Compiler Design Lab 4

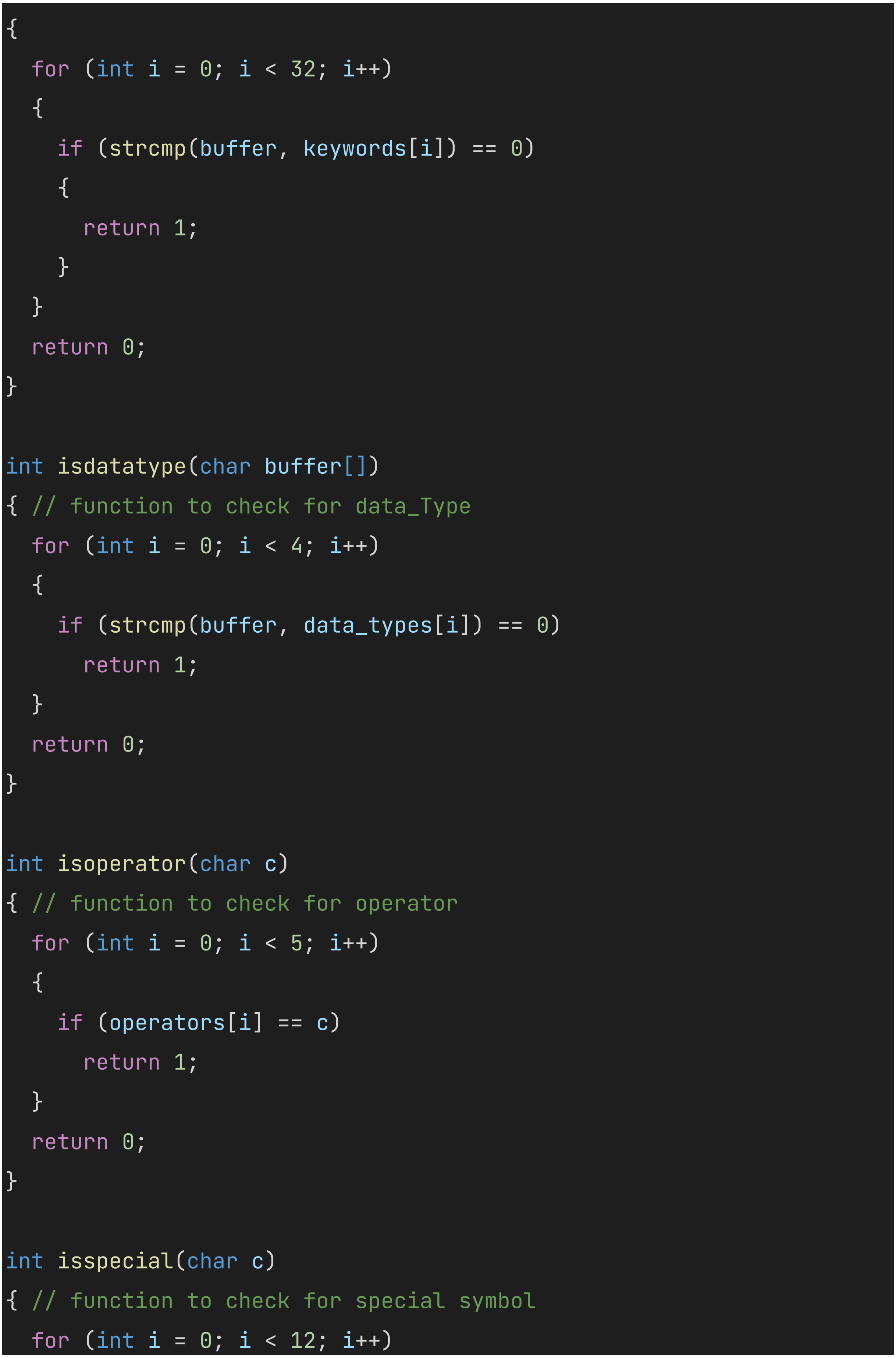
Dipesh Singh - 190905520

Question 1 : Using getNextToken( ) implemented in Lab No 3,design a Lexical Analyser to implement local and global symbol table to store tokens for identifiers using array of structure.

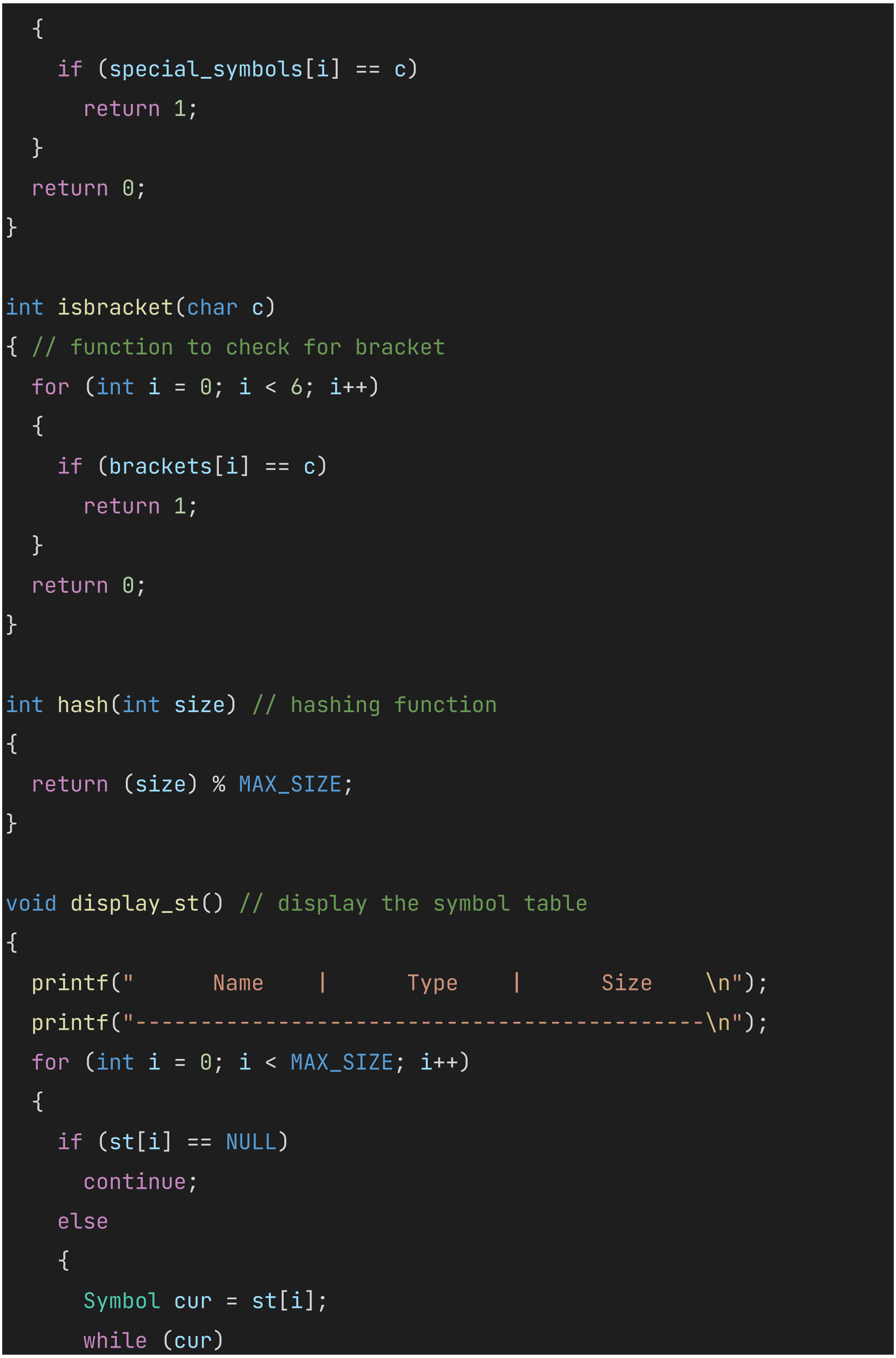
Source Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX_SIZE 20

