Лабораторная работа №12

по дисциплине «Процедурное программирование на C»

Обработка линейных односвязных списков

Кострицкий А. С., Ломовской И. В.

Москва — 2019

Цель работы

1. Научиться работать с односвязными линейными списками.

Варианты

1. В виде списка представлены степени и коэффициенты в убывающем порядке полинома с целыми коэффициентами.

   Пример:

   $$\forall x \in \mathbb{R} \quad P(x) = 4x^3 + 2x^2 + 6,$$

   $$\texttt{List} \mapsto \boxed{4,\ 3} \mapsto \boxed{2,\ 2} \mapsto \boxed{6,\ 0} \mapsto \varnothing.$$

   Требуется:

   (a) Реализовать подпрограмму вычисления $P(a)$ по введённому с клавиатуры a.

   (b) Реализовать подпрограмму вычисления производной $\frac{d}{dx}P(x)$.

   (c) Реализовать подпрограмму сложения двух полиномов.

   (d) Реализовать подпрограмму разделения полинома на полиномы чётных и нечётных степеней.

   Интерфейс:

   (a) При старте программы пользователь вводит одно из четырёх слов: val, ddx, sum, dvd. При вводе val за ним с новой строки следуют через пробел в одну строку множители и степени полинома от старшей к младшей, а со следующей строки - аргумент a.

   (b) Выводить полином, сохранённый в виде списка, на экран в виде множителей и степеней через пробел от старшей к младшей. После окончания вывода печатать букву L.

   Пример работы:

```
in>
val
4 2 1 0
7
<out
197
----
in>
ddx
4 2 12 1 1 0
<out
8 1 12 0 L
----
```

```
in>
sum
4 2 12 1 1 0
8 1 12 0
<out
4 2 20 1 13 0 L
----
in>
dvd
4 2 12 1 1 0
<out
4 2 1 0 L
12 1 L
----
```

2. В виде списка представлены коэффициенты в убывающем порядке разложенного по схеме Горнера полинома с целыми коэффициентами.

Пример:

$$\forall x \in \mathbb{R} \quad P(x) = 4x^3 + 2x^2 + 6 = 4x^3 + 2x^2 + 0x + 6 = ((4x + 2) \cdot x + 0) \cdot x + 6,$$

$$\texttt{List} \mapsto \boxed{4} \mapsto \boxed{2} \mapsto \boxed{0} \mapsto \boxed{6} \mapsto \varnothing.$$

Требуется:

(a) Реализовать подпрограмму вычисления $P(a)$ по введённому с клавиатуры a.

(b) Реализовать подпрограмму вычисления производной $\frac{d}{dx}P(x)$.

(c) Реализовать подпрограмму сложения двух полиномов.

(d) Реализовать подпрограмму разделения полинома на полиномы чётных и нечётных степеней.

Интерфейс:

(a) При старте программы пользователь вводит одно из четырёх слов: val, ddx, sum, dvd. При вводе val за ним с новой строки следуют через пробел в одну строку сначала старшая степень полинома, потом — множители полинома от старшей степени к младшей, а со следующей строки - аргумент a.

(b) Выводить полином, сохранённый в виде списка, на экран в виде множителей через пробел от старшей степени к младшей. После окончания вывода печатать букву L.

Пример работы:

```
in>
val
4 4 2 0 6
```

```
7
<out
1476
----
in>
ddx
4 4 2 0 6
<out
12 4 0 L
----
in>
sum
4 4 2 0 6
12 4 0
<out
4 14 4 6 L
----
in>
dvd
4 4 2 0 6
<out
2 6 L
4 0 L
----
```

3. В виде списка представлены степени разложения целого положительного числа n на простые множители.

Пример:

$$n = 1980 = 2^2 * 3^2 * 5 * 11 = 2^2 * 3^2 * 5^1 * 7^0 * 11^1 * 13^0 * 17^0 * 19^0 * \ldots$$

$$\texttt{List} \mapsto \boxed{2} \mapsto \boxed{2} \mapsto \boxed{1} \mapsto \boxed{0} \mapsto \boxed{1} \mapsto \varnothing$$

Требуется:

(a) Реализовать подпрограмму умножения двух таких чисел.

(b) Реализовать подпрограмму возведения числа в квадрат.

(c) Реализовать подпрограмму деления без остатка.

Интерфейс:

(a) При старте программы пользователь вводит одно из четырёх слов: out, mul, sqr, div. При вводе out далее идёт одно целое число, которое нужно вывести на экран в виде списка. В остальных случаях за словом вводится одно или два числа, над которыми требуется выполнить операцию, реализованную в виде соответствующей подпрограммы. Если в результате получается нуль, считать ситуацию ошибочной.

(b) Выводить число, сохранённое в виде списка, на экран в виде степеней множителей в одну строку через пробел от младшего к старшему. После вывода в той же строке печатать букву L. Помните, что при реализации списка в первую очередь стоит задуматься над тем, что можно интерпретировать, как пустой список.

