Министерство науки и высшего образования Российской Федерации

Казанский национальный исследовательский технический университет – КАИ им. А.Н. Туполева

Институт компьютерных технологий и защиты информации

Отделение СПО ИКТЗИ «Колледж информационных технологий»

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Методические указания к лабораторной работе №9

Тема: «Разработка командной строки для управления ходом выполнения программы»

Казань 2024

Составитель преподаватель СПО ИКТЗИ Шмидт Ильдар Рафаилович

Методические указания к лабораторным работам по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ» предназначены для студентов направления подготовки 09.02.07 «Информационные системы и программирование»

КНИТУ-КАИ, 2024

ПРОЦЕСС СДАЧИ ВЫПОЛНЕННОЙ РАБОТЫ

По итогам выполнения работы студент:

1. демонстрирует преподавателю правильно работающие программы;
2. демонстрирует приобретённые знания и навыки отвечает на пару небольших вопросов преподавателя по составленной программе, возможностям её доработки;
3. демонстрирует отчет по выполненной лабораторной работе.

Итоговая оценка складывается из оценок по трем указанным составляющим.

ЛАБОРАТОРНАЯ РАБОТА №9

ТЕМА: «РАЗРАБОТКА КОМАНДНОЙ СТРОКИ ДЛЯ УПРАВЛЕНИЯ ХОДОМ ВЫПОЛНЕНИЯ ПРОГРАММЫ»

ЦЕЛЬ РАБОТЫ

Изучение алгоритмов вычисления функциональных выражений в обратной польской записи и особенностей их программной реализации.

ХОД РАБОТЫ

1. Модифицированный алгоритм вычисления функциональных выражений в обратной польской записи

Как правило арифметические выражения удобно преобразовывать в обратную польскую запись (ОПЗ), чтобы избавиться от скобок, содержащихся в выражении. Известный алгоритм можно преобразовать для случая функциональных выражений, представив ее как n-местную операцию.

Предположим, у нас есть функция многих переменных $f(x, y, z)$, выполняющее определенное действие в программе. В соответствии с обратной польской записью она будет выглядеть иначе – *x y z f*

В выражении *x y z f* выделим следующие элементы, актуальные для дальнейшего анализа:

1. ‘x’ – переменная, хранящая значение;
2. ‘y’ – переменная, хранящая значение;
3. ‘z’ – переменная, хранящая значение;
4. ‘ ’ – оператор, являющийся символом разделителем между аргументами функции;
5. ‘f’ – оператор, определяющий выполнение функции с тремя параметрами, записанными в аргументы.

Для обработки функциональных выражений определим два стека: **стек операндов**, где будут храниться значения, передаваемые в функцию, и **стек операторов**, но в программной реализации ограничимся лишь **стеком операторов**.

Стек – это линейная структура данных, в которую элементы добавляются и удаляются **только с одного конца**, называемого **вершиной** стека. Стек работает по принципу «элемент, помещенный в стек последним, извлечен будет первым». Иногда этот принцип обозначается сокращением **LIFO** (Last In – First Out, т.е. последним зашел – первым вышел).

Линейная структура данных «Стек» реализована в классе `Stack<T>` пространства имен `using System.Collections.Generic;`.

В классе `Stack` можно выделить два основных метода, которые позволяют управлять элементами:

1. `Push`: добавляет элемент в стек на первое место;
2. `Pop`: извлекает и возвращает первый элемент из стека;
3. `Peek`: просто возвращает первый элемент из стека без его удаления.

Алгоритм обработки входной строки, представляющей собой функциональное выражение, следующий:

1. Рассматриваем поочередно каждый символ. Если этот символ – число или символ, не являющийся оператором (например, символьный аргумент функции), то помещаем его в стек операндов;

2. Если текущий символ - знак операции, то помещаем его в стек операторов;

3. Если текущий символ – знак оператора-разделителя, то игнорируем его;

9. Если текущий символ - открывающая скобка, то помещаем ее в стек операций;

5. Если текущий символ - закрывающая скобка, то извлекаем символы из стека операций до тех пор, пока не встретим в стеке открывающую скобку, которую следует просто уничтожить. Закрывающая скобка также уничтожается;

6. После обработки входной строки извлекаем элемент из стека операций и выполняем метод, соответствующий данному знаку операции. В

параметры найденного метода передаем извлекаемые поочередно элементы из стека операций.

