LAB 3

Juhi Mehta
190905412
Roll No: 55
Batch B3

Solved example: Identify arithmetic and relational operators from given file

Code:

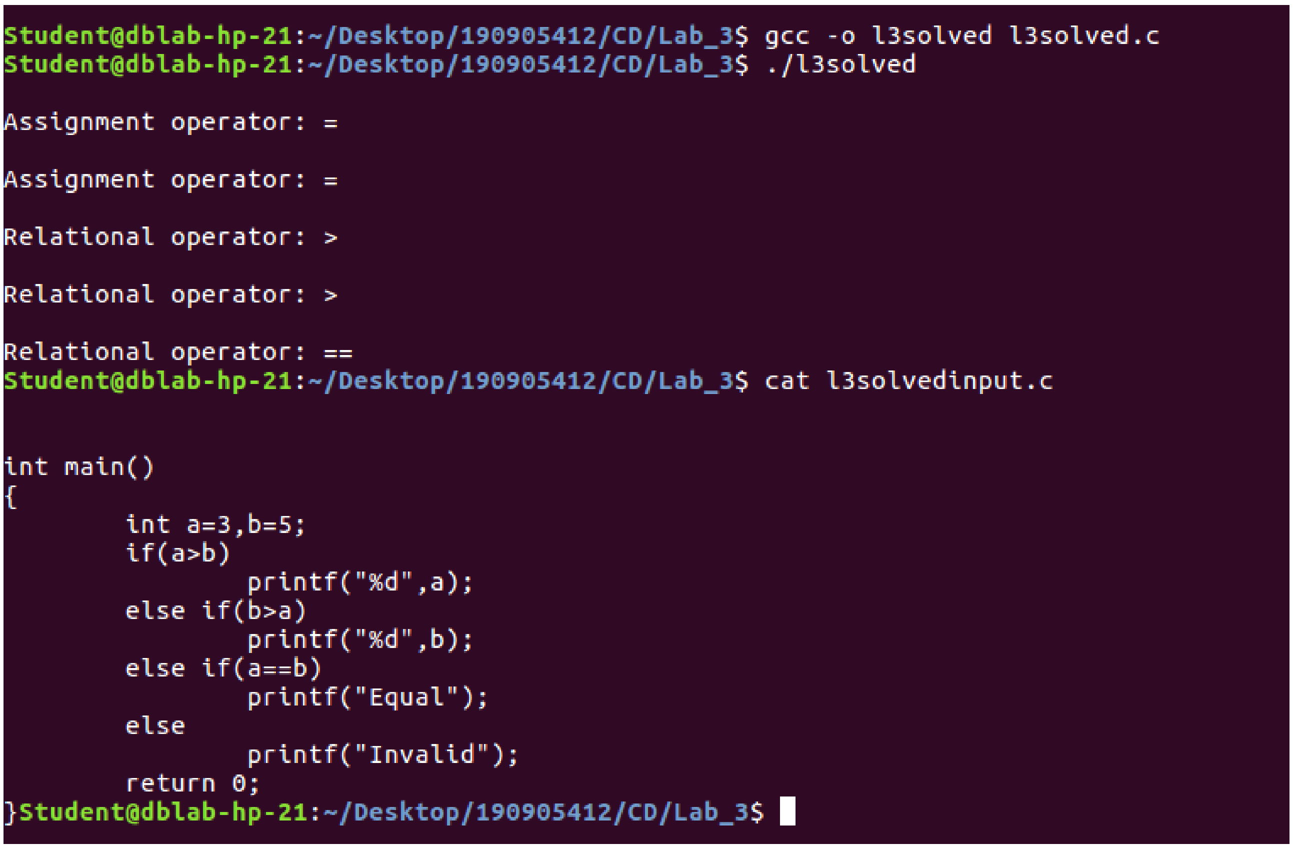
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
	char c,buff[10];
	FILE *fp = fopen("l3solvedinput.c","r");
	c = fgetc(fp);
	if(fp == NULL)
	{
		printf("Cannot open input file");
		exit(0);
	}
	int i;
	while(c != EOF)
	{
		i=0;
		buff[0]='\0';

		if(c=='=')
		{
			buff[i++]=c;
			c=fgetc(fp);
			if(c=='=')
			{
				buff[i++]=c;
				buff[i]='\0';
				printf("\nRelational operator: %s\n",buff);
			}
			else
			{
				buff[i]='\0';
				printf("\nAssignment operator: %s\n",buff);
			}
		}
		else
		{
			if(c=='>'||c=='<'||c=='!')
			{
```