int removeExcess()
{ // to remove spaces, tabs and comments
  FILE *fa, *fb;
  int ca, cb;
  fa = fopen("input.c", "r");
  if (fa == NULL)
  {
    printf("Cannot open file \n");
    exit(0);
  }
  fb = fopen("space_output.c", "w");
  ca = getc(fa);
  while (ca != EOF)
  {
    if (ca == ' ' || ca == '\t')
    {
      putc(' ', fb);
      while (ca == ' ' || ca == '\t')
        ca = getc(fa);
    }
    if (ca == '/')
    {
      cb = getc(fa);
      if (cb == '/')
      {
        while (ca != '\n')
          ca = getc(fa);
      }
      else if (cb == '*')
      {
        do
        {
          while (ca != '*')
            ca = getc(fa);
          ca = getc(fa);
        } while (ca != '/');
      }
      else
      {
        putc(ca, fb);
        putc(cb, fb);
      }
    }
    else
      putc(ca, fb);
    ca = getc(fa);
  }
  fclose(fa);
  fclose(fb);
  return 0;
}

int removePreprocess()
{ // to ignore preprocessor directives
  FILE *finp = fopen("space_output.c", "r");
  char c = 0;
  char buffer[100];
  buffer[0] = '\0';
  int i = 0;
  char *includeStr = "include", *defineStr = "define", *mainStr = "main";
  int mainFlag = 0, row = 1;
  while (c != EOF)
  {
    c = fgetc(finp);
    if (c == '#' && mainFlag == 0)
    {
      c = 'a';
      while (isalpha(c) != 0)
      {
        c = fgetc(finp);
        buffer[i++] = c;
      }
      buffer[i] = '\0';
      if (strstr(buffer, includeStr) != NULL || strstr(buffer, defineStr) != NULL)
      {
        row++;
        while (c != '\n')
        {
          c = fgetc(finp);
        }
      }
      else
      {
        for (int j = 0; j < i; j++)
          ;
        while (c != '\n')
        {
          c = fgetc(finp);
        }
      }
      i = 0;
      buffer[0] = '\0';
    }
    else
    {
      if (mainFlag == 0)
      {
        buffer[i++] = c;
        buffer[i] = '\0';
        if (strstr(buffer, mainStr) != NULL)
        {
          mainFlag = 1;
        }
      }
      if (c == ' ' || c == '\n')
      {
        buffer[0] = '\0';
        i = 0;
      }
    }
  }
  fclose(finp);
  return row;
}

char keywords[32][10] = {
    "auto",
    "double",
    "int",
    "struct",
    "break",
    "else",
```

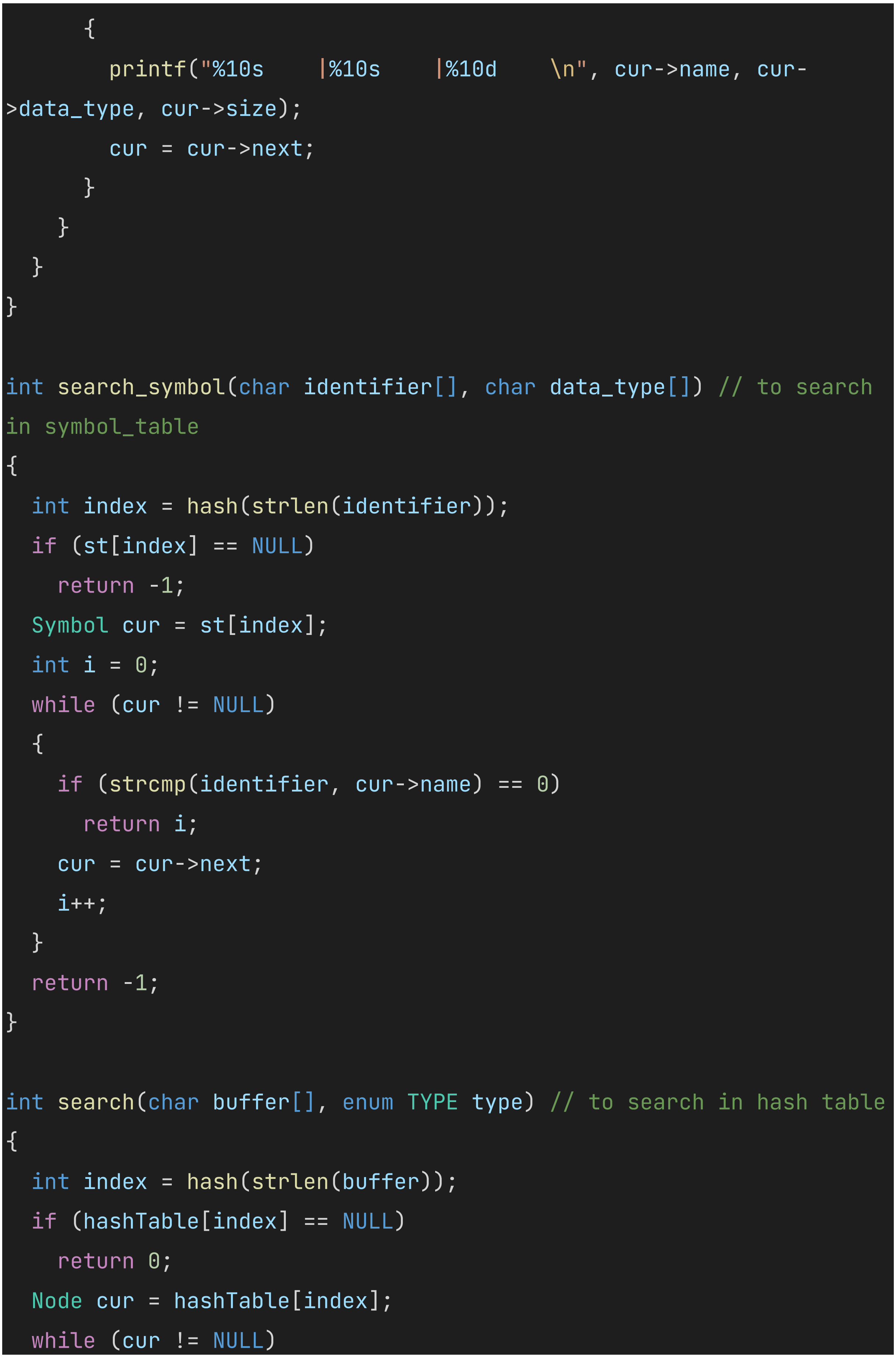
```
    "long",
    "switch",
    "case",
    "enum",
    "register",
    "typedef",
    "char",
    "extern",
    "return",
    "union",
    "const",
    "float",
    "short",
    "unsigned",
    "continue",
    "for",
    "signed",
    "void",
    "default",
    "goto",
    "sizeof",
    "voltile",
    "do",
    "if",
    "static",
    "while"};            // list of keywords
char data_types[][10] = { // list of data types
    "double",
    "int",
    "char",
    "float"};
char operators[5] = { // list of operators
    '+',
    '-',
    '/',
    '%',
    '*'};
char brackets[6] = { // list of brackets
    '(',
    ')',
    '[',
    ']',
    '{',
    '}'};
char special_symbols[12] = { // list of special symbols
    '*',
    ';',
    ':',
    '.',
    ',',
    '^',
    '&',
    '!',
    '>',
    '<',
    '~',
    '`'};

enum TYPE // lexeme type enumerator
{
  IDENTIFIER,
  KEYWORD,
  STRING_LITERAL,
  NUMERIC_CONSTANT,
  OPERATOR,
  BRACKET,
  SPECIAL_SYMBOL,
  RELATIONAL_OPERATOR,
  CHARACTER_CONSTANT
};

char types[][30] = { // map for type to string
    "IDENTIFIER",
    "KEYWORD",
    "STRING_LITERAL",
    "NUMERIC_CONSTANT",
    "OPERATOR",
    "BRACKET",
    "SPECIAL_SYMBOL",
    "RELATIONAL_OPERATOR",
    "CHARACTER_CONSTANT"};

typedef struct node
{
  char *cur;
  int row, col;
  struct node *next;
  enum TYPE type;
} * Node; // element for hash table

typedef struct symbol
{
  char *name;
  char *data_type;
  struct symbol *next;
  unsigned int size;
} * Symbol; // element for symbol table

