22BCE0682 Siddhant Bhagat

Lab Assessment 5

Q) Three Address Code:

Code:

```
#include <iostream>
#include <string>
#include <regex>

using namespace std;

bool isNumber(const string &input) {
    string modifiedInput = input;
    if (modifiedInput[0] == '#') {
        modifiedInput = modifiedInput.substr(1);
    }
    try {
        stod(modifiedInput);
        return true;
    } catch (invalid_argument &) {
        return false;
    }
}

bool isDecimal(const string &input) {
    string modifiedInput = input;
    if (modifiedInput[0] == '#') {
        modifiedInput = modifiedInput.substr(1);
    }
    return isNumber(modifiedInput) &&
           (modifiedInput.find('.') != string::npos ||
            modifiedInput.find('e') != string::npos ||
            modifiedInput.find('E') != string::npos);
}

bool isFloatTypeVariable(const string &variable) {
    return variable[0] == 'f' ||
           variable.find("float") != string::npos ||
           variable.find("decimal") != string::npos ||
           variable.find("real") != string::npos;
}

string transformToAssembly(const string &tac) {
    string sanitizedCode = regex_replace(tac, regex("\\s+"), "");
    size_t delimiterPosition = sanitizedCode.find(':');
    if (delimiterPosition == string::npos) {
        return "Invalid input format";
    }
```

```
    int operationCode = 0;
    for (char ch : tac) {
        if (ch == '+') operationCode = 1;
        else if (ch == '-') operationCode = 2;
        else if (ch == '*') operationCode = 3;
        else if (ch == '/') operationCode = 4;
    }

    string targetVariable = sanitizedCode.substr(0, delimiterPosition);
    string expressionPart = sanitizedCode.substr(delimiterPosition + 1);

    regex expressionPattern(R"((\w+)=(\w+|#?\d+(?:\.\d+)?)[+/*-
](\w+|#?\d+(?:\.\d+)?);?)");
    smatch matchedGroups;

    if (regex_search(expressionPart, matchedGroups, expressionPattern)) {
        string resultVar = matchedGroups[1];
        string firstOperand = matchedGroups[2];
        string secondOperand = matchedGroups[3];

        bool isResultFloat = isDecimal(firstOperand) || isDecimal(secondOperand) ||
                             isFloatTypeVariable(firstOperand) ||
isFloatTypeVariable(secondOperand);

        string assemblyCode;

        if (isNumber(secondOperand)) {
            assemblyCode += "MOV #" + secondOperand + ", R1\n";
        } else {
            assemblyCode += isResultFloat ? "MOVF " + secondOperand + ", R1\n" : "MOV
" + secondOperand + ", R1\n";
        }

        if (operationCode == 1) {
            if (isNumber(firstOperand)) {
                assemblyCode += (isDecimal(firstOperand) || isDecimal(secondOperand))
?
                                "ADDF #" + firstOperand + ", R1\n" :
                                "ADD #" + firstOperand + ", R1\n";
            } else {
                assemblyCode += isResultFloat ? "ADDF " + firstOperand + ", R1\n" :
"ADD " + firstOperand + ", R1\n";
            }
        } else if (operationCode == 2) {
            if (isNumber(firstOperand)) {
                assemblyCode += (isDecimal(firstOperand) || isDecimal(secondOperand))
?
                                "SUBF #" + firstOperand + ", R1\n" :
                                "SUB #" + firstOperand + ", R1\n";
            } else {
                assemblyCode += isResultFloat ? "SUBF " + firstOperand + ", R1\n" :
"SUB " + firstOperand + ", R1\n";
```

```
            }
        } else if (operationCode == 3) {
            if (isNumber(firstOperand)) {
                assemblyCode += (isDecimal(firstOperand) || isDecimal(secondOperand))
?
                                "MULF #" + firstOperand + ", R1\n" :
                                "MUL #" + firstOperand + ", R1\n";
            } else {
                assemblyCode += isResultFloat ? "MULF " + firstOperand + ", R1\n" :
"MUL " + firstOperand + ", R1\n";
            }
        } else if (operationCode == 4) {
            if (isNumber(firstOperand)) {
                assemblyCode += (isDecimal(firstOperand) || isDecimal(secondOperand))
?
                                "DIVF #" + firstOperand + ", R1\n" :
                                "DIV #" + firstOperand + ", R1\n";
            } else {
                assemblyCode += isResultFloat ? "DIVF " + firstOperand + ", R1\n" :
"DIV " + firstOperand + ", R1\n";
            }
        }

        assemblyCode += isResultFloat ? "MOVF R1, " + resultVar : "MOV R1, " +
resultVar;

        return assemblyCode;
    }

    return "Invalid expression format";
}

int main() {
    string userInput;
    cout << "Enter the Three Address Code: ";
    getline(cin, userInput);
    string fullInput = "Enter the Three Address Code : " + userInput;
    cout << transformToAssembly(fullInput) << endl;
    return 0;
}
```

