**The University of Technology**

Written Group Report

Group members: Brandon Chung, Ryhane Roberts, Phillip Drummond

Id#: 1603729, 1603792, 1606155

Tutor: Phillip Smith

Group Contributions

Brandon Chung: Developed near to all of the code implemented in the project.

Implementation using the Java or C++ Programming Language.

User interface and Visual display of calculator model.

Storage of current expression in text file.

Correct functionality (entry/evaluation of operators, operations, keys.)

Input validation etc.

Phillip Drummond:

Added documentation.

Created the User Manual.

Ryhane Smith:

Added the Big O Notation below.

Big O Notation

Node.h – All methods present are constant

OperatorNode.h – All methods present are constant.

OperatorStack.h – All other methods not listed are constant.

O(f(count())) - Linear (n)

int count(){

1 if(!top){

1 return 0;

}

1 int count = 0;

1 OperatorStack \* tempOperatorStack = new OperatorStack();

N while(top){

1 count++;

1 tempOperatorStack -> push(this -> stackTop());

1 this -> pop();

}

N for(int x = 0; x < count; x++){

1 this -> push(tempOperatorStack -> stackTop());

1 tempOperatorStack -> pop();

}

1 return count;

}

Big O(f(count()))

= 4+3n+2n+1

= 5+5n

= 5n

= n

O(f(empty())) - Linear (n)

void empty(){

n while(top){

1 OperatorNode \* temp = top;

1 top = top -> getNextNode();

1 delete temp;

}

}

BigO(f(empty())) = 3n

= n

Stack – Similarly to count() and empty() above these methods are linear. All other methods are constant.

Main –

O(f(solveFace(std::string face))) - Linear

void solveFace(std::string face){

n while(face.back() == ' '){

1 face = face.substr(0, face.size() - 1);

}

1 std::stringstream buffer;

1 buffer << face;

N while(buffer.rdbuf() -> in\_avail()){

1 std::string currentValue;

1 buffer >> currentValue;

1 if(isNumber(currentValue)){

1 numberStack -> push(std::stod(currentValue));

1 } else if(isOperator(currentValue) && currentValue.length() == 1){

1 calculateOnStack(currentValue[0]);

1 } else{

1 numberStack -> empty();

1 return;

}

}

}

BigO(f(solveFace(std::string face))) - Linear

= 3n + 9n

= n

O(f(trimStringZeroes(std::string s))) – Linear

std::string trimStringZeroes(std::string s){

/\* Deletes characters until a number other than zero or period is found. \*/

N while(s.back() == '0' && s.size() > 0){

1 s = s.substr(0, s.size() - 1);

1 if(s.back() == '.'){

1 return s.substr(0, s.size() - 1);

}

}

1 return s;

}

BigO(f(trimStringZeroes(std::string s)))

= 3n + 1

= 3n

= n

O(f(convertInfixToPostfix(std::string infix))) - Quadratic

std::string convertInfixToPostfix(std::string infix){

1 OperatorStack \* operatorStack = new OperatorStack();

1 std::string postfix = "";

1 std::stringstream source;

1 source << infix;

n while(source.rdbuf() -> in\_avail()){

1 std::string data;

1 source >> data;

1 if(isNumber(data)){

1 postfix = postfix + data + " ";

1 } else if(isOperator(data) && data.length() == 1){

1 if(operatorStack -> count() == 0){

1 operatorStack -> push(data[0]);

1 } else {

1 if(operatorPriority(data[0]) < operatorPriority(operatorStack -> stackTop())) {

n while (operatorStack->count() > 0 && operatorPriority(data[0]) < operatorPriority(operatorStack- >stackTop())) {

1 postfix = postfix + (operatorStack->stackTop()) + " ";

1 std::cout << "ddd" << std::endl;

1 operatorStack->pop();

1 }

1 }

1 operatorStack -> push(data[0]);

1 }

1 } else {

1 operatorStack -> empty();

1 throw -5;

}

}

n while(operatorStack -> count() > 0){

1 postfix = postfix + (operatorStack -> stackTop()) + " ";

1 operatorStack -> pop();

}

1 delete operatorStack;

1 return postfix;

}

BigO(f(convertInfixToPostfix(std::string infix)))

= 4+ 9n(9n) + 2n + 2

= 18n2 + 2n

= n2 + n

= n2

O(f(main())) - Polynomial

1 char input = 0;

n while(input != 'x') { /\* Entering x exits calculator. \*/

n while (input != 'x' && input != '=') {

1 paintCalculator();

1 input = getch(); /\* Receives user input \*/

1 if (checkInput(input)) {

1 calculatorFace = calculatorFace + input;

1 } else if (input == 8) {

1 calculatorFace = calculatorFace.substr(0, calculatorFace.size() - 1);

1 } else if (input == 'm'){

1 loadMemory();

1 } else if(input == 'f'){

1 storeAnswer();

1 } else if(input == 'c'){

1 clearFace();

1 } else if(input == 'p'){

1 try {

N2 calculatorFace = convertInfixToPostfix(calculatorFace);

1 } catch (int i){

1 calculatorFace = "";

}

}

}

1 if(input == '=') {

1 try {

n solveFace(calculatorFace);

1 if (numberStack->count() == 1) {

1 overwriteMemory();

1 calculatorFace = calculatorFace + "\n" + std::to\_string(numberStack->stackTop());

1 } else {

1 throw -4;

}

1 } catch (int i) {

1 if (i == -1 || i == -2) {

1 calculatorFace = "Syntax error";

1 } else if (i == -3) {

1 calculatorFace = "Invalid symbol entered";

1 } else if (i == -4) {

1 calculatorFace = "Invalid expression";

}

1 } catch (const std::out\_of\_range& oor) { /\* Validates whether or not entered double is too large. \*/

1 calculatorFace = "Number too large\n";

}

n calculatorFace = trimStringZeroes(calculatorFace);

1 paintCalculator();

1 clearFace();

n numberStack->empty();

1 input = getch();

1 if (checkInput(input)) {

1 calculatorFace = calculatorFace + input;

1 } else if (input == 8) {

1 calculatorFace = calculatorFace.substr(0, calculatorFace.size() - 1);

1 } else if (input == 'm'){

1 loadMemory();

1 } else if(input == 'f'){

1 storeAnswer();

1 } else if(input == 'c'){

1 clearFace();

}

}

}

BigO(f(main()))

= 1 + n(n(16n + n2) + 29 + 3n)

= 1 + 16n3 + n4 + 29n + 3n2

=1 + n3 + n4 + n + n2

=n4