Пример работы:

```
in>
out
24
<out
3 1 L
----
in>
mul
16
3
<out
4 1 L
----
in>
div
4
9
<out
(NO OUT, ERROR)
----
in>
sqr
121
<out
0 0 0 0 4 L
----
```

4. В виде списка представлены множители и степени разложения целого положительного числа n на простые множители.

Пример:

$$n = 1980 = 2^2 * 3^2 * 5 * 11 = 2^2 * 3^2 * 5^1 * 7^0 * 11^1 * 13^0 * 17^0 * 19^0 * \dots$$

сохраняется в виде

$$\texttt{List} \mapsto \boxed{2,\ 2} \mapsto \boxed{3,\ 2} \mapsto \boxed{5,\ 1} \mapsto \boxed{11,\ 1} \mapsto \varnothing$$

Требуется:

(a) Реализовать подпрограмму умножения двух таких чисел.

(b) Реализовать подпрограмму возведения числа в квадрат.

(c) Реализовать подпрограмму деления без остатка.

Интерфейс:

(a) При старте программы пользователь вводит одно из четырёх слов: out, mul, sqr, div. При вводе out далее идёт одно целое число, которое нужно вывести на экран в виде списка. В остальных случаях за словом вводится одно или два числа, над которыми требуется выполнить операцию, реализованную в виде соответствующей подпрограммы. Если в результате получается нуль, считать ситуацию ошибочной.

(b) Выводить число, сохранённое в виде списка, на экран в одну строку через пробел по два числа, от старшего множителя и его степени к младшему. После вывода в той же строке печатать единицу (подумайте, почему).

Пример работы:

```
in>
out
121
<out
11 2 1
----
in>
mul
8
14
<out
7 1 2 4 1
----
in>
div
4
9
<out
(NO OUT, ERROR)
----
in>
sqr
121
<out
11 4 1
----
```

5. В виде списка представлены статические части полудинамической строки. Полудинамические строки используются в приложениях, где все обрабатываемые строки можно условно разделить на несколько множеств строк примерно одного размера, например, если в приложении чаще используются строки из четырёх символов и из восьми символов. Усложнив структуру программы, можно попытаться уменьшить накладные расходы на обработку динамической памяти. Размер статических кусков определяется студентом в разумных пределах. Можно реализовать плотное хранение или неплотное, при реализации неплотного следует дополнительно реализовать функцию compact.

Пример:

$$s = \text{«Please eat eshche etich»}$$

Если используются статические куски по четыре символа, то исходная строка сохраняется в виде

$$\texttt{List} \mapsto \boxed{plea} \mapsto \boxed{se\_e} \mapsto \boxed{at\_e} \mapsto \boxed{shch} \mapsto \boxed{e\_et} \mapsto \boxed{ich\backslash 0} \mapsto \varnothing$$

Обращаем Ваше внимание на то, что статические куски сами в этом случае валидными строками не являются.

Требуется:

(a) Реализовать подпрограмму конкатенации двух таких строк.

(b) Реализовать подпрограмму удаления двойных пробелов.

(c) Реализовать подпрограмму поиска подстроки в такой строке.

Интерфейс:

(a) При старте программы пользователь вводит одно из четырёх слов: out, cat, sps, pos. При вводе out или sps далее с новой строки идёт строка-аргумент, cat - две конкатенируемые строки, pos - строка и искомая подстрока.

Пример работы:

```
in>
out
mama mylas w rame
<out
mama mylas w rame
----
in>
cat
dimitriy zhigalkeen
artyomka
<out
dimitriy zhigalkeenartyomka
----
in>
sps
shla sasha   po shosse    i sosala sooshkoo
<out
shla sasha po shosse i sosala sooshkoo
----
in>
pos
masmashmashas
mashas
<out
7
----
```

6. В виде списка представлены перечисленные по строкам ненулевые элементы целочисленной матрицы M. Обратите внимание, что специально не указываются размеры матрицы — представление в виде списка ненулевых элементов позволяет достраивать матрицу до матрицы любого размера заполнением нулями.

   Пример:

$$M = \begin{pmatrix} 17 & 0 & 0 & 0 & \dots & 0 \\ 0 & 4 & 11 & 0 & \dots & 0 \\ 2 & 3 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix},$$

$$\texttt{List} \mapsto \boxed{0,\ 0,\ 17} \mapsto \boxed{1,\ 1,\ 4} \mapsto \boxed{1,\ 2,\ 11} \mapsto \boxed{2,\ 0,\ 2} \mapsto \boxed{2,\ 1,\ 3} \mapsto \varnothing.$$

   Требуется:

   (a) Реализовать подпрограмму сложения матриц.

   (b) Реализовать подпрограмму умножения матриц.

   (c) Реализовать подпрограмму удаления из матрицы строки с максимальным элементом по матрице.

   Интерфейс:

   (a) При старте программы пользователь вводит одно из четырёх слов: out, add, mul, lin, а далее - одну или две матрицы, в зависимости от команды.

   (b) Матрица вводится по строкам, причём сначала печатаются число строк и число столбцов.