```
                        buff[i++]=c;
                        c=fgetc(fp);
                        if(c=='=')
                        {
                                buff[i++]=c;
                        }
                        buff[i]='\0';
                        printf("\nRelational operator: %s\n",buff);
                }
                else
                        buff[i]='\0';
        }
        c=      fgetc(fp);
    }
}
```

Output:



```
Student@dblab-hp-21:~/Desktop/190905412/CD/Lab_3$ gcc -o l3solved l3solved.c
Student@dblab-hp-21:~/Desktop/190905412/CD/Lab_3$ ./l3solved

Assignment operator: =

Assignment operator: =

Relational operator: >

Relational operator: >

Relational operator: ==
Student@dblab-hp-21:~/Desktop/190905412/CD/Lab_3$ cat l3solvedinput.c


int main()
{
        int a=3,b=5;
        if(a>b)
                printf("%d",a);
        else if(b>a)
                printf("%d",b);
        else if(a==b)
                printf("Equal");
        else
                printf("Invalid");
        return 0;
}Student@dblab-hp-21:~/Desktop/190905412/CD/Lab_3$ 
```

1) Design a lexical analyzer which contains getNextToken( ) for a simple C program to create a structure of token each time and return, which includes row number, column number and token type. The tokens to be identified are arithmetic operators, relational operators, logical operators, special symbols, keywords, numerical constants, string literals and identifiers. Also, getNextToken() should ignore all the tokens when encountered inside single line

or multiline comment or inside string literal. Preprocessor directive should also be stripped.

Code:

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

struct token
{
   char lexeme[64];
   unsigned int row, col;
};

static int row = 1, col = 1;
char buf[2048];

char specialsymbols[] = {'?', ';', ':', ',', '.'};

const char *keywords[] = {"int", "return", "for", "while", "if", "else","printf", "case", "break",
"float", "const", "bool"};

char arithmeticsymbols[] = {'*', '%'};

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

int charBelongsTo(int c, char *arr)
{
   int len = 0;
   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;
}
```

```
void fillToken(struct token *t, char c, int row, int col)
{
   t->lexeme[0] = c;
   t->lexeme[1] = '\0';
   t->row = row;
   t->col = col;
}

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

struct token getNextToken(FILE *f)
{
   int c;
   struct token tkn =
      {
         .row = -1};

   int gotToken = 0;
   while (!gotToken && (c = fgetc(f)) != EOF)
   {
      if (charBelongsTo(c, specialsymbols))
      {
         fillToken(&tkn, c, row, col);
         gotToken = 1;
         ++col;
      }
      else if (charBelongsTo(c, arithmeticsymbols))
      {
         fillToken(&tkn, c, row, col);
         gotToken = 1;
         ++col;
      }
      else if (c == '(')
      {
         fillToken(&tkn, c, row, col);
         gotToken = 1;
         ++col;
      }
      else if (c == ')')
      {
         fillToken(&tkn, c, row, col);
         gotToken = 1;
         ++col;
      }
      else if (c == '{')
      {
         fillToken(&tkn, c, row, col);
```

```
    gotToken = 1;
    ++col;
}
else if (c == '}')
{
    fillToken(&tkn, c, row, col);
    gotToken = 1;
    ++col;
}
else if (c == '[')
{
    fillToken(&tkn, c, row, col);
    gotToken = 1;
    ++col;
}
else if (c == ']')
{
    fillToken(&tkn, c, row, col);
    gotToken = 1;
    ++col;
}
else if (c == '+')
{
    int d = fgetc(f);
    if (d != '+')
    {
        fillToken(&tkn, c, row, col);
        gotToken = 1;
        ++col;
        fseek(f, -1, SEEK_CUR); //go back 1 step *
    }
    else
    {
        fillToken(&tkn, c, row, col);
        strcpy(tkn.lexeme, "++");
        gotToken = 1;
        col += 2; //skip next as it is already included
    }
}
else if (c == '-')
{
    int d = fgetc(f);
    if (d != '-')
    {
        fillToken(&tkn, c, row, col);
        gotToken = 1;
        ++col;
        fseek(f, -1, SEEK_CUR); //go back 1 step *
    }
    else
    {
        fillToken(&tkn, c, row, col);
        strcpy(tkn.lexeme, "--");
        gotToken = 1;
        col += 2; //skip next as it is already included
    }
}
else if (c == '=')
{
