpy(tkn.token_name, "++");
        gotToken = 1;
        col += 2;
      }
    }
    //check if - or --
    else if (c == '-') {
      int d = fgetc(fin);
      if (d != '-') {
        fillToken( & tkn, c, row, col, "ArithmeticOperator");
        gotToken = 1;
        ++col;
        fseek(fin, -1, SEEK_CUR);
      } else {
        fillToken( & tkn, c, row, col, "UnaryOperator");
        strcpy(tkn.token_name, "--");
        gotToken = 1;
        col += 2;
      }
    }
    //check if = or ==
    else if (c == '=') {
      int d = fgetc(fin);
      if (d != '=') {
        fillToken( & tkn, c, row, col, "AssignmentOperator");
        gotToken = 1;
        ++col;
        fseek(fin, -1, SEEK_CUR);
      } else {
        fillToken( & tkn, c, row, col, "RelationalOperator");
        strcpy(tkn.token_name, "==");
        gotToken = 1;
        col += 2;
      }
    }
    //check if number
    else if (isdigit(c)) {
      tkn.row = row;
      tkn.col = col++;
      tkn.token_name[0] = c;
      int k = 1;
      while ((c = fgetc(fin)) != EOF && isdigit(c)) {
        tkn.token_name[k++] = c;
        col++;
      }
      tkn.token_name[k] = '\0';
      strcpy(tkn.type, "Number");
    gotToken = 1;
    fseek(fin, -1, SEEK_CUR);
}
//to remove preprocessor directives
else if (c == '#') {
  while ((c = fgetc(fin)) != EOF && c != '\n')
  ;
  newLine();
} else if (c == '\n') {
  newLine();
  c = fgetc(fin);
  if (c == '#') {
    while ((c = fgetc(fin)) != EOF && c != '\n')
    ;
    newLine();
  } else if (c != EOF)
    fseek(fin, -1, SEEK_CUR);
} else if (isspace(c))
  ++col;
//check if identifier
else if (isalpha(c) || c == '_') {
  tkn.row = row;
  tkn.col = col++;
  tkn.token_name[0] = c;
  int k = 1;
  while ((c = fgetc(fin)) != EOF && isalnum(c)) {
    tkn.token_name[k++] = c;
    ++col;
  }
  tkn.token_name[k] = '\0';
  if (isKeyword(tkn.token_name)) {
    strcpy(tkn.type, "Keyword");
    strcpy(dtype, tkn.token_name); //changes from here
  } else {
    strcpy(tkn.type, "Identifier"); //changes from here
    // c = getc(fin);
    if (c == '(') {
      //function
      insertTable(tkn.token_name, dtype, 1);
    } else {
      //identifier
      insertTable(tkn.token_name, dtype, 0);
    }
    strcpy(tkn.token_name, "ID"); //changes till here
  }
  gotToken = 1;
  fseek(fin, -1, SEEK_CUR);
}
//check for comments or operator
else if (c == '/') {
  int d = fgetc(fin);
  ++col;
    if (d == '/') {
      while ((c = fgetc(fin)) != EOF && c != '\n')
        ++col;
      if (c == '\n')
        newLine();
    } else if (d == '*') {
      do {
        if (d == '\n')
          newLine();
        while ((c == fgetc(fin)) != EOF && c != '*') {
          ++col;
          if (c == '\n')
            newLine();
        }
        ++col;
      } while ((d == fgetc(fin)) != EOF && d != '/' && (++col));
      ++col;
    } else {
      fillToken( & tkn, c, row, --col, "ArithmeticOperator");
      gotToken = 1;
      fseek(fin, -1, SEEK_CUR);
    }
}
//string literal
else if (c == '"') {
  tkn.row = row;
  tkn.col = col;
  strcpy(tkn.type, "StringLiteral");
  int k = 1;
  tkn.token_name[0] = '"';
  while ((c = fgetc(fin)) != EOF && c != '"') {
    tkn.token_name[k++] = c;
    ++col;
  }
  tkn.token_name[k] = '"';
  gotToken = 1;
}
//RelOp or Logical OP
else if (c == '<' || c == '>' || c == '!') {
  fillToken( & tkn, c, row, col, "RelationalOperator");
  ++col;
  int d = fgetc(fin);
  if (d == '=') {
    ++col;
    strcat(tkn.token_name, "=");
  } else {
    if (c == '!')
      strcpy(tkn.type, "LogicalOperator");
    fseek(fin, -1, SEEK_CUR);
  }
  gotToken = 1;
} else if (c == '&' || c == '|') {
        int d = fgetc(fin);
        if (c == d) {
          tkn.token_name[0] = tkn.token_name[1] = c;
          tkn.token_name[2] = '\0';
          tkn.row = row;
          tkn.col = col;
          ++col;
          gotToken = 1;
          strcpy(tkn.type, "LogicalOperator");
        } else
          fseek(fin, -1, SEEK_CUR);
        ++col;
      } else
        ++col;
    }
    return tkn;
}
```