   (c) При выводе матрицы на экран выводить поля элементов списка через пробел при обходе по строкам, как в примере.

   Пример работы:

```
in>
out
2 2
1 0
3 -2
<out
0 0 1 1 0 3 1 1 -2
----
in>
add
2 2
1 0
3 -2
3 3
1 0 1
```

```
0 0 0
0 0 5
<out
0 0 2 0 2 1 1 0 3 1 1 -2 2 2 5
----
in>
mul
2 2
1 0
3 -2
3 3
1 0 1
0 0 0
0 0 5
<out
0 0 1 0 2 1 1 0 3 1 2 3
----
```

Примечания

1. Для каждой структуры данных в первую очередь реализовать набор подпрограмм группы CDIO — подпрограммы создания, удаления, ввода и вывода.

2. Вынести подпрограммы для каждой структуры данных в отдельный модуль.

3. Следовать правилу Тараса Бульбы для памяти: «Если в подпрограмме есть запрос динамической памяти, то либо в ней же должно осуществляться освобождение памяти, либо в имени подпрограммы должно быть *указание* для программиста на наличие запроса памяти внутри подпрограммы.» В качестве *указаний*, например, можно использовать слова и словосочетания: «allocate», «create», «set length», «new» и другие.

Взаимодействие с системой тестирования

1. Решение задачи оформляется студентом в виде многофайлового проекта. Для сборки проекта используется программа make, сценарий сборки помещается под версионный контроль. В сценарии должны присутствовать цель `app.exe` для сборки основной программы и цель `test.exe` — для сборки модульных тестов.

2. Исходный код лабораторной работы размещается студентом в ветви `lab_LL`, а решение каждой из задач — в отдельной папке с названием вида `lab_LL_CC_PP`, где LL — номер лабораторной, CC — вариант студента, PP — номер задачи.

   Пример: решения восьми задач седьмого варианта пятой лабораторной размещаются в папках `lab_05_07_01`, `lab_05_07_02`, `lab_05_07_03`, ..., `lab_05_07_08`.

3. Исходный код должен соответствовать оглашённым в начале семестра правилам оформления.

4. Если для решения задачи студентом создаётся отдельный проект в IDE, разрешается поместить под версионный контроль файлы проекта, добавив перед этим необходимые маски в список игнорирования. Старайтесь добавлять маски общего вида. Для каждого проекта должны быть созданы, как минимум, два варианта сборки: `Debug` — с отладочной информацией, и `Release` — без отладочной информации. Крайне рекомендуем использовать IDE Qt Creator.

5. Для каждой программы ещё до реализации студентом заготавливаются и помещаются под версионный контроль функциональные тесты, демонстрирующие её работоспособность. Входные данные следует располагать в файлах вида `in_TT.txt`, выходные — в файлах вида `out_TT.txt`, где TT — номер тестового случая.

   Под версионный контроль также помещается файл вида `FuncTestsDesc.md` с описанием в свободной форме содержимого каждого из тестов. Вёрстка файла на языке Markdown обязательной при этом не является, достаточно обычного текста.

   Разрешается помещать под версионный контроль сценарии автоматического прогона функциональных тестов.

   Если Вы используете при автоматическом прогоне функциональных тестов сравнение строк, не забудьте проверить используемые кодировки. Помните, что UTF-8 и UTF-8(BOM) — две разные кодировки.

   Пример: функциональные тесты для задачи с двенадцатью классами эквивалентности должны размещаться в файлах `in_01.txt`, `in_02.txt`, ..., `in_12.txt`, `out_01.txt`, `out_02.txt`, ..., `out_12.txt`. В файле `FuncTestsDesc.md` при этом может содержаться следующая информация:

```
# Тесты для лабораторной работы №X
## Входные данные
int a, int b, int c
## Выходные данные
int d, int e
- in_01 -- негативный -- вместо числа a вводится символ
- in_02 -- негативный -- вместо числа b вводится символ
- in_03 -- негативный -- недостаточно аргументов вводятся с консоли
- in_04 -- позитивный -- обычный тест
- in_05 -- позитивный -- вводятся три одинаковых числа
...
```

6. Для каждой подпрограммы должны быть подготовлены модульные тесты, которые демонстрируют её работоспособность.

7. Все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены к моменту выхода из программы. Для контроля можно использовать, например, программы valgrind или Dr. Memory.

8. Успешность ввода должна контролироваться. При первом неверном вводе программа должна возвращать код ошибки. Обратите внимание, что даже в этом случае все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены.

9. Вывод Вашей программы может содержать текстовые сообщения и числа. Тестовая система анализирует только числа в потоке вывода, поэтому они могут быть использованы только для вывода результатов — использовать числа в информационных сообщениях запрещено.

   Пример: сообщение «Input point 1:» будет неверно воспринято тестовой системой, а сообщения «Input point A:» или «Input first point:» — правильно.

10. Если не указано обратное, числа двойной точности следует выводить, округляя до шестого знака после запятой.