Лабораторная работа №10

по дисциплине «Программирование на Си»

Обработка списков и абстрактных типов данных

Кострицкий А. С., Ломовской И. В.

Москва — 2023 — TS2310291627

Содержание

Цель работы

Цель работы — приобрести навыки работы со списками.

Студенты должны получить и закрепить на практике следующие знания и умения:

1. Работа с динамической памятью.
2. Работа с бестиповыми указателями и указателями на функцию.
3. Обработка линейного односвязного списка (добавление элемента, удаление элемента, поиск элемента, комбинированные действия).
4. Обработка текстовых файлов.
5. Организация корректной работы с ресурсами (динамически выделенная память, файловые описатели).
6. Использование в программе аргументов командной строки.
7. Контроль правильности работы с динамической памятью с помощью специального ПО.

Задача №1

Общее задание

Реализовать линейный односвязный список с непрямым владением:

1. Информационная часть узла представлена указателем на void.

```
typedef struct node node_t;

struct node
{
    void *data;
    node_t *next;
};
```

2. Список формируется на основе данных, которые читаются из текстового файла. В качестве данных может, например, выступать информация о студенте: фамилия и год рождения. Область данных выбирается студентом самостоятельно и должна отличаться от указанного примера.

3. Элементы списка данными ***не владеют***, т. е. хранят лишь указатель на них. При удалении элемента из списка данные из хранилища не удаляются.

4. Для решения некоторых задач требуется функция сравнения элементов. Сравнение элементов реализуется как отдельная функция. Функции, нуждающиеся в сравнении элементов, получают в качестве параметра указатель на компаратор.

5. Имена файлов, выполняемая операция и т. п. указываются через параметры командной строки, результаты работы записываются в файл.

6. В качестве одного из необходимых результатов работы должно быть приведено условие осмысленной задачи в выбранной предметной области, которая может быть решена с использованием разработанных вами функций (см. индивидуальное задание) и для решения которой достаточно того объёма данных, которые считываются из исходного файла. Также должна быть написана программа, решающая поставленную задачу.

Индивидуальное задание

Индивидуальные задачи выбираются студентом самостоятельно. При желании преподаватель может распределить их по вариантам. Относительная сложность каждой задачи указана в скобках после условия.

Задачи на работу с одним элементом списка

Необходимо решить любые две задачи.

1. Напишите функцию поиска элемента в списке (нужна функция сравнения элементов). *(Относительная сложность: 1.)*

```
node_t* find(node_t *head, const void *data,
    int (*comparator)(const void*, const void *));
```

2. Напишите функцию pop_front, которая возвращает указатель на данные из элемента, который расположен в «голове» списка. При этом из списка сам элемент удаляется. *(Относительная сложность: 1.)*

```
void* pop_front(node_t **head);
```

3. Напишите функцию pop_end, которая возвращает указатель на данные из элемента, который расположен в «хвосте» списка. При этом из списка сам элемент удаляется. *(Относительная сложность: 1.)*

```
void* pop_back(node_t **head);
```

4. Напишите функцию insert, которая вставляет элемент перед указанным элементом списка (в качестве параметров указываются адреса обоих элементов). *(Относительная сложность: 2.)*

```
void insert(node_t **head, node_t *elem, node_t *before);
```

Задачи на работу с целым списком

Необходимо решить одну любую задачу.

1. Напишите функцию copy, которая по указанному списку создаёт его копию (данные при этом не копируются). *(Относительная сложность: 1.)*

```
int copy(node_t *head, node_t **new_head);
// функция возвращает код ошибки, потому что она выделяет память
```

2. Напишите функцию append, которая получает два списка a и b, добавляет список b в конец a. Список b при этом оказывается пустым. *(Относительная сложность: 1.)*

```
void append(node_t **head_a, node_t **head_b);
```

3. Напишите функцию remove_duplicates, которая получает упорядоченный список и оставляет в нем лишь первые вхождения каждого элемента. Совпадение определяется с помощью функции сравнения элементов. При удалении элемента списка данные не удаляются. *(Относительная сложность: 2.)*

```
void remove_duplicates(node_t **head,
int (*comparator)(const void*, const void*));
```

4. Напишите функцию reverse, которая обращает список. Идеи реализации:
   (a) Использование pop_front и двух списков. *(Относительная сложность: 1.)*
   (b) Использование 3-х указателей на соседние элементы списка. *(Относительная сложность: 2.)*
   (c) Рекурсия. *(Относительная сложность: 2.)*

```
node_t* reverse(node_t *head);
// возвращается «новая» голова
```

Сортировка списка

```
node_t* sort(node_t *head, int (*comparator)(const void *, const void *));
// возвращается «новая» голова
```

Необходимо реализовать одну из двух сортировок.

1. Сортировка вставками

   Напишите функцию sorted_insert, которая получает упорядоченный список, и элемент, который нужно вставить в этот список, чтобы не нарушить его упорядоченности.

```
void sorted_insert(node_t **head, node_t *element,
int (*comparator)(const void *, const void *));
```

   Напишите функцию sort, которая получает список и упорядочивает его по возрастанию, используя функцию sorted_insert. *(Относительная сложность: 3.)*

2. Сортировка слиянием

   Напишите функцию front_back_split, которая получает список и делит его на две половины. Если в списке нечетное число элементов, "серединный"элемент должен попасть в первую половину.

```
void front_back_split(node_t* head, node_t** back);
```

   Напишите функцию sorted_merge, которая получает два упорядоченных списка и объединяет их в один.

```
node_t* sorted_merge(node_t **head_a, node_t **head_b,
int (*comparator)(const void *, const void *));
// Списки становятся пустыми, элементы из них
// «переходят» в упорядоченный
```

Используя функции front_back_split и sorted_merge напишите функцию sort, которая реализует рекурсивный алгоритм сортировки слиянием. *(Относительная сложность: 4.)*

Задача №2

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.