Node hashTable[MAX_SIZE]; // hash table
Symbol st[MAX_SIZE];      // symbol table

int iskeyword(char buffer[]) // function to check for keyword
```



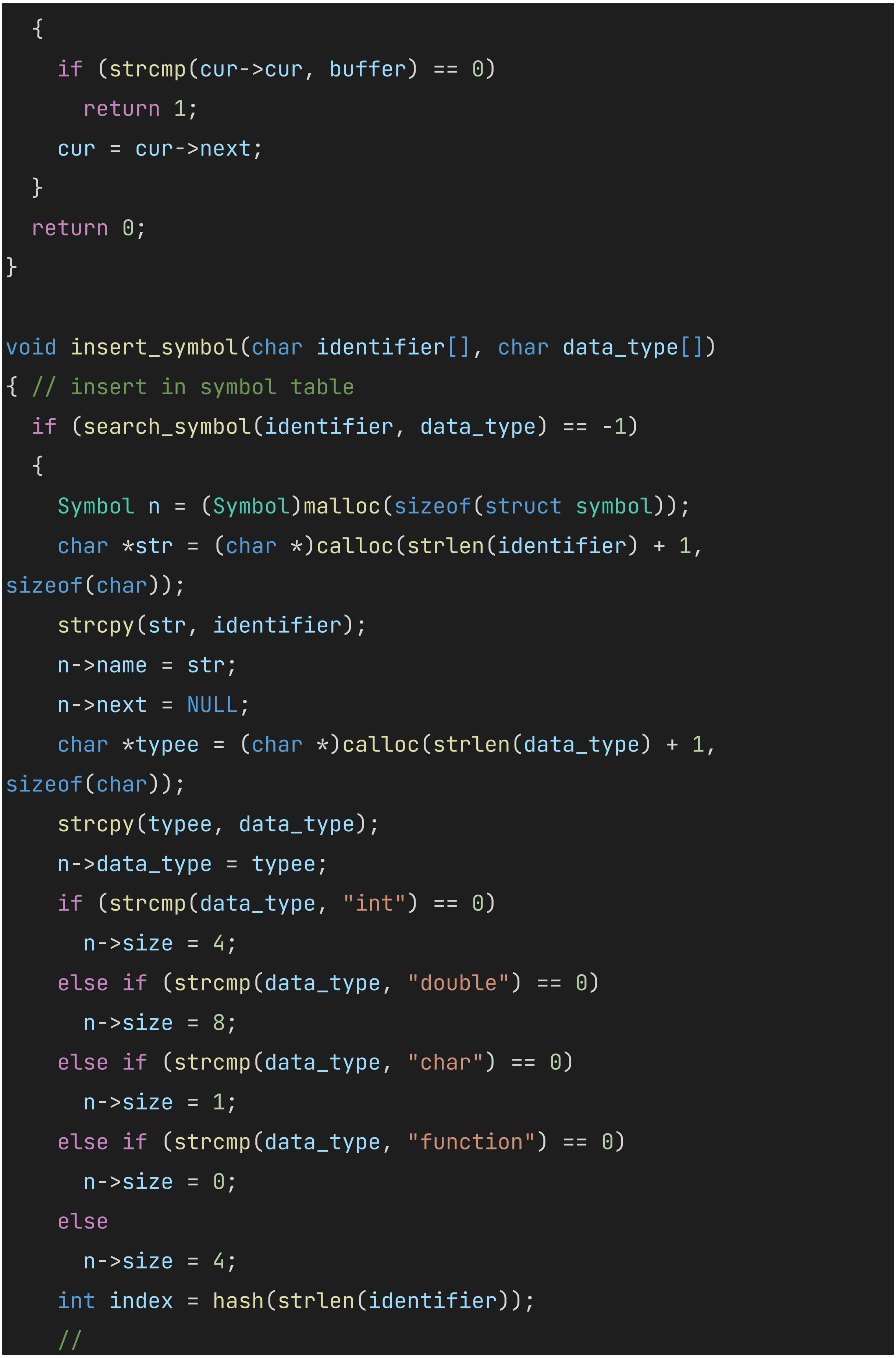
```
{
  for (int i = 0; i < 32; i++)
  {
    if (strcmp(buffer, keywords[i]) == 0)
    {
      return 1;
    }
  }
  return 0;
}

int isdatatype(char buffer[])
{ // function to check for data_Type
  for (int i = 0; i < 4; i++)
  {
    if (strcmp(buffer, data_types[i]) == 0)
      return 1;
  }
  return 0;
}

int isoperator(char c)
{ // function to check for operator
  for (int i = 0; i < 5; i++)
  {
    if (operators[i] == c)
      return 1;
  }
  return 0;
}

