|  |  |
| --- | --- |
| Программа для преобразования неравенства к операции «меньше» | |
| Программа и методика испытаний | |
| Студент | Еремеев М.А. |
| Преподаватель | доц. Сычев О.А. |
| Сдано |  |

Приложение 1

Методика тестирования функции isOperand

/\*!

\* \Проверяет, является ли строка операндом

\* \param[in] str – строка, которая будет проверена

\* \return - true, если строка операндом; false, если строка не является операндом

\*/

bool isOperand(std::string str)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Названии теста | Входные данные | Ожидаемый  результат |
| 1 | Все символы - цифры | 1234 | true |
| 2 | Строка начинается с «0» cо всеми цифрами меньше 8 | 01234 | true |
| 3 | Строка начинается с «0» c использованием цифр больше 7 | 01238 | false |
| 4 | Строка является шестнадцатеричным числом без использования букв | 0x1234 | true |
| 5 | Строка начинается с «0x» с использованием допустимых букв | 0xABCD | true |
| 6 | Строка начинается с «0x» с использованием недопустимых букв | 0xABCZ | false |
| 7 | Строка начинается с буквы | A1234 | true |
| 8 | Строка начинается на «\_» | \_1234 | true |
| 9 | Строка начинается с цифры, но содержит буквы | 1a234 | false |
| 10 | В имени переменной содержатся недопустимые символы | \_12\*4/ | false |
| 11 | Строка начинается с недопустимого символа | (abc | false |
| 12 | В числе недопустимые символы | 12:23 | false |

Приложение 2

Методика тестирования функции checkRootOperation

/\*!

\* \Проверяет, является ли корневая операция допустимой

\* \param[in] \_operator – тип операции

\* \return - true, если операция допустима; false, если строка операция не допустима

\*/

bool checkRootOperator( ExprNodeType \_operator)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Названии теста | Входные данные | Ожидаемый  результат |
| 1 | Корневая операция - «+» | Plus | false |
| 2 | Корневая операция - «-» | Minus | false |
| 3 | Корневая операция - «/» | Division | false |
| 4 | Корневая операция - «\*» | Multiplication | false |
| 5 | Корневая операция - «%» | DivRemainder | false |
| 6 | Корневая операция - «~» | UnaryMinus | false |
| 7 | Корневая операция - «!» | Not | false |
| 8 | Корневая операция - «>=» | GreaterEqual, | true |
| 9 | Корневая операция - «<=» | LessEqual | true |
| 10 | Корневая операция - «>» | Greater | true |
| 11 | Корневая операция - «<» | Less | true |
| 12 | Корневой узел - операнд | Operand | false |

Приложение 3

Методика тестирования функции transformInequalityToLessOperator

/\*!

\* \Преобразует узел дерева разбора выражения к операции «меньше»

\* \param[in/out] node – указатель на узел, который должен быть преобразован

\*/

void transformInequalityToLessOperator( ExprNode \* node)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Названии теста | Входные данные | Ожидаемый  результат |
| 1 | Текущая операция – «меньше» |  |  |
| 2 | Текущая операция – «меньше или равно» |  |  |
| 3 | Текущая операция – «больше» |  |  |
| 4 | Текущая операция – «больше или равно» |  |  |

Приложение 4

Методика тестирования функции swapOperands

/\*!

\* \Меняет местами операнды узла дерева разбора выражения

\* \this – узел, операнды которого поменяются местами

\*/

void ExprNode :: swapOperands()

|  |  |  |  |
| --- | --- | --- | --- |
| № | Названии теста | Входные данные | Ожидаемый  результат |
| 1 | Операнды являются значениями |  |  |
| 2 | Операнды содержат узлы |  |  |

Приложение 5

Методика тестирования функции addUnaryOperationBefore

/\*!

\* \Добавляет унарную операцию перед текущей

\* \param[in] \_operator- операция, которая будет добавлена

\*/

void ExprNode :: addUnaryOperatorBefore ( ExprNodeType \_operator)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Названии теста | Входные данные | | Ожидаемый  результат |
| Текущий узел | Операция |
| 1 | Текущий узел - значение |  | Not |  |
| 2 | Текущий узел - операция |  | Not |  |
| 3 | Текущий узел - промежуточный |  | Not |  |

Приложение 6

Методика тестирования функции getRpnOfExprTree

/\*!

\* \Создает строку с обратной польской

\* \return – строка с обратной польской записью

\*/

std::string ExprNode :: getRpnOfExprTree()

| № | Названии теста | Входные данные | Ожидаемый  результат |
| --- | --- | --- | --- |
| 1 | Корневой узел не имеет дочерних |  | x 5 > |
| 2 | Корневой узел имеет дочерние |  | x 5 + 7 <= |
| 3 | Корневой узел - значение |  | 5 |
| 4 | Дерево содержит унарные операции |  | x ~ 5 >= |
| 5 | Проверка операторов «-», «\*», «/» |  | 5 x - 7 x \* / |
| 6 | Проверка операторов «%», «!», «<» |  | 5 x % 7 ! < |
| 7 | Строка содержит переменные на «\_» |  | \_123 1 + |
| 8 | Строка содержит восьмеричные числа |  | 0127 1 + |
| 9 | Строка содержит шестнадцатеричные числа |  | 0x12AB 1 + |

Приложение 7

Методика тестирования функции stringToExprTree

/\*!

\* \Создает дерево разбора выражения строки в обратной польской записи

\* \param[in] rpnString – строка в обратной польской записи

\* \param[out] errors – указатель контейнер с ошибками

\* \return – указатель на дерево разбора выражения

\*/

ExprNode \* stringToExprTree(std::string rpnString, std::set <Error> \* errors)

| № | Названии теста | Входные данные | Ожидаемый результат | |
| --- | --- | --- | --- | --- |
| ExprNode \* | \* errors |
| 1 | Строка содержит одну операцию | 5 x > |  | - |
| 2 | Строка содержит вложенные операции | 5 x + 7 >= |  | - |
| 3 | Строка содержит унарные операции | 5 ~ x <= |  | - |
| 4 | Проверка операторов «-», «\*», «/» | 5 x - 7 x \* / |  | - |
| 5 | Проверка операторов «%», «!», «<» | 5 x % 7 ! < |  | - |
| 6 | Строка содержит переменные на «\_» | \_123 1 + |  | - |
| 7 | Строка содержит восьмеричные числа | 0127 1 + |  | - |
| 8 | Строка содержит шестнадцатеричные числа | 0x12AB 1 + |  | - |
| 9 | Строка содержит только значение | 5 |  | - |
| 10 | В строке недостаточно операндов для операции | 5 x + > | - | Error{  type = missingOperand;  strWithError = “>”;  position = 4;  } |
| 11 | Строка содержит лишний операнд | 5 x 6 > | - | Error{  type = redundantOperand;  strWithError = “5”;  position = 1;  } |
| 12 | Строка содержит неизвестный символ | 5 ( > | - | Error{  type = unknownSymbol;  strWithError = “(”;  position = 2;  } |
| 13 | Строка содержит комбинацию неизвестных символов | \_12\*4/ | - | Error{  type = unknownSymbol;  strWithError = “\_12\*4/”;  position = 1;  } |