    int d = fgetc(f);
    if (d != '=')
    {
        fillToken(&tkn, c, row, col);
        gotToken = 1;
        ++col;
        fseek(f, -1, SEEK_CUR); //go back 1 step *
    }
    else
    {
        fillToken(&tkn, c, row, col);
        strcpy(tkn.lexeme, "==");
        gotToken = 1;
        col += 2; //skip next as it is already included
    }
}
else if (isdigit(c))
{
    tkn.row = row;
    tkn.col = col++;
    tkn.lexeme[0] = c;
    int k = 1;
    while ((c = fgetc(f)) != EOF && isdigit(c))
    {
        tkn.lexeme[k++] = c;
        col++;
    }
    tkn.lexeme[k] = '\0';
    gotToken = 1;
    fseek(f, -1, SEEK_CUR); //go back 1 step *
}
else if (c == '#')
{
    while ((c = fgetc(f)) != EOF && c != '\n')
        ;
    newLine();
}
else if (c == '\n')
{
    newLine();
    c = fgetc(f);
    if (c == '#')
    {
        while ((c = fgetc(f)) != EOF && c != '\n')
            ;
```

```
                newLine();
            }
            else if (c != EOF)
                fseek(f, -1, SEEK_CUR);
        }
        else if (isspace(c))
            ++col;
        else if (isalpha(c) || c == '_')
        {
            tkn.row = row;
            tkn.col = col++;
            tkn.lexeme[0] = c;
            int k = 1;
            while ((c = fgetc(f)) != EOF && isalnum(c))
            {
                tkn.lexeme[k++] = c;
                ++col;
            }
            tkn.lexeme[k] = '\0';
            int check = 1;
            for(int i = 0 ; i < 7 ; i++){
                if(strcmp(tkn.lexeme,keywords[i]) == 0){
                    check = 0;
                    break;
                }
            }
            if(check == 1){
                strcpy(tkn.lexeme,"ID");
            }
            gotToken = 1;

            fseek(f, -1, SEEK_CUR);
        }
        else if (c == '/')
        {
            int d = fgetc(f);
            ++col;
            if (d == '/')
            {
                while ((c = fgetc(f)) != EOF && c != '\n')
                    ++col;
                if (c == '\n')
                    newLine();
            }
            else if (d == '*')
            {
                do
                {
                    if (d == '\n')
                        newLine();
                    while ((c == fgetc(f)) != EOF && c != '*')
                    {
```

```
                ++col;
                if (c == '\n')
                    newLine();
            }
            ++col;
        } while ((d == fgetc(f)) != EOF && d != '/' && (++col));
        ++col;
    }
    else
    {
        fillToken(&tkn, c, row, --col);
        gotToken = 1;
        fseek(f, -1, SEEK_CUR);
    }
}
else if (c == '"')
{
    tkn.row = row;
    tkn.col = col;
    int k = 1;
    tkn.lexeme[0] = '"';
    while ((c = fgetc(f)) != EOF && c != '"')
    {
        tkn.lexeme[k++] = c;
        ++col;
    }
    tkn.lexeme[k] = '"';
    gotToken = 1;
}
else if (c == '<' || c == '>' || c == '!')
{
    fillToken(&tkn, c, row, col);
    ++col;
    int d = fgetc(f);
    if (d == '=')
    {
        ++col;
        strcat(tkn.lexeme, "=");
    }
    else
    {
        fseek(f, -1, SEEK_CUR);
    }
    gotToken = 1;
}
else if (c == '&' || c == '|')
{
    int d = fgetc(f);
    if (c == d)
    {
        tkn.lexeme[0] = tkn.lexeme[1] = c;
        tkn.lexeme[2] = '\0';
```

```
                tkn.row = row;
                tkn.col = col;
                ++col;
                gotToken = 1;
            }
            else
            {
                tkn.lexeme[0] = c;
                tkn.lexeme[1] = '\0';
                tkn.row = row;
                tkn.col = col;
                ++col;
                gotToken = 1;
                fseek(f, -1, SEEK_CUR);
            }
            ++col;
        }
        else
            ++col;
    }
    return tkn;
}

int main()
{
    printf("Enter file name: ");
    char input[256];
    scanf("%s", input);
    FILE *f = fopen(input, "r");
    if (f == NULL)
    {
        printf("Cannot open file\n");
        exit(0);
    }

    struct token t;
    while ((t = getNextToken(f)).row != -1)
        printf("<%s, %d, %d>\n", t.lexeme, t.row, t.col);

    fclose(f);
    return 0;
}
```

Output:

//I don’t have Ubuntu on my system so I couldn’t run it and upload the screenshot