**Стек, очередь, дек**

***Цель работы***

Научиться работать со структурами данных – стек ,очередь, дек, реализованными с помощью массивов и связных списков.

***Общие сведения***

Стек. Структура данных типа стек применяется очень часто, например, при распознавании синтаксиса в компиляторах, для передачи параметров функциям, при выполнении вызова функции и возвращения из нее.



Рисунок 2.3 – Структура данных типа стек

Стек (stack) это линейная структура данных переменного типа. Позволяет включать и исключать элементы только c одного конца, называемого **вершина** (top) **стека**. В этом случае первым будет выбран элемент, который был записан в стек последним. Информация в этой структуре данных обрабатывается по принципу: «последним пришел, первым ушел». Как говорят, поддерживает дисциплину LIFO (Last In, First Out) . Структура данных стек является структурой с ограниченным доступом, так как доступ разрешается только к элементу, находящемуся в вершине стека. Этот элемент называется **текущим**. Заметим, что под элементом понимается запись. Информацию о позиции текущей записи стека хранит **указатель вершины стека**. Для реализации стека можно использовать последовательное и связное представление. В настоящем разделе рассмотрим последовательное представление.

Для работы со стеком необходим дескриптор содержащий:

– имя стека;

– адрес верхней границы стека;

– указатель на свободный;

– адрес нижней границы;

– описание элемента (длина записи).

Когда стек пуст, указатель указывает на нижнюю границу стека.



Рисунок 2.4 – Пример стека с тремя записями.

Основные операции над стеком:

– создать стек;

– включить запись в стек;

– исключить запись из стека;

– очистить стек;

– определить объем стека (количество записей).

1. Создание стека. При создании последовательного представления необходимо знать предельный размер стека. Под предполагаемый предельный размер резервируется блок памяти. Предельный размер рассчитывается исходя из знания о длине записи *R* и максимально возможного количества записей в стеке *n* .

2. Включение записи в стек. При включении записи в стек необходимо проверить: есть ли место для записи (указатель стека указывает на верхнюю границу стека ). Если нет, то выдать сообщение «переполнение стека», если место есть, то указатель стека переместить вверх на одну запись и записать данные.

3. Исключить запись из стека. Проверить: есть ли в стеке информация (указатель стека равен нижней границе?). Если нет, то выйти. Можно выдать сообщение: «Стек пуст». Если есть, то считать данные и переместить указатель на одну запись вниз.

4. Очистить стек. В указатель стека помещается значение нижней границы стека.

5. Проверка объема стека сводится к определению количества записей в стеке.

Стековые структуры широко применяются в трансляторах, при реализации всевозможных подпрограмм и многоуровневых прерываний, а также при решении задач, алгоритмы которых лучше всего описываются рекурсивными методами.

Очередь (буфер). Очередь – это линейная структура переменного размера. Исключение из очереди допускается с одного конца – с начала очереди. Включение элементов возможно лишь с противоположного конца. Данные в такой структуре обрабатываются в порядке их поступления по принципу: «первым пришел, первым ушел». Очередь это структура типа FIFO (First In, First Out). Данная структура часто применяется в технике, при считывании информации с измерительных приборов. Запись в такой буфер производится по прерыванию, а считывание – когда есть время для обработки поступившей информации.



Рисунок 2.5 – структура данных типа очередь

В этом случае, по мере наполнения и освобождения очереди, она будет «ползти» по оперативной памяти. Для того, чтобы этого не случилось, очередь «закольцовывают», т.е. применяют кольцевой буфер. Кольцевой буфер часто используется в измерительных комплексах, когда каждое регистрируемое событие заносится и обрабатывается отдельно. Кольцевой буфер представляет собой область оперативной памяти, куда информация заносится программой обработки прерываний в темпе ее поступления, а считывание основной программой в темпе, диктуемом скоростью обработки этой информации. Средняя скорость поступления не должна превышать средней скорости обработки данных, иначе буфер быстро заполнится, и данные начнут теряться.

Операции со структурой данных типа кольцевая очередь (кольцевой буфер):

– создать структуру типа буфер;

– очистить буфер (сделать пустым);

– найти первый элемент буфера;

– вставить элемент в конец буфера;

– удалить первый элемент из буфера;

– определить, пуст ли буфер.