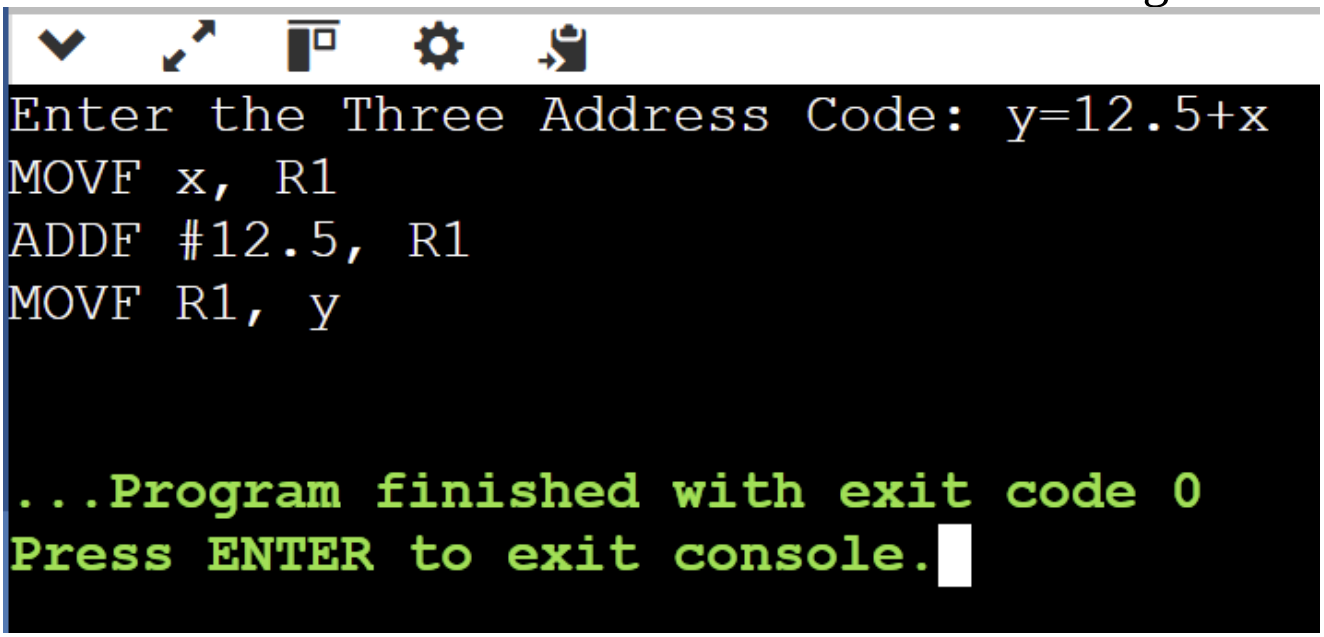
OUTPUT:

```
Enter the Three Address Code: x=ay+by
MOV by, R1
ADD ay, R1
MOV R1, x


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

```
Enter the Three Address Code: x=10*ay
MOV ay, R1
MUL #10, R1
MOV R1, x


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


Enter the Three Address Code: y=12.5+x
MOVF x, R1
ADDF #12.5, R1
MOVF R1, y
...Program finished with exit code 0
Press ENTER to exit console.

22BCE0682 Siddhant Bhagat

COMPILER LAB-5

Q)

Implementation of Constant Folding Optimization

Opens: Thursday, 24 October 2024, 5:20 PM
Due: Thursday, 24 October 2024, 5:30 PM

Write a program to implement constant folding optimization using a C/C++ program for the following expressions and print the optimized constants.

A = b+22*26.5

A= B+45*10/15+100

X = Y+2024*31/21+1979-2000

CODE:

22BCE0682 Siddhant Bhagat

constantfold.cpp

```cpp
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int is_digit(char c)
    {
        return c >= '0' && c <= '9';
    }
int is_letter(char c)
    {
        return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');
    }
        double extract_constant(char *exp, int *i)
    {
        double num = 0.0, factor = 1.0;
int is_fraction = 0;
    while (is_digit(exp[*i]) || exp[*i] == '.')
    {
        if (exp[*i] == '.')
    {
        is_fraction = 1;
        (*i)++;
    }
```

```
    else
    {
    num = num * 10 + (exp[*i] - '0');
    if (is_fraction)
    {
        factor *= 10.0;
        }
        (*i)++;
        }
        }
        return num / factor;
    }
    void extract_variable(char *exp, int *i, char *output, int *j)
    {
        while (is_letter(exp[*i]))
    {
        output[(*j)++] = exp[*i];
        (*i)++;
    }
        }
```

```
43      int main()
44      {
45          int var=0;
46          int cons=0;
47          int sum=0;
48          char expression[200], optimized[200];
49          int i = 0, j = 0;
50          double result = 0, current_value = 0;
51          char current_op = '+';
52          printf("Enter the input string: ");
53          scanf("%[^\n]s", expression);
54          printf("The Constants are: ");
55      while (expression[i] != '=' && expression[i] != '\0')
56      {
57          optimized[j++] = expression[i++];
58      }
59      if (expression[i] == '=')
60      {
61          optimized[j++] = expression[i++];
62      }
```

```
63      while (expression[i] != '\0')
64      {
65      if (is_letter(expression[i]))
66      {
67          extract_variable(expression, &i, optimized, &j);
68          var++;
69  }
70      else if (is_digit(expression[i]) || expression[i] == '.')
71      {
72          double num = extract_constant(expression, &i);
73          cons++;
74          printf("%.2f ", num);
75      if (current_op == '+')
76      {
77          result += current_value;
78          current_value = num;
79          }
80      else if (current_op == '-')
81      {
82          result += current_value;
83          current_value = -num;
84          }
```

