American International University- Bangladesh (AIUB)

Faculty of Science and Technology

:

Course Name:	COMPILER DESIGN	**Course Code:**	CSC3216
Semester :	Spring 22-23	**Sec :**	E
Faculty :	**K. M. IMTIAZ-UD-DIN**	**Term :**	Mid
Marks :	20	**Assignment Name :**	Assignment

Student Name:	**PUNAM DAS**	**Student ID:**	**21-44946-2**	**Sl. No:**	
Submission Date:	**26th November, 2023**	**Department:**	**BSc CSE**		

Syntax Analyzer Code

```
 #include <iostream>
#include <string.h>

using namespace std;

int main() {

  // Declare variables for the syntax tree and input processing
  int syntax_tree_root, start_variable, top_rule, left_variable;
  int input_token_index, statement_index;
  string input_token;
  char statement[6];
  // Define grammar rules
  char grammar_rules[6][10] = {"A->BCDEF.", "B->i.", "C->=.", "D->i.", "E->+.", "F->;."};
  char variable_replacer[50];  // Array to store variable replacements
  char syntax_tree[50];       // Array to store the syntax tree
  int children_counter[50] = {0};  // Array to count children for each node

  cout << "Enter the input token:";  // Prompt the user to enter an input token
  getline(cin, input_token);  // Read the entire line of input

  cout << "The entered token was:" << input_token << endl;  // Display the entered token

  statement_index = 0;  // Initialize statement index
  statement[statement_index] = '\0';  // Null-terminate the statement array
```

```
// Initialize variables related to syntax tree construction
syntax_tree[syntax_tree_root] = grammar_rules[top_rule][left_variable];
variable_replacer[start_variable] = grammar_rules[top_rule][left_variable];

int variable_replacer_current_index = 0;
int syntax_tree_current_index = 1;
int statement_matcher_index = 0;

// Extract variables from input token enclosed in '<>' and display them
for (statement_index = 0; statement_index < 6; statement_index++) {
  if (input_token[input_token_index] == '<') {
    statement[statement_index] = input_token[input_token_index + 1];
    statement_index++;
    cout << statement[statement_index];
    cout << endl;
  }
}

// Initialize syntax tree variables
syntax_tree_root = 0;
start_variable = 0;
top_rule = 0;
left_variable = 0;
syntax_tree[syntax_tree_root] = grammar_rules[top_rule][left_variable];
variable_replacer[start_variable] = grammar_rules[top_rule][left_variable];
cout << "syntax Tree :   " << endl;

// Display the initial syntax tree
```

```
 for (int i = 0; 1 < 50 && syntax_tree[i] != '\0'; i++) {
  cout << syntax_tree[i];
 }
 cout << endl;

 // Main loop for syntax tree construction and matching
 while (true) {
  int grammar_rule_number, rule_right_side_copier_index, copy_flag, parent;

  // Iterate through grammar rules to find a match with the current variable
  for (grammar_rule_number = 0; grammar_rule_number < 6; grammar_rule_number++) {
   if (grammar_rules[grammar_rule_number][left_variable] ==
variable_replacer[variable_replacer_current_index]) {
    copy_flag = 0;
    parent = syntax_tree_current_index - 1;

    // Copy the right-hand side of the matching grammar rule to the syntax tree
    for (rule_right_side_copier_index = 0;
grammar_rules[grammar_rule_number][rule_right_side_copier_index] != '.';
rule_right_side_copier_index++) {
     if (grammar_rules[grammar_rule_number][rule_right_side_copier_index] == '>') {
      copy_flag = 1;
      continue;
     }

     if (copy_flag == 1) {
      children_counter[parent]++;
      syntax_tree[syntax_tree_current_index] =
grammar_rules[grammar_rule_number][rule_right_side_copier_index];
      syntax_tree_current_index++;
```

```
            variable_replacer[variable_replacer_current_index] =
grammar_rules[grammar_rule_number][rule_right_side_copier_index];

            variable_replacer_current_index++;

          }

        }

      }

    }


    variable_replacer_current_index = 0;


    // Match variables and terminals in the syntax tree with the input statement

    if (variable_replacer[variable_replacer_current_index] == 'i' ||
variable_replacer[variable_replacer_current_index] == '=' ||
variable_replacer[variable_replacer_current_index] == '+' ||
variable_replacer[variable_replacer_current_index] == ';') {

      if (variable_replacer[variable_replacer_current_index] == statement[statement_matcher_index]) {

        variable_replacer_current_index++;

        statement_matcher_index++;

      } //Display syntax_error if terminal symbols and do not match with the input token

      else {

        cout << "syntax_error";

        break;

      }

    }


    int i, break_flag = 1;


     // Check if all variables have been replaced with terminal symbols and display the remaining variables
in the syntax tree and input statement

    for (i = 0; i < 50; i++) {
```

```
    if (variable_replacer[i] != 'i' && variable_replacer[i] != '=' && variable_replacer[i] != '+' &&
variable_replacer[i] != ';') {
      cout << variable_replacer[i];
      cout << statement[i];
    } else {
    }
    break_flag = 0;
  }
```