Для работы с буфером создается дескриптор включающий:

– имя очереди (буфера);

– адрес начала буфера;

– адрес конца буфера;

– указатель для записи Put;

– указатель для считывания Get;

– счетчик записей ( начальное значение которого равно нулю);

– максимальное количество записей в очереди (буфере).

При инициализации буфера необходимо, прежде всего, определить размер записи, поскольку размер буфера кратен размеру записи. Указатели *Get* и *Put* устанавливаются равными адресу начала буфера. Счетчик записей *n* устанавливается равным нулю.



Рисунок 2.6 – Начальное состояние буфера



а б в

Рисунок 2.7 - Пример заполнения буфера

На рисунке а показано начало заполнения буфера. В ходе работы с буфером возможна ситуация, когда указатель Put указывает на конец буфера рисунок б. При дальнейшем заполнении, указатель для записи перемещается на начало буфера и производится запись рисунок в.

Алгоритм для записи информации в буфер. В начале необходимо проверить, полон ли буфер (счетчик записей в этом случае равен максимально возможному числу записей). Если буфер полон, необходимо выдать сообщение «переполнение буфера». Если в буфере есть место для записи, необходимо проверить значение указателя Put. Если он указывает на конец буфера, то ему необходимо присвоить значение адреса начала буфера. Далее, для любой ситуации записать данные в буфер, переместить значение указателя на следующий свободный для записи адрес, увеличить значение счетчика записей на единицу.

Алгоритм для считывания. На первом шаге алгоритма проверяется значение счетчика записей. Если он пуст, то в буфере нет ни одной записи. Поэтому необходимо выдать сообщение, что буфер пуст, и выйти из процедуры считывания информации из буфера. Если в буфере есть информации, необходимо проверить на какой адрес указывает указатель Get . Если указывает на конец буфера, то в этом случае указателю Get необходимо присвоить значение начала буфера. Далее, в любом случае считываются данные и переносятся в другой буфер для последующей обработки, значение указателя перемещается на начало следующей записи и счетчик записей уменьшается на единицу.

Пример кольцевой очереди дает такая структура данных, как буфер BIOS для записи кодов клавиш (или, просто, буфер клавиатуры). При нажатии большинства клавиш клавиатуры компьютера в буфер клавиатуры помещается 2-х байтовый код нажатой клавиши, называемый ее BIOS-кодом. Младший байт этого кода равен ASCII-коду символа, либо нулю. Старший байт содержит так называемый СКЭН-код, представляющий собой, грубо говоря, порядковый номер нажатой клавиши. Буфер клавиатуры занимает 32 байта памяти: он рассчитан на 15 нажатий клавиш, генерирующих 2-хбай-товые коды, и поэтому имеет 30 байт для кодов клавиш и еще два байта, которые резервируются под двухбайтовый код для клавиши ENTER.

Дек. Дек – линейная динамическая структура данных, представляет собой очередь с двумя концами.



Рисунок 2.8 – Дек

Частными случаями является дек с ограниченным входом и дек с ограниченным выходом.

Дек можно обрабатывать как очередь или как стек.

В дескриптор дека входят:

– имя дека;

– адрес верхней границы;

– адрес нижней границы;

– указатель нижний;

– указатель верхний;

– количество записей;

– общее количество записей.

В начале работы, когда дек пуст, устанавливают равенство нижнего и верхнего указателей.



Рисунок 2.9 – Дек, значения при инициализации

Основные операции над деком.

– создать структуру данных типа дек;

– очистить дек;

– определить, пусть ли дек?

– добавить значение в начало (конец) дека;

– взять элемент из начала (конца) дека;

– найти начало (конец) дека;

– закончить работу.

## 2.5. Списки

Линейным списком называется упорядоченная последовательность данных *х*1, *х*2,...,*хn* , *n* ≥ 0. При этом каждый элемент списка содержит одинаковое количество полей. Количество элементов списка называется его **длиной**. Структурные свойства списка ограничиваются линейным расположением элементов

*x*1...*хk*, *xk*+1... *xn* , где 1 < *k* < *k*+1 < *n.*

Поскольку списки представляют собой упорядоченную последовательность, надо определить первый элемент списка, который чаще всего называют головой списка. Затем второй элемент и т.д. списки редко имеют фиксированную длину. Обычно, их длина меняется в ходе выполнения программы.