22BCE0682 Siddhant Bhagat

```
    else if (current_op == '*')
    {
        current_value *= num;
        }
    else if (current_op == '/')
    {
        current_value /= num;
        }
    }
    else if (expression[i] == '+' || expression[i] == '-' || expression[i] == '*' || expression[i] == '/')
    {
        current_op = expression[i++];
    }
    else
    {
        i++;
        }
            }
        result += current_value;
        optimized[j] = '\0';
        sum=var+cons;
        printf("\nOptimized expression: %s+%.2f\n", optimized, result);
        printf("Before Optimization:%d\n",sum);
        printf("After Optimization:%d\n",var+1);
        printf("The value of the constant expression is: %.2f\n", result);
        return 0;
        }
```

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int is_digit(char c)
{
return c >= '0' && c <= '9';
}
int is_letter(char c)
{
return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');
}
double extract_constant(char *exp, int *i)
{
double num = 0.0, factor = 1.0;
int is_fraction = 0;
while (is_digit(exp[*i]) || exp[*i] == '.')
{
if (exp[*i] == '.')
{
is_fraction = 1;
```

```
(*i)++;
}
else
{
num = num * 10 + (exp[*i] - '0');
if (is_fraction)
{
factor *= 10.0;
}
(*i)++;
}
}
return num / factor;
}
void extract_variable(char *exp, int *i, char *output, int *j)
{
while (is_letter(exp[*i]))
{
output[(*j)++] = exp[*i];
(*i)++;
}
}
int main()
{
int var=0;
int cons=0;
int sum=0;
char expression[200], optimized[200];
int i = 0, j = 0;
double result = 0, current_value = 0;
char current_op = '+';
printf("Enter the input string: ");
scanf("%[^\n]s", expression);
printf("The Constants are: ");
while (expression[i] != '=' && expression[i] != '\0')
{
optimized[j++] = expression[i++];
}
if (expression[i] == '=')
{
optimized[j++] = expression[i++];
```

```
}
while (expression[i] != '\0')
{
if (is_letter(expression[i]))
{
extract_variable(expression, &i, optimized, &j);
var++;
}
else if (is_digit(expression[i]) || expression[i] == '.')
{
double num = extract_constant(expression, &i);
cons++;
printf("%.2f ", num);
if (current_op == '+')
{
result += current_value;
current_value = num;
}
else if (current_op == '-')
{
result += current_value;
current_value = -num;
}
else if (current_op == '*')
{
current_value *= num;
}
else if (current_op == '/')
{
current_value /= num;
}
}
else if (expression[i] == '+' || expression[i] == '-' || expression[i] == '*' || expression[i]
== '/')
{
current_op = expression[i++];
}
else
{
i++;
}
```

```
}
result += current_value;
optimized[j] = '\0';
sum=var+cons;
printf("\nOptimized expression: %s+%.2f\n", optimized, result);
printf("Before Optimization:%d\n",sum);
printf("After Optimization:%d\n",var+1);
printf("The value of the constant expression is: %.2f\n", result);
return 0;
}
```

OUTPUT:

```
matlab@sjt318scope049:~/22bce682$ gcc constantfold.cpp
matlab@sjt318scope049:~/22bce682$ ./a.out
Enter the input string: a=b+22*26.5;
The Constants are: 22.00 26.50
Optimized expression: a=b+583.00
Before Optimization:3
After Optimization:2
The value of the constant expression is: 583.00
matlab@sjt318scope049:~/22bce682$ ./a.out
Enter the input string: a=b+45*10/15+100;
The Constants are: 45.00 10.00 15.00 100.00
Optimized expression: a=b+130.00
Before Optimization:5
After Optimization:2
The value of the constant expression is: 130.00
```

```
matlab@sjt318scope049:~/22bce682$ x=y+2024*31/21+1979+2000;
matlab@sjt318scope049:~/22bce682$ ./a.out
Enter the input string: x=y+2024*31/21+1979+2000;
The Constants are: 2024.00 31.00 21.00 1979.00 2000.00
Optimized expression: x=y+6966.81
Before Optimization:6
After Optimization:2
The value of the constant expression is: 6966.81
```