**COMPILER CONSTRUCTION LAB 3**

**Lexical Analyzer**

Lexical analyzer reads source code character by character and produces tokens for each valid word. Specialized buffering techniques thus have been developed to reduce the amount of overhead required to process a single input character.

Two pointers to the input are maintained:

Pointer *Lexeme Begin*, marks the beginning of the current lexeme, whose extent we are attempting to determine

Pointer *Forward,* scans ahead until a pattern match is found.

Once the next lexeme is determined, *forward* is set to character at its right end. Then, after the lexeme is recorded as an attribute value of a token returned to the parser, *Lexeme Begin* is set to the character immediately after the lexeme just found. If we use the scheme of Buffer pairs we must check, each time we advance forward, that we have not moved off one of the buffers; if we do, then we must also reload the other buffer. Thus, for each character read, we make two tests: one for the end of the buffer, and one to determine what character is read (the latter may be a multiway branch). We can combine the buffer-end test with the test for the current character if we extend each buffer to hold a sentinel character at the end. The sentinel is a special character that cannot be part of the source program, and a natural choice is the character **EOF.**

Note that **EOF** retains its use as a marker for the end of the entire input. Any **EOF** that

# appears other than at the end of a buffer means that the input is at an end

# 

# Pseudocode

# Question 1: Implement lexical analyzer using input buffering scheme

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <stdbool.h>

#include <string.h>

#define BUFFER\_SIZE 15

bool is\_operator(char ch) {

char oper[] = "!%^&\*-+=~|.<>/?";

for (int i = 0; i < 14; i++) {

if (ch == oper[i]) {

return true;

}

}

return false;

}

bool is\_special\_char(char ch) {//not a number or an alphabet

if (isalnum(ch) || ch == '\_' || is\_operator(ch)) {

return false;

}

return true;

}

void print\_type(char\* token) {

if (isalpha(token[0]) || token[0] == '\_') {

printf("%s is an identifier.\n", token);

}

else if (isdigit(token[0])) {

printf("%s is a number.\n", token);

}

else if (is\_operator(token[0])) {

printf("%s is an operator.\n", token);

}

}

int main() {

char buffer[BUFFER\_SIZE];

int lexeme\_begin = 0, forward = 0, j = 0, k = 0, flag = 0;

char token[BUFFER\_SIZE] = { "" };

//Taking input form user

printf("Enter a message: \n");

gets\_s(buffer, BUFFER\_SIZE);

//traversing message

for (int i = 0; i < BUFFER\_SIZE; i++) {

if (buffer[i] != '\0') {

if (is\_special\_char(buffer[i]))

{

for (int i = lexeme\_begin; i < forward; i++) {

token[j] += buffer[i];

j++;

}

print\_type(token);

memset(token, '\0', BUFFER\_SIZE);

forward++;

lexeme\_begin = forward;

j = 0;

}

else {

forward++;

}

}

else {

for (int i = lexeme\_begin; i < forward; i++) {

token[j] += buffer[i];

j++;

}

print\_type(token);

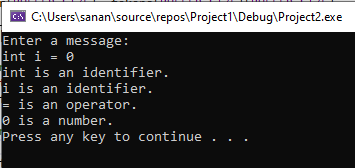
break;

}

}

system("pause");

}



# Question 2: Implement lexical analyzer using Buffer Pairs scheme

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <stdbool.h>

#include <string.h>

#define BUFFER\_SIZE 15

int is\_operator(char ch) {

char oper[] = "!%^&\*-+=~|.<>/?";

for (int i = 0; i < 14; i++) {

if (ch == oper[i]) {

return 1;

}

}

return 0;

}

int is\_special\_char(char ch) {//not a number or an alphabet

if (isalnum(ch) || ch == '\_' || is\_operator(ch)) {

return 0;

}

return 1;

}

void print\_type(char\* token) {

if (isalpha(token[0]) || token[0] == '\_') {

printf("%s is an identifier.\n", token);

}

else if (isdigit(token[0])) {

printf("%s is a number.\n", token);