Упорядоченность элементов списка может задаваться неявно путем их последовательного расположения, как в логической структуре, так и в памяти ЭВМ. Такой список **называется последовательным**.

Второй способ задания упорядоченности – применение указателей, расположенных в элементах и дающих возможность для любого элемента определить его предшественника или последователя. Такие списки называются **связными**. Если каждая запись содержит лишь один указатель, то список **односвязный**, при большем числе указателей список **многосвязный**.

Списки могут быть: линейными однонаправленными (односвязными), линейными двунаправленными, многосвязными, нелинейными многосвязными.

На основе списков можно организовать такие структуры данных как: стек, очередь, дек, дерево, граф.

Если количество записей постоянно, то линейный список сводится к массиву, записи или таблице.

### 2.4.1 Линейный однонаправленный список

Для динамического представления списков каждый элемент списка должен содержать кроме смысловой информации (поле данных) еще и поле типа указатель. Кроме того, должен быть указатель, который указывает на начало линейного списка. Элемент списковой структуры приведен на рисунке 2.10.



Рисунок 2.10 - Элемент списка

Структура данных типа линейный однонаправленный список приведена на рисунке 2.11.



Рисунок 2.11 - Линейный однонаправленный список

Операции над структурой данных типа список:

– создание первого элемента списка;

– вставка элемента: в начало списка, в конец списка, после элемента с заданным ключом, до элемента с заданным ключом;

– чтение элемента с заданным ключом;

– удаление элемента с заданным ключом;

– упорядочивание списка по ключу.

В дескриптор линейного списка входит:

– имя списка (идентификатор);

– тип элементов списка;

– указатель начала списка;

– указатель текущего элемента списка.

Описание простейшего узла представлено ниже:

struct Node {

int d; // информационная часть, тип данных

// может быть любым

Node \*p; // указатель на следующий, такого же

// типа

};

Создание узла. Для этого необходимо создать новый тип данных Node. И создать указатель на первый элемент списка.

Node \*pn = new Node; // создание элемента

// pn указатель на первый

pn -> d = 5; // поле данных равно 5

pn -> p = null; // ссылка на следующий эле-

// мент равна нулю

Вставка элемента в начало списка. Для этого необходимо выполнить следующие действия:

- создать новый элемент (рисунок 2.12);

- заполнить информационное поле;

- в указатель на следующий элемент занести значение указателя на начало списка (рисунок 2.13);

- указателю на начало списка присвоить значение указателя на новый элемент (рисунок 2.14).



Рисунок 2.12 - Создание нового элемента с указателем q.



Рисунок 2.13 - Значение указателя на начало переместить на новый элемент



Рисунок 2.14 – Результат вставки элемента

Node \*q = new Node;

q->d = d1;

q->p = Pn;

Pn = q;

Вставка элемента в конец списка. Для вставки элемента, необходимо знать указатель на последний элемент списка. Последовательность действий (рисунки 2.15, 2.16):

- создать новый элемент;

- занести в него информацию;

- в указатель нового элемента занести значение null;

- в последнем элементе списка, в указатель на следующий элемент занести значение указателя нового элемента.

Node \*q = new Node;

q->d = d2;

q->p = null;

Pn ->p = q;



Рисунок 2.15 - Вставка элемента в конец списка



Рисунок 2.16 Результат вставки элемента в конец списка

Вставка элемента в список после элемента с заданным ключом. Последовательность действий (рисунок 2.17):

- создать элемент;

- занести значение в информационное поле;

- занести значение указателя находящегося в поле элемента с заданным ключом в поле указателя на следующий элемент нового элемента;

- занести значение указателя на новый элемент в поле указателя элемента с заданным ключом.



Рисунок 2.17 - Вставка нового элемента после элемента с заданным ключом

Node \*q = new Node;

q->d = d3;

q->p = Pn->p;

Pn->p = q;

Последовательность действий при вставке элемента перед элементом с выбранным ключом (рисунок 2.18).

- создать элемент;

- в созданный элемент занести значение элемента, перед которым надо вставить новый элемент.

- в элемент, перед которым надо вставить новый элемент, занести значение в информационное поле;

- перенести значение указателя на следующий элемент из поля Pn->p в поле q->p.

- в поле Pn->p занести значение q.