int isspecial(char c)
{ // function to check for special symbol
  for (int i = 0; i < 12; i++)
```



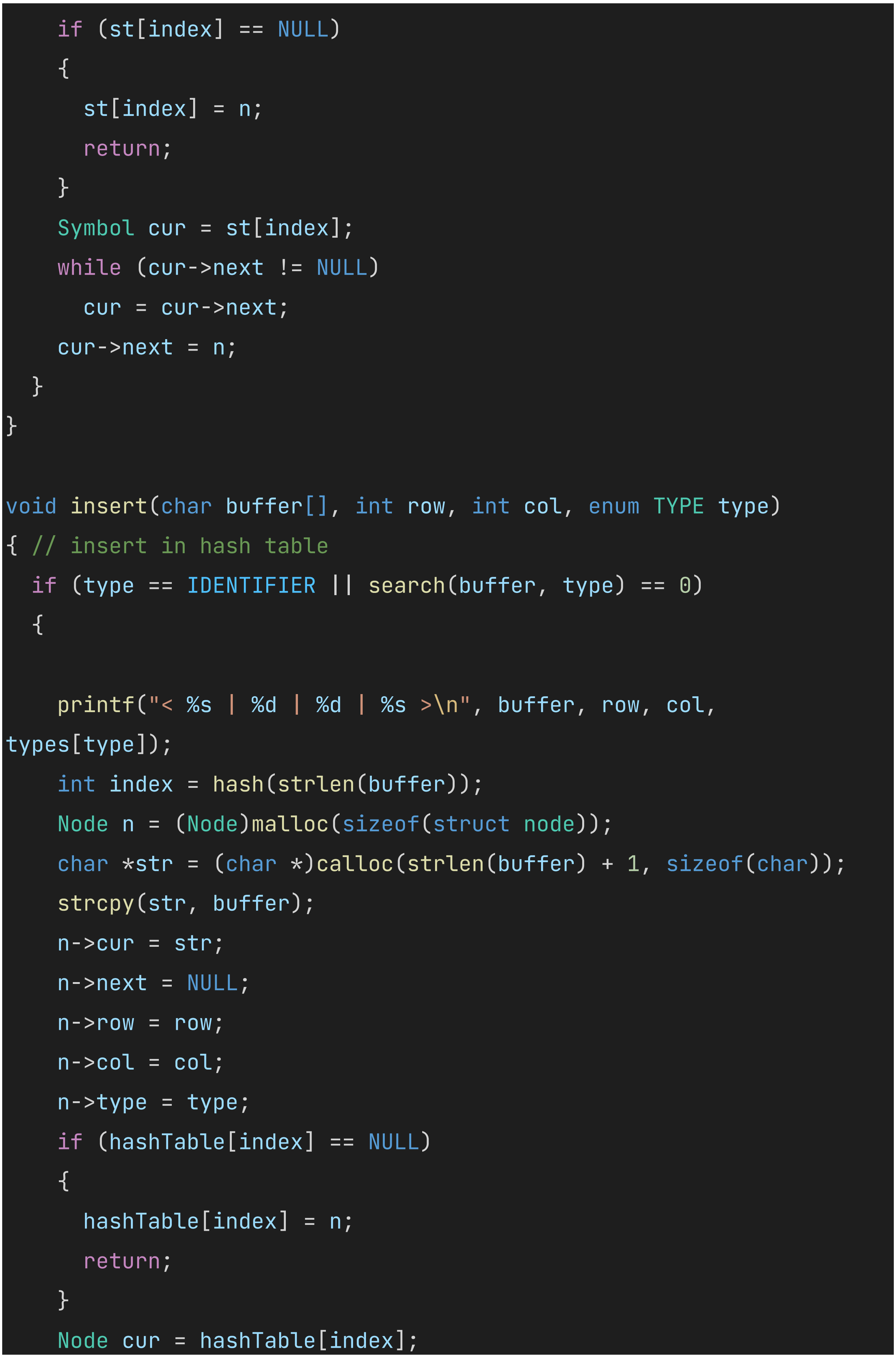
```c
  {
    if (special_symbols[i] == c)
      return 1;
  }
  return 0;
}

int isbracket(char c)
{ // function to check for bracket
  for (int i = 0; i < 6; i++)
  {
    if (brackets[i] == c)
      return 1;
  }
  return 0;
}

int hash(int size) // hashing function
{
  return (size) % MAX_SIZE;
}

void display_st() // display the symbol table
{
  printf("      Name    |      Type    |      Size    \n");
  printf("--------------------------------------------\n");
  for (int i = 0; i < MAX_SIZE; i++)
  {
    if (st[i] == NULL)
      continue;
    else
    {
      Symbol cur = st[i];
      while (cur)
```



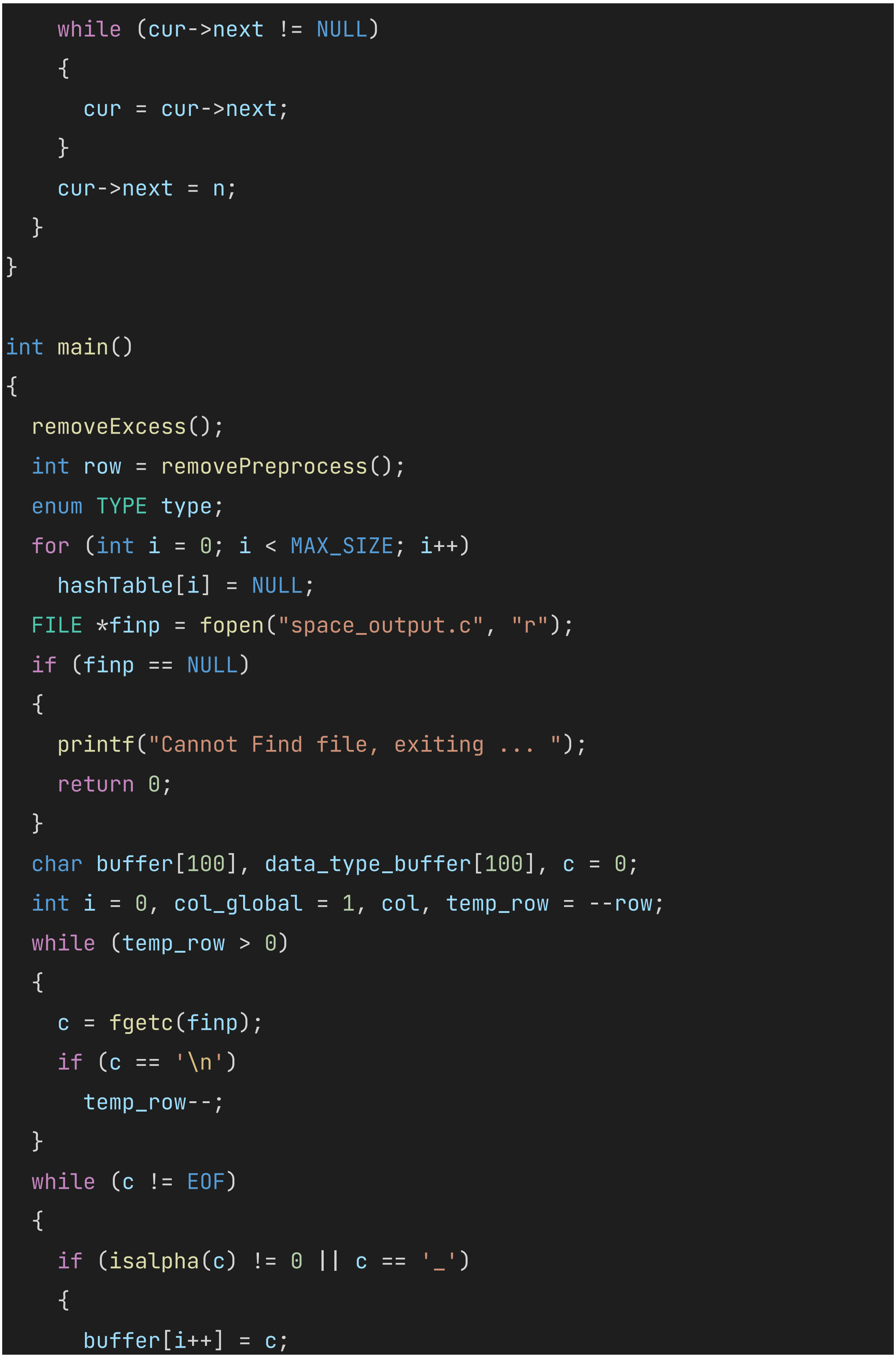
```c
      {
        printf("%10s    |%10s    |%10d    \n", cur->name, cur->data_type, cur->size);
        cur = cur->next;
      }
    }
  }
}

int search_symbol(char identifier[], char data_type[]) // to search in symbol_table
{
  int index = hash(strlen(identifier));
  if (st[index] == NULL)
    return -1;
  Symbol cur = st[index];
  int i = 0;
  while (cur != NULL)
  {
    if (strcmp(identifier, cur->name) == 0)
      return i;
    cur = cur->next;
    i++;
  }
  return -1;
}

