Name: Munna Chauhan

Reg. No.: CH.EN.U4CSE22176

Lab Exp.: 06

-----------------------------------------------------------------------------------------------------------------------------------

Aim: To implementation of intermediate code generation.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Global variables

int i = 1, j = 0, no = 0, tmpch = 90; // tmpch = 90 corresponds to 'Z'

char str[100], left[15], right[15];

// Structure for expression components

struct exp {

int pos;

char op;

} k[15]; // Array of structs to hold operators and their positions

// Function prototypes

void findopr();

void explore();

void fleft(int);

void fright(int);

// Main function

int main() {

printf("\t\tINTERMEDIATE CODE GENERATION\n\n");

printf("Enter the Expression: ");

scanf("%s", str); // Read the expression into str

printf("The intermediate code:\n");

findopr(); // Identify and store operators

explore(); // Generate intermediate code

return 0;

}

// Function to explore the operators and generate code

void explore() {

int i = 0;

// Loop through the stored operators until a null character is found in the op field

while (k[i].op != '\0') {

// Clear left and right strings for the current operation

fleft(k[i].pos);

fright(k[i].pos);

// Assign a temporary variable name (starting from 'Z' and decrementing)

str[k[i].pos] = tmpch--;

// Print the three-address code statement

printf("\tT%c := %s %c %s\n", str[k[i].pos], left, k[i].op, right);

i++;

}

// Process the final result after all operations are reduced

fright(-1); // Get the final expression (which should be a single character/variable)

if (no == 0) {

// If no operators were processed (i.e., it was a single operand)

fleft(strlen(str));

printf("\tT%s := %s\n", right, left);

exit(0); // Exit the program

}

// Print the final assignment

printf("\tT%c := %s\n", right, str[k[-i].pos]); // Note: k[-i] seems like a likely typo in the image, maybe it should be k[i-1] or a simple variable like T0.

// Based on the surrounding logic, it seems to be accessing the final temporary variable name.

// Assuming the original intent was to display the last generated temporary variable.

}

// Function to find the left operand for the operator at position x in str

void fleft(int x) {

int w = 0, flag = 0;

x--; // Start searching one character before the operator

// Loop backwards from x until an operator, '$' (which indicates a reduced expression), or -1 (start of string) is found

while (x != -1 && str[x] != '+' && str[x] != '\*' && str[x] != '=' &&

str[x] != '0' && str[x] != '-' && str[x] != '!' && str[x] != '/' &&

str[x] != ':') {

if (str[x] != '$' && flag == 0) {

left[w++] = str[x]; // Collect the character

left[w] = '\0';

str[x] = '$'; // Mark the character as processed (replaced by '$')

flag = 1;

}

x--;

}

// Reverse the left string because it is collected backwards

int start = 0, end = w - 1;

while (start < end) {

char temp = left[start];

left[start] = left[end];

left[end] = temp;

start++;

end--;

}

}

// Function to find the right operand for the operator at position x in str

void fright(int x) {

int w = 0, flag = 0;

// If x is not -1 (meaning it's not the final step)

if (x != -1) {

x++; // Start searching one character after the operator

} else {

x = 0; // Start from the beginning of the string for the final reduction

}

// Loop until an operator or null character is found

while (x != -1 && str[x] != '\0' && str[x] != '+' && str[x] != '\*' &&

str[x] != '=' && str[x] != ':' && str[x] != '!' && str[x] != '/' &&

str[x] != '-') {

if (str[x] != '$' && flag == 0) {

right[w++] = str[x]; // Collect the character

right[w] = '\0';

str[x] = '$'; // Mark the character as processed (replaced by '$')

flag = 1;

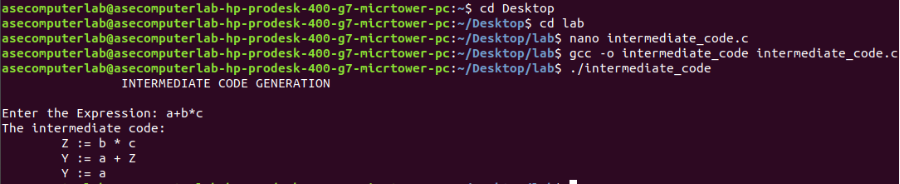
}

x++;

}

}

Output:



Result: Thus, the program to implement intermediate code generation has been executed successfully.