Рисунок 2.18 - В ставка элемента после перед элементом с заданным ключом

Node \*q = new Node;

q->d = d;

Pn-> = d3;

q->p = Pn->p;

Pn->p = q;



Рисунок 2.19 – Результат вставки

Удаление элемента из списка. Для удаления элемента необходимо иметь два указателя на удаляемый элемент Pn и q. Последовательность действий при удалении элемента из начала списка.

- забираем информацию из элемента;

- указателю Pn присваиваем значение указателя на следующий элемент;

- удаляем из памяти элемент, на который указывает указатель q.

int data;

q = Pn;

data = Pn->d;

Pn= Pn->p;

delete q;

При удалении элемента из конца списка, возникает проблема, связанная с тем, что всегда необходимо помнить указатель на предыдущий элемент. Что бы его запомнить приходится каждый раз сканировать весь список. Поэтому чаще всего, элементы из конца списка не удаляют. Для устранения данного недостатка, используют линейные двунаправленные списки.

Удаление элемента после элемента с заданным значением ключа.



Рисунок 2.20 – Удаление элемента из списка



Рисунок 2.21 – Результат удаления

Реализация стека с помощью односвязного списка представлена на рисунке 2.22. Вставка и удаление элементов производится из вершины стека. Алгоритмы обработки приведены выше.



Рисунок 2.22 – Пример стека, реализованного с помощью односвязного списка

Реализация очереди с помощью односвязного списка приведена на рисунке 2.23. Добавление элементов происходит с конца очереди, а удаление – с начала очереди.



Рисунок 2.23 - Очередь

Кольцевой буфер. При реализации, линейный список закольцовывают, т.е. вместо значения null записывают ссылку на первый элемент списка (рисунок 2.24).



Рисунок 2.24 - Кольцевой буфер

### 2.4.2 Двунаправленный список

В ряде задач необходимо иметь возможность продвижения по связному списку в обоих направлениях. Для этого в каждый элемент вводится дополнительный указатель, задающий продвижение по списку в обратном направлении (рисунок 2.25). Такой список называется **двунаправленным**.



Рисунок 2.25 – Элемент двунаправленного списка



Рисунок 2.26 – Двунаправленный список

В записи присутствуют два указателя. Один указывает на следующий элемент, другой на предыдущий. Кроме того, необходимо два указателя для перемещения по списку: указатель начала и указатель конца. В процессе добавления или удаления элементов двусвязного списка происходит изменение прямых и обратных указателей (рисунки 2.25, 22.6)

В дескриптор двунаправленного списка входит:

– имя списка (идентификатор);

– тип элементов списка;

– указатель начала списка;

– указатель конца списка;

– указатель текущего элемента списка.

При использовании двунаправленных списков ускоряются процессы поиски в информационном массиве, но увеличивается расход памяти под указатели.

**Задание**

Язык программирования – любой.

Работа состоит из общей части и индивидуального задания

1. **Общее задание.**

Создать три класса, соответственно для: стека, очереди, дека.

Желательно при этом, чтобы структуры данных были ДИНАМИЧЕСКИМИ, то есть ограниченны только объемом свободной оперативной памяти. Для этого используйте указатели и динамически распределяемую память.

* 1. **Очередь**

Разработайте программу, реализующую одностороннюю очередь с использованием шаблонного класса. Программа содержит определение шаблонного класса для работы с односторонней очередью.

В шаблоне классов реализованы операции с очередью:

-инициализация очереди (конструктор);

- разрушение очереди с освобождением занятой динамической памяти (деструктор);

- занесение элемента в очередь;

- извлечение элемента из очереди;

- печать состояния очереди;

* 1. **Стек**

Разработайте программу, реализующую динамический стек неограниченного размера с использованием шаблонного класса. Программа содержит определение структуры для элемента стека и определение шаблонного класса для работы с динамическим стеком неограниченного размера.

В шаблонном классе реализованы следующие операции со стеком:

-инициализация стека (конструктор);

- разрушение стека с освобождением занятой динамической памяти (деструктор);

- занесение элемента в стек;

- извлечение элемента из стека;

- печать состояния стека.

* 1. **Дек (универсальная двухсторонняя очередь)**

Разработайте программу, реализующую дек с использованием шаблонного класса. Программа содержит определение шаблонного класса для работы с универсальной очередью.