3. Программирование модифицированного алгоритма

Для имитации работы командной строки разместим в главной форме элемент TextBox, который позволяет пользователю вводить текст команды для дальнейшей обработки. Переименуем в последующем описании данный элемент как `textBoxInputString`.

Напишем статический класс ReversePolishNotation для обработки команды, в котором разобьем задачу на три метода:

1. `bool CalculateRPN(string expression)` – открытый метод, предназначенный для обработки выражения в обратной польской записи;
2. `bool IsOperator(char c)` – закрытый метод, проверяющий переданный символ на соответствие какому либо оператору;
3. `bool ApplyOperation(Stack<object> operands, char c)` – закрытый метод, который применяет операцию над операндами переданными в стеке.

Листинг 1.1. Класс RPN

```
1  static class RPN {
2    public static bool CalculateRPN(string expression) {…}
3    private static bool IsOperator(char c) {…}
4    private static bool ApplyOperation(Stack<object> operands, char c) {…}
5  }
```

CalculateRPN

Этот метод принимает строку `expression`, которая представляет собой арифметическое выражение в обратной польской записи. Он итерирует по каждому символу в этом выражении.

- Если символ является цифрой, он преобразует его в int и добавляет в стек операндов.

- Если символ является оператором (определенным в методе IsOperator), он вызывает метод ApplyOperation для применения операции.
- Если символ не является ни цифрой, ни оператором, он добавляет его в стек операндов.

Этот метод можно сократить, так как, и цифры, и символы, могут быть операндами, но не стоит спешить, предложенный метод обрабатывает лишь однозначные числа, эту проблему вам придется решить самостоятельно.

Реализация данного метода представлена в листинге 1.2.

Листинг 1.2. Метод CalculateRPN

Публичный статический метод CalculateRPN с аргументом expression	
1	Создать стек operandsStack
2	Для каждого символа ‘c’ в expression
2.1	Если ‘c’ является цифрой
2.1.1	Добавить ‘c’ в operandsStack
2.2	Иначе если ‘c’ является оператором
2.2.1	Если не удалось применить операцию ApplyOperation(operandsStack, c)
2.2.1.1	Вернуть false
2.3	Иначе
2.3.1	Добавить ‘c’ в operandsStack
3	Вернуть true

IsOperator

Этот метод проверяет, является ли переданный символ оператором (например, ‘M’ – перемещение фигуры, ‘R’ – создание и рисование прямоугольника, ‘D’ – удаление фигуры), если да, то возвращает `true`, иначе `false`.

Реализация функции, проверяющей на несоответствие знаку операции, представлена в листинге 1.3.

Листинг 1.3. Метод IsOperator

Приватный статический метод IsOperator с аргументом ‘c’	
1	Вернуть true, если c равно 'R' или 'M' или 'D', иначе вернуть false

ApplyOperation

Этот метод применяет операцию, если возможно, в противном случае возвращает `false`.

Рассмотрим возможное поведение этого метода:

- Если оператор 'R' и в стеке операндов есть 5 элементов (x, y, ширина, высота и имя фигуры), создается прямоугольник и добавляется в контейнер фигур.
- Если оператор 'M' и в стеке операндов есть 3 элемента (x, y, имя перемещаемой фигуры), ищется фигура с заданным именем и перемещается в указанные координаты.
- Если оператор 'D' и в стеке операндов есть 1 элемент (имя удаляемой фигуры), ищется фигура с заданным именем и удаляется.

Листинг 1.4. Условие несоответствия знаку операции

Приватный статический метод ApplyOperation с аргументами operands и ‘c’	
1	Если ‘c’ равно 'R' и количество элементов в operands равно 5
1.1	Создать Rectagle с параметрами извлеченными из стека operands
1.2	Вызвать метод Draw() для rectagle
1.3	Вызвать метод AddFigure(rectagle) у ShapeContainer
2	Иначе если c равно 'M' и количество элементов в operands равно 3
2.1	Найти фигуру fig в ShapeContainer.figureList, у которой Name равно извлеченному значению из стека operands
2.2	Если fig не равно null
2.2.1	Вызвать метод MoveTo(значения из operands) для fig
3	Иначе если ‘c’ равно 'D' и количество элементов в operands равно 1

3.1	Найти фигуру fig в ShapeContainer.figureList, у которой Name равно извлеченному значению из стека operands
3.2	Если fig не равно null
3.2.1	Вызвать метод DeleteF() для fig
4	Иначе
4.1	Вернуть false
5	Вернуть true

При попытках найти фигуру с определенным именем (строки 2.1 и 3.1, листинг 1.4) важно понимать, что нет необходимости извлекать символ из вершины стека полностью, а нужно лишь воспользоваться им временно, то есть использовать метод Peek(), а не Pop() у стека.