}

else if (is\_operator(token[0])) {

printf("%s is an operator.\n", token);

}

}

int buffer\_run(char\* buffer) {

int lexeme\_begin = 0, forward = 0, j = 0;

char token[BUFFER\_SIZE] = { "" };

for (int i = 0; i < BUFFER\_SIZE; i++) {

if (buffer[i] != '\0') {

if (is\_special\_char(buffer[i]))

{

for (int i = lexeme\_begin; i < forward; i++) {

token[j] += buffer[i];

j++;

}

print\_type(token);

memset(token, '\0', BUFFER\_SIZE);

forward++;

lexeme\_begin = forward;

j = 0;

}

else {

forward++;

}

}

else {

for (int i = lexeme\_begin; i < forward; i++) {

token[j] += buffer[i];

j++;

}

print\_type(token);

break;

}

}

return forward;

}

int main() {

char input[50], buffer\_1[BUFFER\_SIZE], buffer\_2[BUFFER\_SIZE];

int flag = 0;

//Taking input form user

printf("Enter a message: \n");

gets\_s(input, BUFFER\_SIZE \* 2);

int forward = 0, k = 0, num;

//traversing message

for (int i = 0; k <= strlen(input); i++) {

num = k;

if (flag == 0) {

for (int j = 0; j <= BUFFER\_SIZE && input[i \* j] != '\0'; j++) {

buffer\_1[j] = input[num];

num++;

}

forward = buffer\_run(buffer\_1);

k = BUFFER\_SIZE \* i + forward;

flag = 1;

}

else {

for (int j = 0; j <= BUFFER\_SIZE && input[i \* j] != '\0'; j++) {

buffer\_2[j] = input[num];

num++;

}

forward = buffer\_run(buffer\_2);

k = BUFFER\_SIZE \* i + forward;

flag = 0;

}

}

}

# 

**Question 3: Design a regular expression for finding all the words starting with ‘t’ and ‘m’ in the following paragraph**

Once the next lexeme is determined, *forward* is set to character at its right end. Then, after the lexeme is recorded as an attribute value of a token returned to the parser, *Lexeme Begin* is set to the character immediately after the lexeme just found. If we use the scheme of Buffer pairs we must check, each time we advance forward, that we have not moved off one of the buffers; if we do, then we must also reload the other buffer. Thus, for each character read, we make two tests: one for the end of the buffer, and one to determine what character is read (the latter may be a multiway branch). We can combine the buffer-end test with the test for the current character if we extend each buffer to hold a sentinel character at the end. The sentinel is a special character that cannot be part of the source program, and a natural choice is the character **EOF.**

#define \_CRT\_SECURE\_NO\_WARNINGS  
#include <stdio.h>  
#include <stdlib.h>  
#include <stdbool.h>  
  
int main() {  
char para[] = "Once the next lexeme is determined, forward is set to character at its right end. Then, after the lexeme is recorded as an attribute value of a token returned to the parser, Lexeme Begin is set to the character immediately after the lexeme just found.If we use the scheme of Buffer pairs we must check, each time we advance forward, that we have not moved off one of the buffers; if we do, then we must also reload the other buffer.Thus, for each character read, we make two tests : one for the end of the buffer, and one to determine what character is read (the latter may be a multiway branch).We can combine the buffer - end test with the test for the current character if we extend each buffer to hold a sentinel character at the end. The sentinel is a special character that cannot be part of the source program, and a natural choice is the character.";  
char word[200] = "";  
bool flag = false;  
int i, j;  
for (i = 1; i < strlen(para); i++) {  
if ((para[i] == 'm' || para[i] == 'T' || para[i] == 't' || para[i] == 'M')  
&& (para[i - 1] == ' ' || i == 0)) {  
flag = true;  
}  
if (flag == true && para[i] == ' ') {  
printf("%s ", word);  
for (j = strlen(word); j >= 0; j--) {   
word[j] = '\0';   
}  
flag = false;  
}  
if (flag) {  
word[strlen(word)] = para[i];  
}  
}  
}

# 