В шаблоне классов реализованы операции с очередью:

-инициализация очереди (конструктор);

- разрушение очереди с освобождением занятой динамической памяти (деструктор);

- занесение элемента с левого конца;

- занесение элемента с правого конца;

- извлечение элемента с левого конца;

- извлечение элемента с правого конца;

- печать состояния очереди с использованием указателя на левый конец очереди;

- печать состояния очереди с использованием указателя на правый конец очереди.

1. **Индивидуальное задание**

2.1 Используя разработанные классы решить задачу, соответствующую варианту.

|  |  |
| --- | --- |
| ФИО | Вариант |
| Бедарева Анастасия | 1 |
| Белецкй Евгений | 2 |
| Вахитов Артем | 3 |
| Джаналиев Марат | 4 |
| Дятлова Полина | 5 |
| Загреба Максим | 6 |
| Иванов Георгий | 7 |
| Ипатова Ксения | 8 |
| Канунников **А**ртур | 9 |
| Канунникова Виктория | 10 |
| Климов Вячеслав | 11 |
| Колесникова Анна | 12 |
| Кузнецов Владислав | 13 |
| Кузнецов Дмитрий | 14 |
| Кузнецова Ирина | 15 |
| Куртукова Анна | 16 |
| Л**о**мовцева Ирина | 17 |
| Мариненко Сергей | 18 |
| Мащенко Вадим | 19 |
| Мельник Дмитрий | 20 |
| Никишаев Егор | 21 |
| Реуцких Виктория | 22 |
| Фельчук Максим | 23 |
| **Чубаров Артур** | **24** |
| Сенькова Анастасия | 25 |

2.2 Залить все программы в новый проект на свой аккаунт github.

2.3 Написать отчет и защитить у преподавателя.

**Варианты заданий**

**1** Отсортировать строки файла, содержащие названий книг, в алфавитном порядке с использованием двух***деков*.**

**2 *Дек***содержит последовательность символов для шифровки сообщений. Дан текстовый файл, содержащий зашифрованное сообщение. Пользуясь ***деком****,*расшифровать текст. Известно, что при шифровке каждый символ сообщения заменялся следующим за ним в***деке***по часовой стрелке через один.

**3 *Дек***содержит последовательность символов для шифровки сообщений. Дан текстовый файл, содержащий сообщение. Пользуясь***деком*,**зашифровать текст, заменяя каждый символ сообщения следующим за ним в ***деке***против часовой стрелки через один.

**4** Написать программу, моделирующую железнодорожный сортировочный узел. Исходный файл содержит информацию об имеющихся вагонах двух типов, при этом количество вагонов обоих типов одинаково. Последовательность элементов файла неупорядочена, в каждом элементе файла: тип вагона и идентификационный номер вагона. Используя ***стек***(“тупик”), за один просмотр исходного файла сформировать новый файл (“состав вагонов”), в котором типы вагонов чередуются.

**5** Даны три стержня и*n* дисков различного размера. Диски можно надевать на стержни, образуя из них башни. Перенести*n*дисков со стержня*А*на стержень*С***,**сохранив их первоначальный порядок. При переносе дисков необходимо соблюдать следующие правила:

на каждом шаге со стержня на стержень переносить только один диск;

диск нельзя помещать на диск меньшего размера;

для промежуточного хранения можно использовать стержень*В***.**

Реализовать алгоритм, используя три ***стека***вместо стержней*А***,***В***,***С***.**Информация о дисках хранится в исходном файле.

**6** Дан файл из вещественных чисел. Используя ***очередь*,**за один просмотр файла напечатать сначала все числа, меньшие *a***,**затем все числа из интервала*[a,b]***,**и, наконец, все остальные числа, сохраняя исходный порядок в каждой группе.

**7** Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс круглых скобок в тексте, используя***стек*.**

**8** Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс круглых скобок в тексте, используя ***очередь*.**

**9** Дан текстовый файл. Используя***очередь*,**переписать содержимое его строк в новый текстовый файл, перенося при этом в конец каждой строки все входящие в нее цифры, сохраняя исходный порядок следования среди цифр и среди остальных символов строки.

**10** Дан файл из символов. Используя***очередь*,**за один просмотр файла напечатать сначала все цифры, затем все буквы, и, наконец, все остальные символы, сохраняя исходный порядок в каждой группе символов.

**11** Дан текстовый файл. Используя***стек***, сформировать новый текстовый файл, каждая строка которого содержит символы соответствующей строки исходного файла, записанные в обратном порядке.

