#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include <stdlib.h>

// Token types

typedef enum {

TOKEN\_KEYWORD,

TOKEN\_IDENTIFIER,

TOKEN\_OPERATOR,

TOKEN\_LITERAL,

TOKEN\_SEPARATOR,

TOKEN\_UNKNOWN,

TOKEN\_EOF

} TokenType;

// Token structure

typedef struct {

TokenType type;

char value[50];

} Token;

// List of keywords

const char \*keywords[] = {"if", "else", "while", "int", "print"};

const int keyword\_count = 5;

// List of operators

const char \*operators[] = {"+", "-", "\*", "/", "=", "==", "<", ">"};

const int operator\_count = 8;

// List of separators

const char \*separators = "(){};";

// Check if a string is a keyword

int is\_keyword(const char \*str) {

for (int i = 0; i < keyword\_count; i++) {

if (strcmp(str, keywords[i]) == 0)

return 1;

}

return 0;

}

// Check if a character is an operator

int is\_operator(char c) {

for (int i = 0; i < operator\_count; i++) {

if (strlen(operators[i]) == 1 && operators[i][0] == c)

return 1;

}

return 0;

}

// Check if a character is a separator

int is\_separator(char c) {

return strchr(separators, c) != NULL;

}

// Check if a character starts an identifier

int is\_identifier\_start(char c) {

return isalpha(c) || c == '\_';

}

// Check if a character is part of an identifier

int is\_identifier\_part(char c) {

return isalnum(c) || c == '\_';

}

// Check if a character starts a number

int is\_number\_start(char c) {

return isdigit(c);

}

// Get the next token from input

Token get\_next\_token(const char \*\*input) {

const char \*ptr = \*input;

Token token = {TOKEN\_UNKNOWN, ""};

// Skip whitespace

while (isspace(\*ptr)) {

ptr++;

}

// End of input

if (\*ptr == '\0') {

token.type = TOKEN\_EOF;

strcpy(token.value, "EOF");

\*input = ptr;

return token;

}

// Check for separators

if (is\_separator(\*ptr)) {

token.type = TOKEN\_SEPARATOR;

token.value[0] = \*ptr;

token.value[1] = '\0';

ptr++;

}

// Check for operators

else if (is\_operator(\*ptr)) {

token.type = TOKEN\_OPERATOR;

token.value[0] = \*ptr;

token.value[1] = '\0';

ptr++;

// Check for double-character operators

if (\*ptr == '=' && token.value[0] == '=') {

token.value[1] = '=';

token.value[2] = '\0';

ptr++;

}

}

// Check for identifiers or keywords

else if (is\_identifier\_start(\*ptr)) {

char buffer[50];

int i = 0;

while (is\_identifier\_part(\*ptr) && i < 49) {

buffer[i++] = \*ptr++;

}

buffer[i] = '\0';

if (is\_keyword(buffer)) {

token.type = TOKEN\_KEYWORD;

} else {

token.type = TOKEN\_IDENTIFIER;

}

strcpy(token.value, buffer);

}

// Check for literals

else if (is\_number\_start(\*ptr)) {

char buffer[50];

int i = 0;

while (isdigit(\*ptr) && i < 49) {

buffer[i++] = \*ptr++;

}

buffer[i] = '\0';

token.type = TOKEN\_LITERAL;

strcpy(token.value, buffer);

}

// Unknown token

else {

token.type = TOKEN\_UNKNOWN;

token.value[0] = \*ptr;

token.value[1] = '\0';

ptr++;

}

\*input = ptr;

return token;

}

// Main function for testing

int main() {

const char \*code = "int x = 42; if (x > 10) { print x; }";

const char \*input = code;

printf("Tokenizing code:\n%s\n\n", code);

Token token;

do {

token = get\_next\_token(&input);

printf("Token: %-12s Type: %d\n", token.value, token.type);

} while (token.type != TOKEN\_EOF);

return 0;

}