int search(char buffer[], enum TYPE type) // to search in hash table
{
  int index = hash(strlen(buffer));
  if (hashTable[index] == NULL)
    return 0;
  Node cur = hashTable[index];
  while (cur != NULL)
```



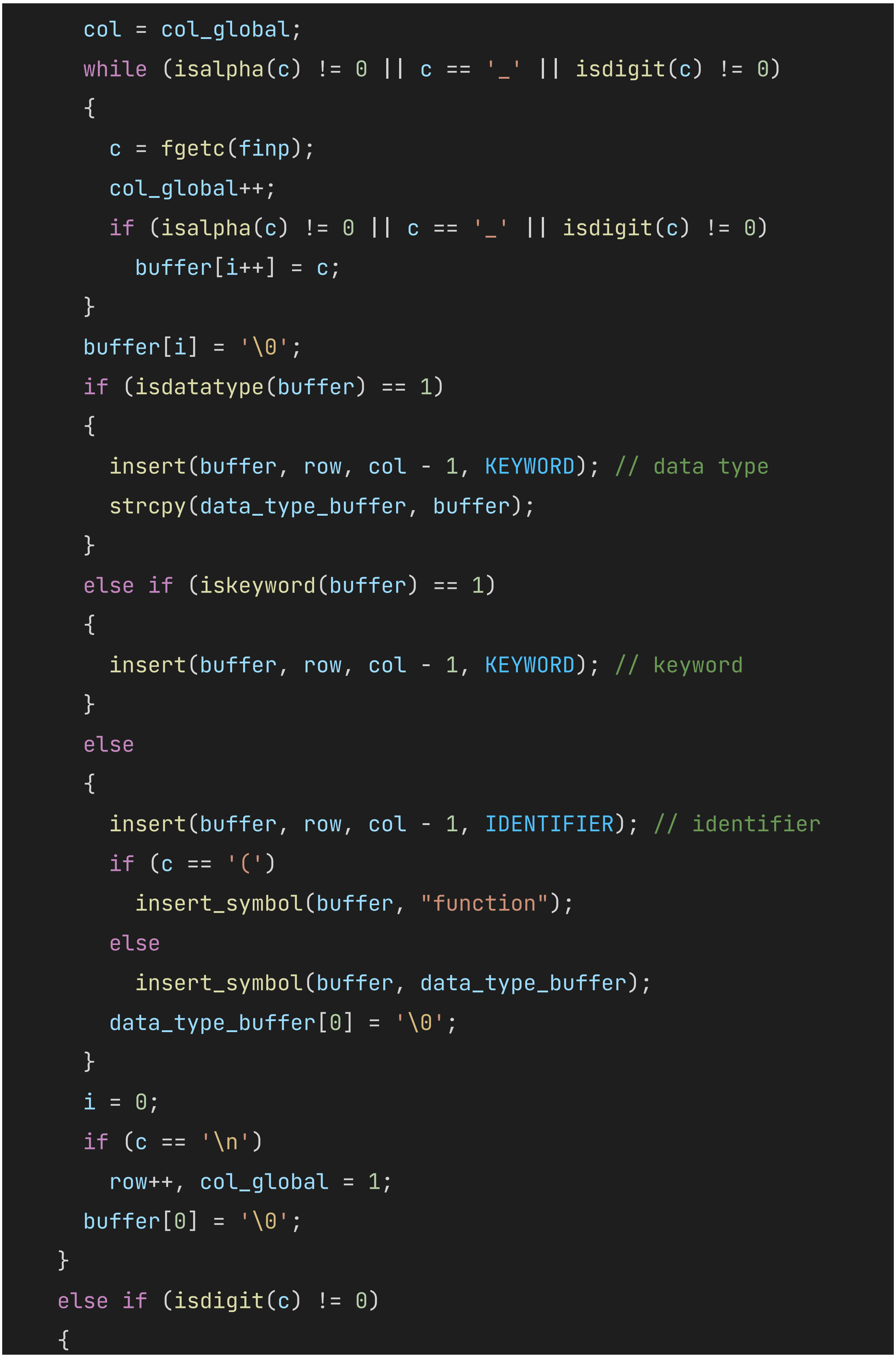
```
  {
    if (strcmp(cur->cur, buffer) == 0)
      return 1;
    cur = cur->next;
  }
  return 0;
}

void insert_symbol(char identifier[], char data_type[])
{ // insert in symbol table
  if (search_symbol(identifier, data_type) == -1)
  {
    Symbol n = (Symbol)malloc(sizeof(struct symbol));
    char *str = (char *)calloc(strlen(identifier) + 1,
sizeof(char));
    strcpy(str, identifier);
    n->name = str;
    n->next = NULL;
    char *typee = (char *)calloc(strlen(data_type) + 1,
sizeof(char));
    strcpy(typee, data_type);
    n->data_type = typee;
    if (strcmp(data_type, "int") == 0)
      n->size = 4;
    else if (strcmp(data_type, "double") == 0)
      n->size = 8;
    else if (strcmp(data_type, "char") == 0)
      n->size = 1;
    else if (strcmp(data_type, "function") == 0)
      n->size = 0;
    else
      n->size = 4;
    int index = hash(strlen(identifier));
    //
```



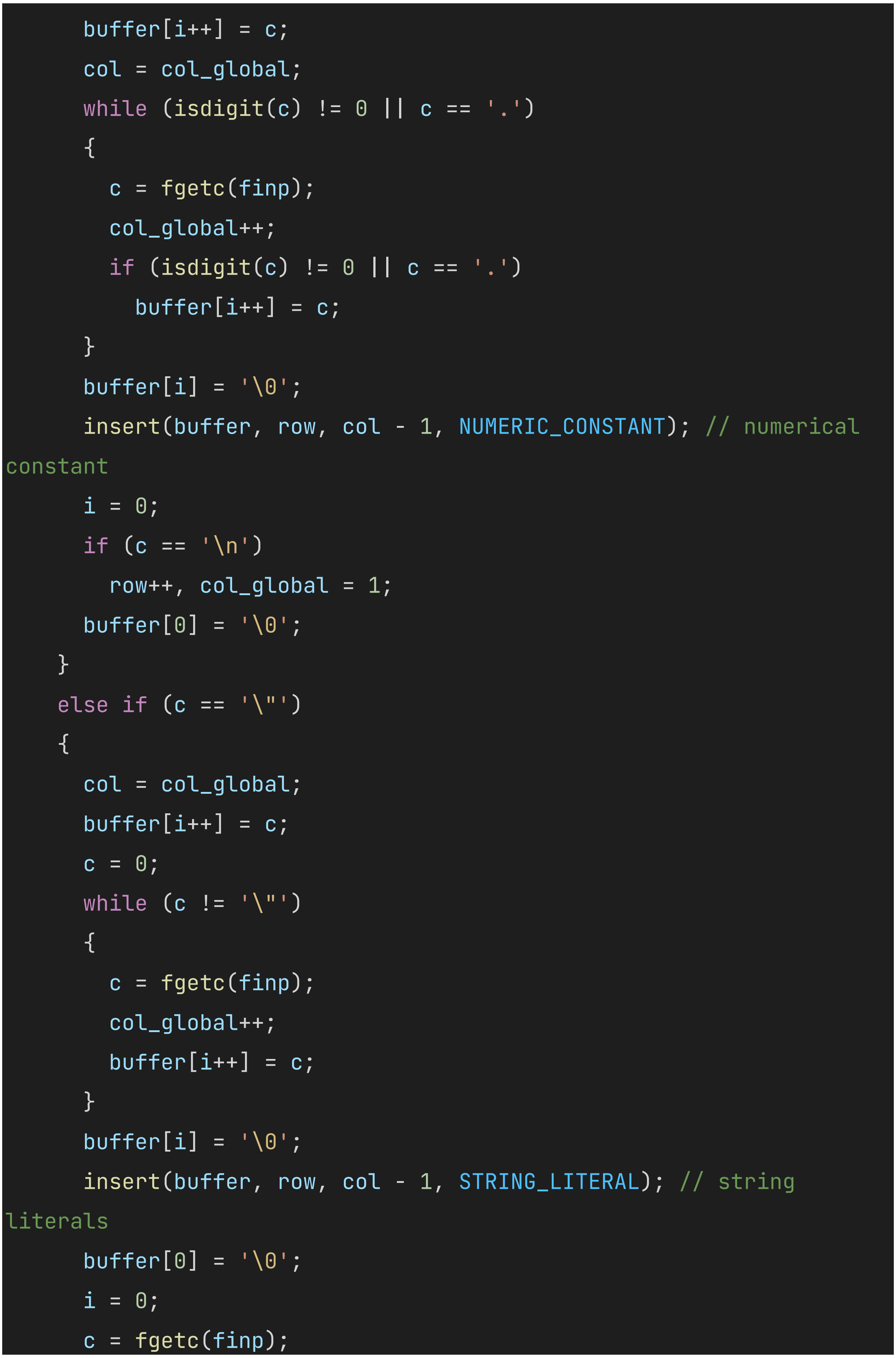
```
    if (st[index] == NULL)
    {
      st[index] = n;
      return;
    }
    Symbol cur = st[index];
    while (cur->next != NULL)
      cur = cur->next;
    cur->next = n;
  }
}

void insert(char buffer[], int row, int col, enum TYPE type)
{ // insert in hash table
  if (type == IDENTIFIER || search(buffer, type) == 0)
  {

    printf("< %s | %d | %d | %s >\n", buffer, row, col,
types[type]);
    int index = hash(strlen(buffer));
    Node n = (Node)malloc(sizeof(struct node));
    char *str = (char *)calloc(strlen(buffer) + 1, sizeof(char));
    strcpy(str, buffer);
    n->cur = str;
    n->next = NULL;
    n->row = row;
    n->col = col;
    n->type = type;
    if (hashTable[index] == NULL)
    {
      hashTable[index] = n;
      return;
    }
    Node cur = hashTable[index];
```



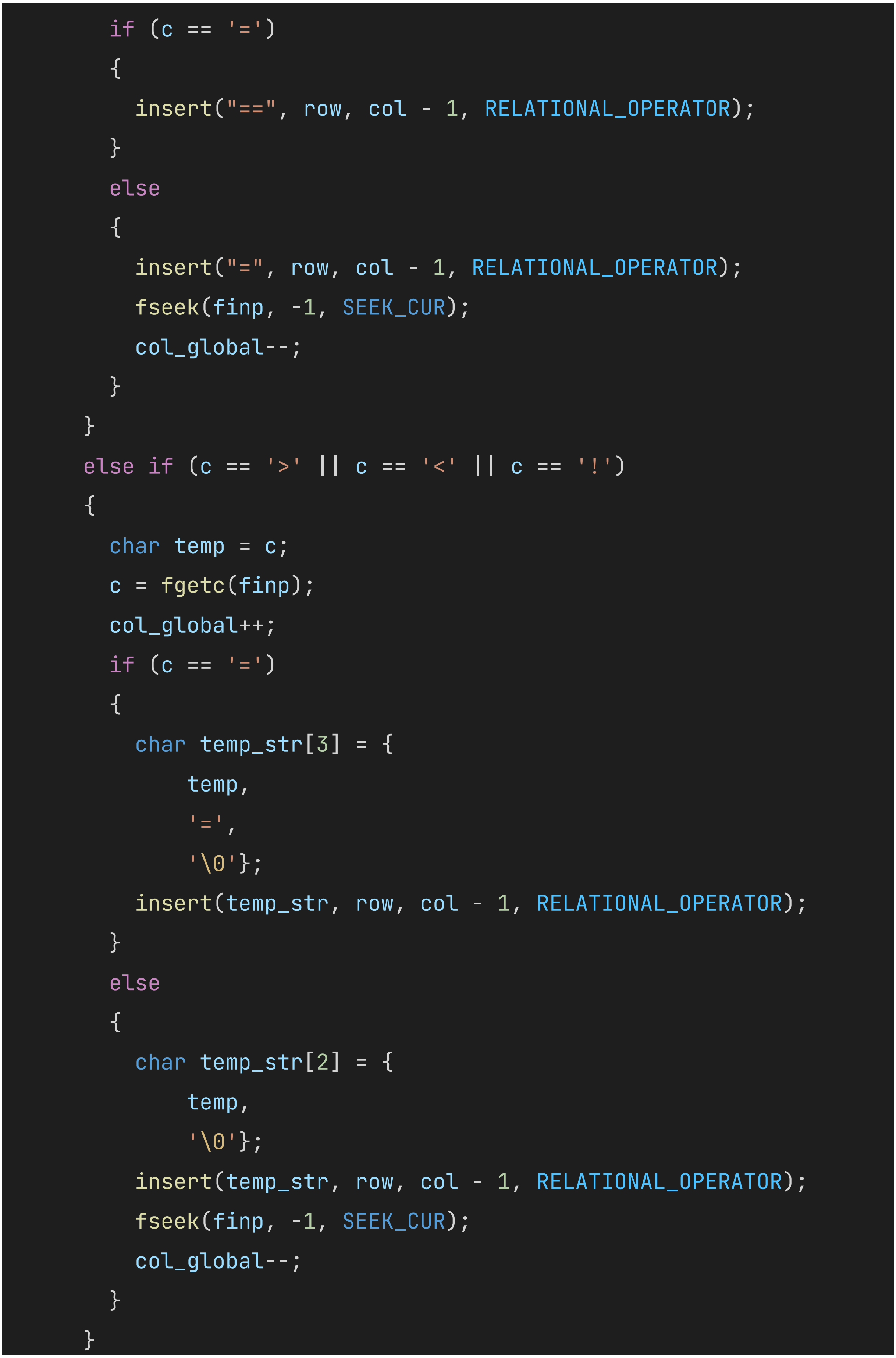
```c
    while (cur->next != NULL)
    {
      cur = cur->next;
    }
    cur->next = n;
  }
}