Рассмотрим обработку команды “`n 8 4 1 1 R`”, в которой

- n – имя фигуры;
- 8 – высота;
- 4 – ширина;
- 1 – начальный Y;
- 1 – начальный X.

1. Обработка символов происходит слева направо;
2. Поскольку `n` не является операндом и числом, то этот символ помещается в стек блоком `else` (строка 2.3, листинг 1.2);

operandsStack	n

3. Символы ‘`8`’, ‘`4`’, ‘`1`’, ‘`1`’ являются цифрами, а значит они помещаются в стек блоком `if` (строка 2.1, листинг 1.2);

operandsStack	n 8 4 1 1

4. Доходим до символа ‘`R`’, с помощью метода `IsOperator` узнаем, что данный символ является оператором (строка 2.2, листинг 1.2). Не кладем его в стек, а сразу же пытаемся выполнить операцию методом

ApplyOperation, передавая ему operandsStack и символ оператора (строка 2.2.1, листинг 1.2);

5. В методе ApplyOperation проверяем соответствие оператора и количество необходимых операндов (строка 1, листинг 1.4). Условие выполняется;

6. Создаем объект прямоугольника с параметрами из стека (строка 1.1, листинг 1.4);

New Rectagle	(	Convert.ToInt32 (operands.Pop())	Convert.ToInt32 (operands.Pop())	Convert.ToInt32 (operands.Pop())	Convert.ToInt32 (operands.Pop())	Convert.ToInt32 (operands.Pop())	)
operands Stack		1	1	8	5	n	
		X	Y	Width	Height	Name	

7. Вызываем метод Draw() для отрисовки фигуры;

8. Добавляем созданную фигуру в контейнер.

9. Так как все прошло успешно, метод ApplyOperation возвращает true методу CalculateRPN, тот в свою очередь тоже возвращает true.

Листинг 1.5. Метод ApplyOperation (частичный)

```
1  private static bool ApplyOperation(Stack<object> operands, char c)
2  {
3      if (c == 'R' && operands.Count == 5)
4      {
5          Rectagle rectagle = new Rectagle((int)operands.Pop(), (int)operands.Pop(), (int)operands.Pop(), (int)operands.Pop(), operands.Pop().ToString());
6          rectagle.Draw();
7          ShapeContainer.AddFigure(rectagle);
8      } else { return false; }
9      return true;
10 }
```

Частичная реализация функции, способной обработать операцию создания и рисования прямоугольника, представлена в листинге 1.5.

Стоит обратить внимание, что данный метод возвращает false, только в случае несоответствия количества операндов к определенному оператору, такие проверки как соответствие типа данных здесь не представлены.

Для обработки нажатия на кнопку ENTER создаем событие `KeyDown` для элемента `textBoxInputString`. Реализация метода обработчика события представлен в листинге 1.6.

Листинг 1.6. – Обработчик нажатия клавиши Enter

```
1  private void textBoxInputString_KeyDown(object sender, KeyEventArgs e)
2  {
3  if(e.KeyCode == Keys.Enter)
4  {
5  try
6  {
7  //выполняется обработка входной строки
8  }
9  catch
10 {
11 //добавляется информация о некорректной команде в историю команд
12 }
13 textBoxInputString.Text = "";
14 }
15 }
```

В строке 3 выполняется проверка условия нажатия клавиши ENTER.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Модифицировать программу, реализованную на предыдущей лабораторной работе «Создание и использование библиотеки классов для графических примитивов». Обновленная версия программы должна включать в себя следующие изменения:

1. Удаление всех элементов управления из формы (кнопок, лейблов, полей для ввода и прочих), кроме поля рисунка `PictureBox`, где будет размещаться битовая карта;
2. Добавление командной строки (для ее реализации можно использовать элемент `TextBox`), где будут указываться команды, которые должна будет выполнять программа (прорисовка, перемещение и удаление фигур);
3. Добавить историю команд, где будут размещаться выполненные и неудачные команды.