   Правила взаимодействия:

   (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.

Правила взаимодействия:

(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 количество слагаемых 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 букв «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
3 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.

Правила взаимодействия:

(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:

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

   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.

   Правила взаимодействия:

   (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[1] строки[2]. 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.

[1] 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.

[2] Эту структуру данных, не ограничиваясь одними строками, обычно называют *развёрнутым списком*.

Правила взаимодействия:

(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.

Правила взаимодействия:

(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. Хотя для решения второй задачи достаточно реализовать линейный односвязный список, можно выбрать иной вид списка по согласованию с преподавателем.

Задача №3

Реализовать абстрактный тип данных «ассоциативный массив» согласно интерфейсу в заголовочном файле `associative_array.h`. Необходимо предоставить две различные реализации одного и того же интерфейса.

Для этой задачи требуются только модульные тесты. Модульные тесты реализуются студентом в одном экземпляре на обе реализации.

Файловая структура проекта:

```
# содержит описание интерфейса и модульных тестов
/lab_10_03_common
     associative_array.h
     check_main.c
     check_func_1.h
     check_func_1.c
...
# содержит первую реализацию интерфейса и makefile для сборки
/lab_10_03_01
     associative_array_impl_1.c
     вспомогательные файлы, которые нужны для associative_array_impl_1.c
     символьные ссылки на файлы из lab_10_3_common
     makefile
# содержит вторую реализацию интерфейса и makefile для сборки
/lab_10_03_02
     associative_array_impl_2.c
     вспомогательные файлы, которые нужны для associative_array_impl_2.c
     символьные ссылки на файлы из lab_10_3_common
     makefile
```

Взаимодействие с системой тестирования

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

2. В сценарии сборки рекомендуется обозначить, помимо прочих, следующие цели:

   (a) `unit` – сборка и прогон модульных тестов.

   (b) `func` – прогон функциональных тестов.

   (c) `clean` – очистка генерируемых файлов.

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

   Пример: решения восьми задач седьмого варианта пятой лабораторной размещаются в папках `lab_05_01_07`, `lab_05_02_07`, `lab_05_03_07`, ..., `lab_05_08_07`.

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

5. 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.

6. Для каждой программы ещё до реализации студентом заготавливаются и помещаются под версионный контроль в подпапку `func_tests` функциональные тесты, демонстрирующие её работоспособность.

   Позитивные входные данные следует располагать в файлах вида `pos_TT_in.txt`, выходные — в файлах вида `pos_TT_out.txt`, аргументы командной строки при наличии — в файлах вида `pos_TT_args.txt`, где TT — номер тестового случая.

   Негативные входные данные следует располагать в файлах вида `neg_TT_in.txt`, выходные — в файлах вида `neg_TT_out.txt`, аргументы командной строки при наличии — в файлах вида `neg_TT_args.txt`, где TT — номер тестового случая.

   Разрешается помещать под версионный контроль в подпапку `func_tests` 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.

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

   Пример: восемь позитивных и шесть негативных функциональных тестов без дополнительных ключей командной строки должны размещаться в файлах `pos_01_in.txt`, `pos_01_out.txt`, ..., `neg_06_out.txt`. В файле `readme.md` при этом может содержаться следующая информация:

```
# Тесты для лабораторной работы №LL

## Входные данные
Целые a, b, c

## Выходные данные
Целые d, e

## Позитивные тесты:
- 01 - обычный тест;
- 02 - в качестве первого числа нуль;
...
- 08 - все три числа равны.

## Негативные тесты:
- 01 - вместо первого числа идёт буква;
- 02 - вместо второго числа идёт буква;
...
- 06 - вводятся слишком большие числа.
```

7. Рекомендуется задавать следующую структуру проекта:
   (a) Все файлы исходных кодов хранятся в подпапке `src`.
   (b) Все файлы заголовков хранятся в подпапке `inc`.
   (c) Для каждого модуля создаётся и помещается в подпапку `unit_tests` один файл с модульными тестами, имя которого повторяет имя модуля с префиксом «`check_`». Основная программа модульного тестирования носит название «`check_main.c`».
   (d) Функциональные тесты оформляются в соответствие с предыдущими пунктами.
   (e) Сценарий сборки и конечные приложения генерируются в корне проекта.
   (f) Все остальные генерируемые файлы, в том числе объектные файлы и файлы статистики gcov, создаются в подпапке `out`.

Пример: папка с проектом для лабораторной работы, состоящего из текста программы и двух модулей, будет иметь следующий вид:

```
/lab_LL_CC_PP/
    app.exe
    makefile
    unit_tests.exe
    /inc/
        unit_a.h
        unit_b.h
    /out/
        main.o
        unit_a.o
        unit_b.o
    /src/
        main.c
        unit_a.c
        unit_b.c
    /func_tests/
      ...
    /unit_tests/
        check_main.c
        check_unit_a.c
        check_unit_b.c
```

8. Для каждой подпрограммы должны быть подготовлены модульные тесты с помощью фреймворка check, которые демонстрируют её работоспособность.

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

10. Успешность ввода должна контролироваться. При первом неверном вводе программа должна прекращать работу с ненулевым кодом возврата.

    Обратите внимание, что даже в этом случае все динамические ресурсы, которые уже были Вами успешно запрошены, должны быть высвобождены.

11. 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.

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

Памятка преподавателя

1. *Только для ЛР№10.* Совпадение структур и типов данных у студента и в задании не проверяется тестовой системой.