int main()
{
  removeExcess();
  int row = removePreprocess();
  enum TYPE type;
  for (int i = 0; i < MAX_SIZE; i++)
    hashTable[i] = NULL;
  FILE *finp = fopen("space_output.c", "r");
  if (finp == NULL)
  {
    printf("Cannot Find file, exiting ... ");
    return 0;
  }
  char buffer[100], data_type_buffer[100], c = 0;
  int i = 0, col_global = 1, col, temp_row = --row;
  while (temp_row > 0)
  {
    c = fgetc(finp);
    if (c == '\n')
      temp_row--;
  }
  while (c != EOF)
  {
    if (isalpha(c) != 0 || c == '_')
    {
      buffer[i++] = c;
```



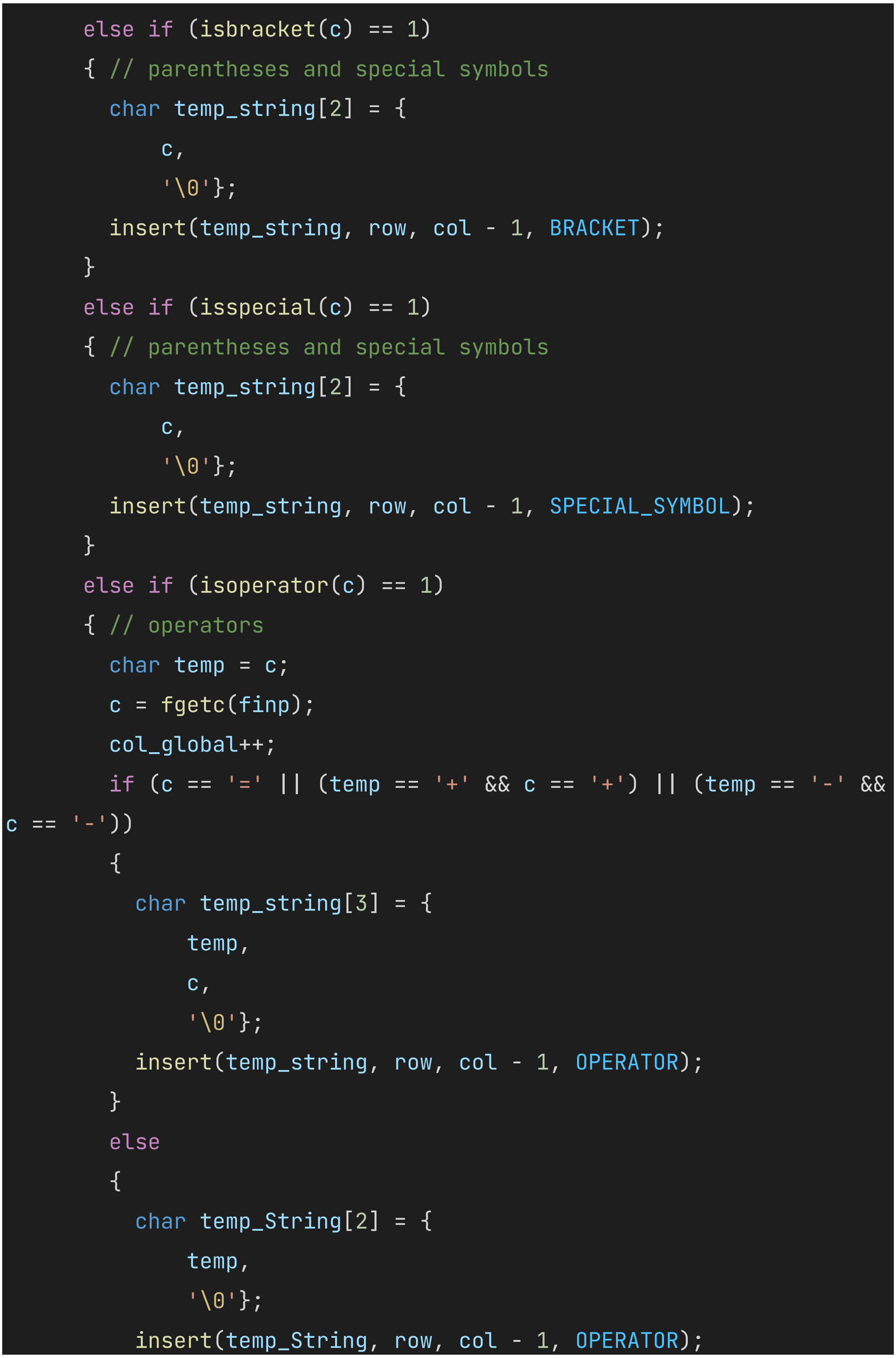
```
      col = col_global;
      while (isalpha(c) != 0 || c == '_' || isdigit(c) != 0)
      {
        c = fgetc(finp);
        col_global++;
        if (isalpha(c) != 0 || c == '_' || isdigit(c) != 0)
          buffer[i++] = c;
      }
      buffer[i] = '\0';
      if (isdatatype(buffer) == 1)
      {
        insert(buffer, row, col - 1, KEYWORD); // data type
        strcpy(data_type_buffer, buffer);
      }
      else if (iskeyword(buffer) == 1)
      {
        insert(buffer, row, col - 1, KEYWORD); // keyword
      }
      else
      {
        insert(buffer, row, col - 1, IDENTIFIER); // identifier
        if (c == '(')
          insert_symbol(buffer, "function");
        else
          insert_symbol(buffer, data_type_buffer);
        data_type_buffer[0] = '\0';
      }
      i = 0;
      if (c == '\n')
        row++, col_global = 1;
      buffer[0] = '\0';
    }
    else if (isdigit(c) != 0)
    {
```



```
      buffer[i++] = c;
      col = col_global;
      while (isdigit(c) != 0 || c == '.')
      {
        c = fgetc(finp);
        col_global++;
        if (isdigit(c) != 0 || c == '.')
          buffer[i++] = c;
      }
      buffer[i] = '\0';
      insert(buffer, row, col - 1, NUMERIC_CONSTANT); // numerical
constant
      i = 0;
      if (c == '\n')
        row++, col_global = 1;
      buffer[0] = '\0';
    }
    else if (c == '\"')
    {
      col = col_global;
      buffer[i++] = c;
      c = 0;
      while (c != '\"')
      {
        c = fgetc(finp);
        col_global++;
        buffer[i++] = c;
      }
      buffer[i] = '\0';
      insert(buffer, row, col - 1, STRING_LITERAL); // string
literals
      buffer[0] = '\0';
      i = 0;
      c = fgetc(finp);
      col_global++;
    }
    else if (c == '\'')
    {
      col = col_global;
      buffer[i++] = c;
      c = 0;
      c = fgetc(finp);
      col_global++;
      buffer[i++] = c;
      if (c == '\\')
      {
        c = fgetc(finp);
        col_global++;
        buffer[i++] = c;
      }
      c = fgetc(finp);
      col_global++;
      buffer[i++] = c;
      buffer[i] = '\0';
      insert(buffer, row, col - 1, CHARACTER_CONSTANT); // character constants
      buffer[0] = '\0';
      i = 0;
      c = fgetc(finp);
      col_global++;
    }
    else
    {
      col = col_global;
      if (c == '=')
      { // relational and logical operators
        c = fgetc(finp);
        col_global++;
```