Команды должны выполняться при нажатии кнопки ENTER на клавиатуре.

На рисунке 9.1 представлен предполагаемый интерфейс программы для данной лабораторной работы.

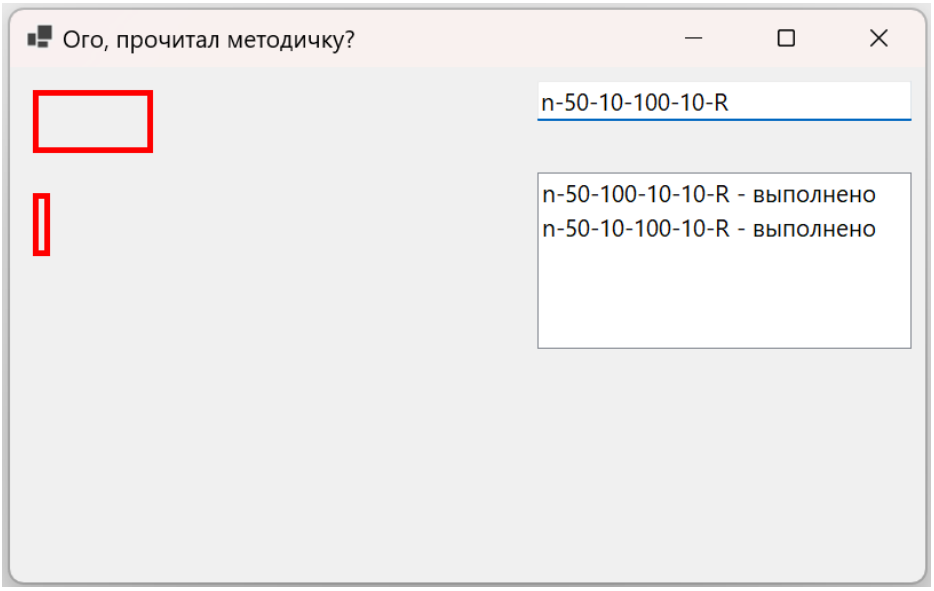

Рисунок 9.1 – Графический интерфейс программы с командной строкой

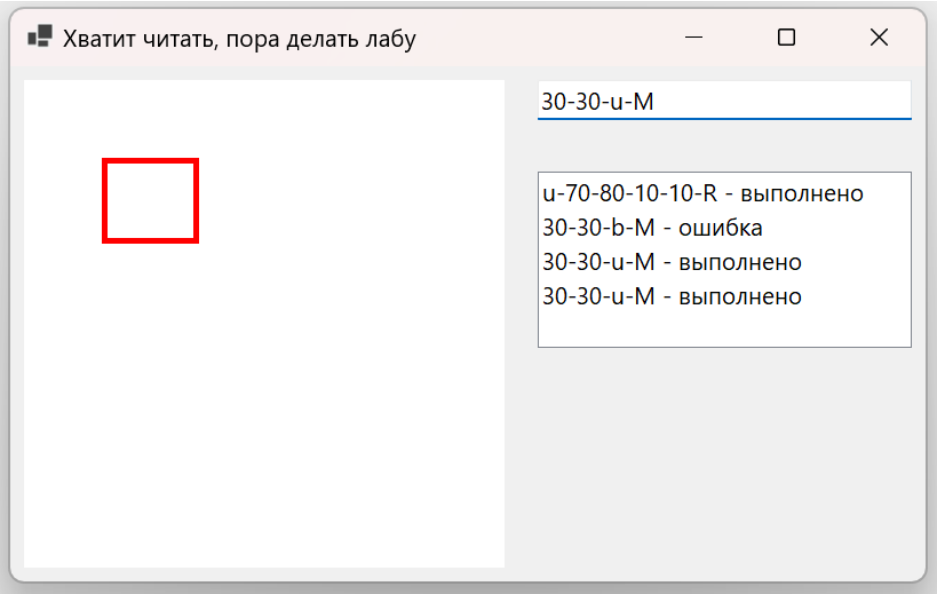

Рисунок 9.2 – Графический интерфейс программы с командной строкой

Список и формат записи команд, выполняемых через командную строку разрабатываемой программы, выбирается согласно варианту индивидуального задания.