**12** Дан файл из целых чисел. Используя ***очередь*,**за один просмотр файла напечатать сначала все отрицательные числа, затем все положительные числа, сохраняя исходный порядок в каждой группе.

**13** Дан текстовый файл. Используя***стек*,**сформировать новый текстовый файл, содержащий строки исходного файла, записанные в обратном порядке: первая строка становится последней, вторая – предпоследней и т.д.

**14** Дан текстовый файл. Используя***очередь*,**переписать содержимое его строк в новый текстовый файл, перенося при этом в начало каждой строки все входящие в нее буквы, затем все цифры, и, наконец, все остальные символы строки, сохраняя исходный порядок в каждой группе символов.

**15** Дан текстовый файл. Используя***стек*,**вычислить значение логического выражения, записанного в текстовом файле в следующей форме:

< ЛВ > ::= **T**|**F**| (**N**<ЛВ>) | (<ЛВ>**A**<ЛВ>) | (<ЛВ>**X**<ЛВ>) | (<ЛВ>**O**<ЛВ>),

где буквами обозначены логические константы и операции:

**T**– True,**F**– False,**N**– Not,**A**– And,**X**– Xor,**O**– Or.

**16** Дан текстовый файл. В текстовом файле записана формула следующего вида:

<Формула> ::= <Цифра> | **M**(<Формула>,<Формула>) |**N**(Формула>,<Формула>)

< Цифра > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

где буквами обозначены функции:

**M**– определение максимума,**N**– определение минимума.

Используя ***стек***, вычислить значение заданного выражения.

**17** Дан текстовый файл. Используя ***стек***, проверить, является ли содержимое текстового файла правильной записью формулы вида:

< Формула > ::= < Терм > | < Терм > + < Формула > | < Терм > - < Формула >

< Терм > ::= < Имя > | (< Формула >)

< Имя > ::= **x**|**y**|**z**

**18** В текстовом файле хранится выражение, записанное в постфиксной форме. Используя ***стек***, вычислить значение выражения.

*Пример выражения:***235+\*76-\*=>16**

**19** В текстовом файле хранится выражение, записанное в инфиксной форме. Используя ***стек***, перевести его в постфиксную форму и в таком виде записать в новый текстовый файл.

*Пример выражения:***a + b / c / d \* e => a b c / d / e \* +**

**20** В текстовом файле хранится выражение, записанное в постфиксной форме. Используя ***стек***, перевести его в инфиксную форму и в таком виде записать в новый текстовый файл.

*Пример выражения:****a b + c \* d – f \**=> *((a + b) \* c – d) \* f***

**21** Используя стек, решить следующую задачу. В текстовом файле *LOG* записано без ошибок логическое выражение (ЛВ) в следующей форме:

<ЛВ> ::= true | false | (¬<ЛВ>) | (<ЛВ>∧<ЛB>) | *(<* ЛВ >v< ЛВ >)

Вычислить (как boolean) значение этого выражения. Знаки - ¬, ∧, ∨ означают соответственно отрицание, конъюнкцию И ДИЗЪЮНКЦИЮ.

**22** Используя стек, решить следующую задачу. В текстовом файлеf записана без ошибок формула следующего вида:

<формула> ::= <цифра> | М(<формула>, <формула>) | m(<формула>, <формула>)

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9, где М обозначает функцию max, a m- min. Вычислить как целое число значение данной формулы. Например, М(5,m (6,8))=6.

**23** Последовательность символов, ограниченную точкой, занести в два стека, содержащих гласные и согласные буквы. Вывести текст и элементы из обоих стеков.

**24** Используя стек, решить следующую задачу.

В текстовом файле T записан текст, сбалансированный по круглым скобкам:

<текст> ::= <пусто> | <элемент><текст>

<элемент> ::= <буква> | (<текст>)

Требуется для каждой пары скобок напечатать номера их позиций в тексте в порядке возрастания номеров позиций открывающих скобок. Например, для текста А + (45 - F(X) \* (В - С)) нужно напечатать

3 17; 8 10; 12 16.

**25** Используя очередь, решить следующую задачу. Содержимое текстового файла f, разделенное на строки, переписать в текстовый файл *g,* перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного порядка как среди цифр, так и среди остальных литер строки).