```
  if (c == '=')
  {
    insert("==", row, col - 1, RELATIONAL_OPERATOR);
  }
  else
  {
    insert("=", row, col - 1, RELATIONAL_OPERATOR);
    fseek(finp, -1, SEEK_CUR);
    col_global--;
  }
}
else if (c == '>' || c == '<' || c == '!')
{
  char temp = c;
  c = fgetc(finp);
  col_global++;
  if (c == '=')
  {
    char temp_str[3] = {
        temp,
        '=',
        '\0'};
    insert(temp_str, row, col - 1, RELATIONAL_OPERATOR);
  }
  else
  {
    char temp_str[2] = {
        temp,
        '\0'};
    insert(temp_str, row, col - 1, RELATIONAL_OPERATOR);
    fseek(finp, -1, SEEK_CUR);
    col_global--;
  }
}
```

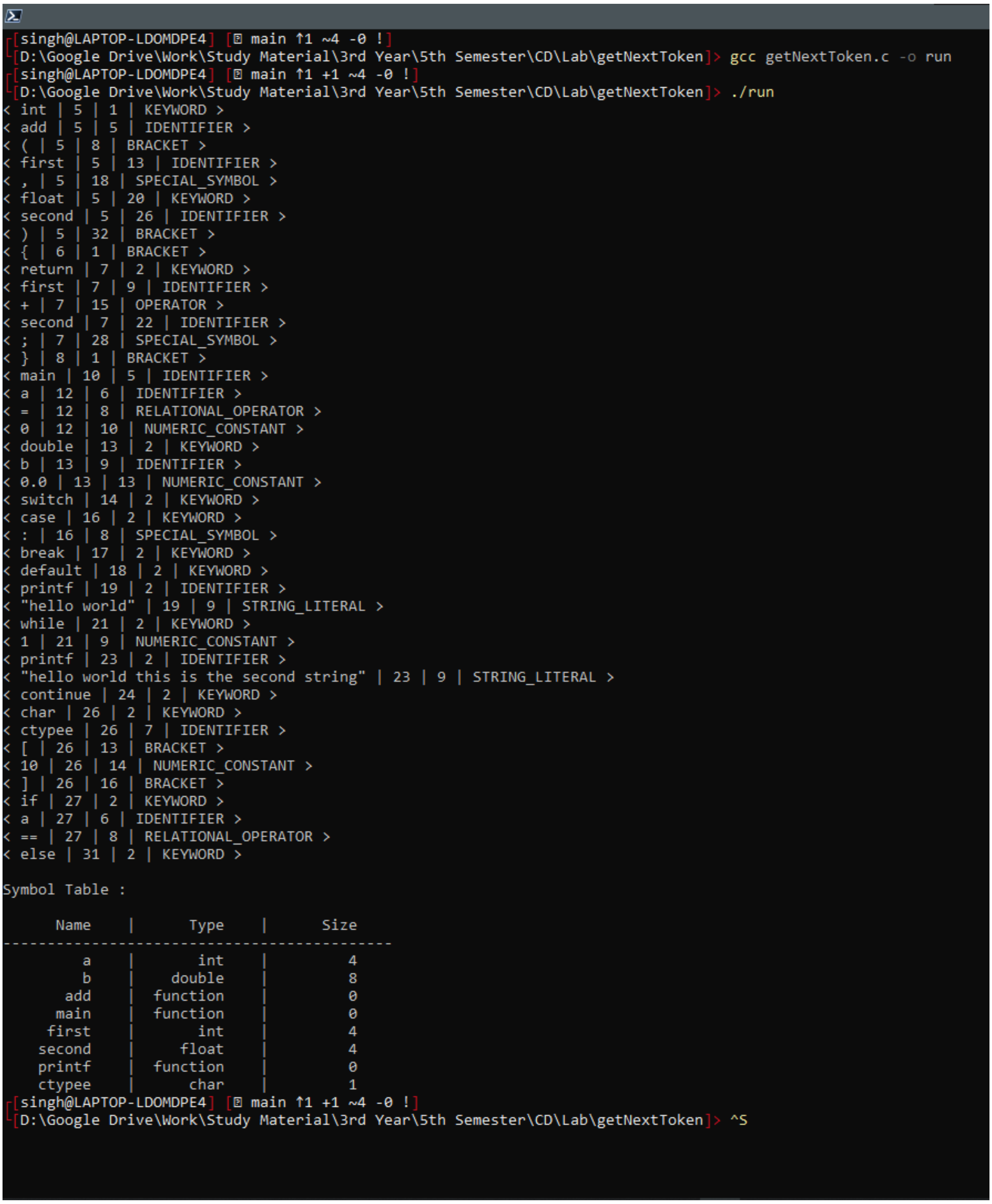


```
      else if (isbracket(c) == 1)
      { // parentheses and special symbols
        char temp_string[2] = {
            c,
            '\0'};
        insert(temp_string, row, col - 1, BRACKET);
      }
      else if (isspecial(c) == 1)
      { // parentheses and special symbols
        char temp_string[2] = {
            c,
            '\0'};
        insert(temp_string, row, col - 1, SPECIAL_SYMBOL);
      }
      else if (isoperator(c) == 1)
      { // operators
        char temp = c;
        c = fgetc(finp);
        col_global++;
        if (c == '=' || (temp == '+' && c == '+') || (temp == '-' &&
c == '-'))
        {
          char temp_string[3] = {
              temp,
              c,
              '\0'};
          insert(temp_string, row, col - 1, OPERATOR);
        }
        else
        {
          char temp_String[2] = {
              temp,
              '\0'};
          insert(temp_String, row, col - 1, OPERATOR);
          fseek(finp, -1, SEEK_CUR);
          col_global--;
        }
      }
      else if (c == '\n') // new line
        row++, col_global = 1;
      c = fgetc(finp);
      col_global++;
    }
  }
  printf("\nSymbol Table : \n\n");
  display_st();
  return 0;
}
```

input.c :

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

int add(int first, float second)
{
 return first + (int)second;
}

int main()
{
 int a = 0;
 double b = 0.0;
 switch (0)
 {
 case 0:
  break;
 default:
  printf("hello world");
 }
 while (1)
 {
  printf("hello world this is the second string");
  continue;
 }
 char ctypee[10];
 if (a == 1)
 {
  return 0;
 }
 else
  return 1;
 return 0;
}
```

Output :



```
┌[singh@LAPTOP-LDOMDPE4] [⎇ main ↑1 ~4 -0 !]
└[D:\Google Drive\Work\Study Material\3rd Year\5th Semester\CD\Lab\getNextToken]> gcc getNextToken.c -o run
┌[singh@LAPTOP-LDOMDPE4] [⎇ main ↑1 +1 ~4 -0 !]
└[D:\Google Drive\Work\Study Material\3rd Year\5th Semester\CD\Lab\getNextToken]> ./run
< int | 5 | 1 | KEYWORD >
< add | 5 | 5 | IDENTIFIER >
< ( | 5 | 8 | BRACKET >
< first | 5 | 13 | IDENTIFIER >
< , | 5 | 18 | SPECIAL_SYMBOL >
< float | 5 | 20 | KEYWORD >
< second | 5 | 26 | IDENTIFIER >
< ) | 5 | 32 | BRACKET >
< { | 6 | 1 | BRACKET >
< return | 7 | 2 | KEYWORD >
< first | 7 | 9 | IDENTIFIER >
< + | 7 | 15 | OPERATOR >
< second | 7 | 22 | IDENTIFIER >
< ; | 7 | 28 | SPECIAL_SYMBOL >
< } | 8 | 1 | BRACKET >
< main | 10 | 5 | IDENTIFIER >
< a | 12 | 6 | IDENTIFIER >
< = | 12 | 8 | RELATIONAL_OPERATOR >
< 0 | 12 | 10 | NUMERIC_CONSTANT >
< double | 13 | 2 | KEYWORD >
< b | 13 | 9 | IDENTIFIER >
< 0.0 | 13 | 13 | NUMERIC_CONSTANT >
< switch | 14 | 2 | KEYWORD >
< case | 16 | 2 | KEYWORD >
< : | 16 | 8 | SPECIAL_SYMBOL >
< break | 17 | 2 | KEYWORD >
< default | 18 | 2 | KEYWORD >
< printf | 19 | 2 | IDENTIFIER >
< "hello world" | 19 | 9 | STRING_LITERAL >
< while | 21 | 2 | KEYWORD >
< 1 | 21 | 9 | NUMERIC_CONSTANT >
< printf | 23 | 2 | IDENTIFIER >
< "hello world this is the second string" | 23 | 9 | STRING_LITERAL >
< continue | 24 | 2 | KEYWORD >
< char | 26 | 2 | KEYWORD >
< ctypee | 26 | 7 | IDENTIFIER >
< [ | 26 | 13 | BRACKET >
< 10 | 26 | 14 | NUMERIC_CONSTANT >
< ] | 26 | 16 | BRACKET >
< if | 27 | 2 | KEYWORD >
< a | 27 | 6 | IDENTIFIER >
< == | 27 | 8 | RELATIONAL_OPERATOR >
< else | 31 | 2 | KEYWORD >

Symbol Table :

      Name    |      Type     |      Size
--------------------------------------------
         a    |        int    |        4
         b    |     double    |        8
       add    |   function    |        0
      main    |   function    |        0
     first    |        int    |        4
    second    |      float    |        4
    printf    |   function    |        0
    ctypee    |       char    |        1
┌[singh@LAPTOP-LDOMDPE4] [⎇ main ↑1 +1 ~4 -0 !]
└[D:\Google Drive\Work\Study Material\3rd Year\5th Semester\CD\Lab\getNextToken]> ^S
```