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

№	Набор команд	Формат команд
1	Создание прямоугольника	x.y.w.h.name.R
	Перемещение прямоугольника	dx.dy.name.M
	Удаление прямоугольника	name.D
2	Создание квадрата	a,x,y,name,S
	Перемещение квадрата	name,dx,dy,M
	Удаление квадрата	name,D
3	Создание эллипса	y;x;h;w;name;E
	Перемещение эллипса	dx;dy;name;M
	Удаление эллипса	name;D
4	Создание окружности	y*x*w*name*C
	Перемещение окружности	name*d*dy*M
	Удаление окружности	name*D
5	Создание прямоугольника	w-h-x-y-R
	Перемещение прямоугольника	dx-dy-name-M
	Удаление прямоугольника	name-D
6	Создание квадрата	name+x+y+a+S
	Перемещение квадрата	name+dy+dx+M
	Удаление квадрата	name+D
7	Создание эллипса	name!h!w!x!y!E
	Перемещение эллипса	dy!dx!name!M
	Удаление эллипса	name!D
8	Создание окружности	name|w|x|y|C
	Перемещение окружности	name|dy|dx|M
	Удаление окружности	name|D
9	Создание сложной фигуры	x.y.name.w.h.F
	Перемещение сложной фигуры	dy.dx.name.M
	Удаление сложной фигуры	name.D

№	Набор команд	Формат команд
10	Создание сложной фигуры	w h name x y F
	Перемещение сложной фигуры	name dy dx M
	Удаление сложной фигуры	name D
10	Создание прямоугольника	x.y.w.h.name.R
	Перемещение прямоугольника	dx.dy.name.M
	Удаление прямоугольника	name.D
12	Создание квадрата	a,x,y,name,S
	Перемещение квадрата	name,dx,dy,M
	Удаление квадрата	name,D
13	Создание эллипса	y;x;h;w;name;E
	Перемещение эллипса	dx;dy;name;M
	Удаление эллипса	name;D
14	Создание окружности	y*x*w*name*C
	Перемещение окружности	name*d*dy*M
	Удаление окружности	name*D
15	Создание прямоугольника	w-h-x-y-R
	Перемещение прямоугольника	dx-dy-name-M
	Удаление прямоугольника	name-D
16	Создание квадрата	name+x+y+a+S
	Перемещение квадрата	name+dy+dx+M
	Удаление квадрата	name+D
17	Создание эллипса	name!h!w!x!y!E
	Перемещение эллипса	dy!dx!name!M
	Удаление эллипса	name!D
18	Создание окружности	name|w|x|y|C
	Перемещение окружности	name|dy|dx|M
	Удаление окружности	name|D
19	Создание сложной фигуры	x.y.name.w.h.F

№	Набор команд	Формат команд
	Перемещение сложной фигуры	dy.dx.name.M
	Удаление сложной фигуры	name.D
20	Создание сложной фигуры	w h name x y F
	Перемещение сложной фигуры	name dy dx M
	Удаление сложной фигуры	name D
21	Создание прямоугольника	x.y.w.h.name.R
	Перемещение прямоугольника	dx.dy.name.M
	Удаление прямоугольника	name.D
22	Создание квадрата	a,x,y,name,S
	Перемещение квадрата	name,dx,dy,M
	Удаление квадрата	name,D
23	Создание эллипса	y;x;h;w;name;E
	Перемещение эллипса	dx;dy;name;M
	Удаление эллипса	name;D
24	Создание окружности	y*x*w*name*C
	Перемещение окружности	name*d*dy*M
	Удаление окружности	name*D
25	Создание прямоугольника	w-h-x-y-R
	Перемещение прямоугольника	dx-dy-name-M
	Удаление прямоугольника	name-D

Пояснение условных обозначений:

name – имя фигуры

x – координата базовой точки фигуры по оси X

y – координата базовой точки фигуры по оси Y

w – длина фигуры

h – ширина фигуры

a – сторона квадрата

dx – смещение фигуры по оси X

dy – смещение фигуры по оси Y