Lexical Analyzer Code

```
#include <iostream>
#include <string.h>
using namespace std;

int main() {

  int i;  // Declare an integer variable i
  int id = 1;  // Initialize an identifier counter to 1
  int x = 0;  // Initialize an integer variable x to 0
  string statement;  // Declare a string variable named statement

  cout << "Enter the statement:";  // Prompt the user the enter a statement
  getline(cin, statement);  // Read the entire line of input

  cout << "The entered string is : " << statement << endl;  // Display the entered string
//process for reads the stream of characters and making up in to lexeme then print as token
  for (int i = 0; i < 15; i++) {  // Loop through the first 15 characters of the statement

    if (isalpha(statement[i])) {  // Check if the lexeme is an alphabet
// lexeme that is mapped into the token and print as token
      cout << "<id," << id << ">";  // Display an identifier with its corresponding id
      ++id;  // Increment the identifier counter
    } else if (isdigit(statement[i])) {  // Check if the lexeme is a digit
      if (isdigit(statement[i + 1])) {}  // If the next lexeme is also a digit
      else {
```

```
// lexeme that is mapped into the token and print as token

cout << "<" << statement[i - 1] << statement[i] << ">";  // Display a token form of the current and previous digits

      }

    } else {  // If the lexeme is neither an alphabet nor a digit

// all operator lexeme that is mapped into the token and print as token

      if (statement[i] == '=')

        cout << "<=>";  // print the equality operator

      if (statement[i] == '+')

        cout << "<+>";  // print the addition operator

      if (statement[i] == '-')

        cout << "<->";  // print  the subtraction operator

      if (statement[i] == '*')

        cout << "<*>";  // print the multiplication operator

      if (statement[i] == '/')

        cout << "</>";  // print the division operator

    }

  }


  return 0;  // Return 0 to indicate successful execution

}
```

Output Screenshot:

OnlineGDB beta
online compiler and debugger for c/c++

Welcome, Punam

Lexical Analyzer

Create New Project

My Projects

Classroom new

Learn Programming

Programming Questions

Upgrade

Logout

Learn Python with KodeKloud

Run | Debug | Stop | Share | Saved | { } Beautify | Language C++

main.cpp

```cpp
#include <iostream>
#include <string.h>
using namespace std;

int main() {

  int i;  // Declare an integer variable i
  int id = 1;  // Initialize an identifier counter to 1
  int x = 0;  // Initialize an integer variable x to 0
  string statement;  // Declare a string variable named statement

  cout << "Enter the statement:";  // Prompt the user the enter a statement
  getline(cin, statement);  // Read the entire line of input

  cout << "The entered string is : " << statement << endl;  // Display the entered string

  for (int i = 0; i < 15; i++) {  // Loop through the first 15 characters of the statement

    if (isalpha(statement[i])) {  // Check if the character is an alphabet
      cout << "<id," << id << ">";  // Display an identifier with its corresponding id
      ++id;  // Increment the identifier counter
    } else if (isdigit(statement[i])) {  // Check if the character is a digit
      if (isdigit(statement[i + 1])) {}  // If the next character is also a digit, do nothing
      else {
        cout << "<" << statement[i - 1] << statement[i] << ">";  // Display a number formed by the current and previous digits
      }
    } else {  // If the character is neither an alphabet nor a digit
```

input

```
Enter the statement:A=B+C*20
The entered string is : A=B+C*20
<id,1><=><id,2><+><id,3><*><20>

...Program finished with exit code 0
Press ENTER to exit console.
```

FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy