Теоретический материал: стеки, очереди, деки (C++)

Сайт: Дистанционная подготовка
Курс: Структуры данных
Book: Теоретический материал: стеки, очереди, деки (C++)
Printed by: Гость
Date: Понедельник 19 Март 2018, 20:14

Table of Contents

Стек, очередь, дек

Стеки, очереди, деки в STL

Стек, очередь, дек

Стек

Стеком (англ. *stack*) называется хранилище данных, в котором можно работать только с одним элементом: тем, который был добавлен в стек последним. Стек должен поддерживать следующие операции:

push
: Добавить (положить) в конец стека новый элемент

pop
: Извлечь из стека последний элемент

back
: Узнать значение последнего элемента (не удаляя его)

size
: Узнать количество элементов в стеке

clear
: Очистить стек (удалить из него все элементы)

Хранить элементы стека мы будем в массиве. Для начала будем считать, что максимальное количество элементов в стеке не может превосходить константы MAX_SIZE, тогда для хранения элементов массива необходимо создать массив размера MAX_SIZE.

Объявим структуру данных типа stack.

```
const int MAX_SIZE=1000;

struct stack {
    int m_size;           // Количество элементов в стеке
    int m_elems[MAX_SIZE]; // Массив для хранения элементов

    stack();              // Конструктор
    ~stack();             // Деструктор
    void push(int d);     // Добавить в стек новый элемент
    int pop();            // Удалить из стека последний элемент
                          // и вернуть его значение
    int back();           // Вернуть значение последнего элемента
    int size();           // Вернуть количество элементов в стеке
    void clear();         // Очистить стек
};
```

Объявленная здесь структура данных stack реализует стек целых чисел. Поле структуры m_size хранит количество элементов в стеке в настоящее время, сами элементы хранятся в элементах массива m_elems с индексами 0..m_size-1. Элементы, добавленные позже, получают большие номера.

Упражнение A - простой стек

Реализуйте структуру данных "стек", реализовав все указанные здесь методы. Напишите программу (функцию main), содержащую описание стека и моделирующую работу стека. Функция main считывает последовательность команд и в зависимости от команды выполняет ту или иную

операцию. После выполнения одной команды программа должна вывести одну строчку. Возможные команды для программы:

push n
Добавить в стек число n (значение n задается после команды). Программа должна вывести ok.

pop
Удалить из стека последний элемент. Программа должна вывести его значение.

back
Программа должна вывести значение последнего элемента, не удаляя его из стека.

size
Программа должна вывести количество элементов в стеке.

clear
Программа должна очистить стек и вывести ok.

exit
Программа должна вывести bye и завершить работу.

Гарантируется, что набор входных команд удовлетворяет следующим требованиям: максимальное количество элементов в стеке в любой момент не превосходит 100, все команды pop_back и back корректны, то есть при их исполнении в стеке содержится хотя бы один элемент.

Пример протокола работы программы

Ввод	Вывод
push 2	ok
push 3	ok
push 5	ok
back	5
size	3
pop	5
size	2
push 7	ok
pop	7
clear	ok
size	0
exit	bye

Упражнение B - стек с обработкой ошибок

Аналогично предыдущему заданию, только снимается ограничение на корректность вызовов методов back и pop. Данные операции должны перед исполнением проверять, содержится ли в стеке хотя бы один элемент. Если во входных данных встречается операция back или pop, при этом стек пуст, то программа должна вместо числового значения вывести строку error.

При этом должна быть реализована двойная защита: вызов методов forward и pop для пустого стека не должен приводить к обращению к несуществующим элементам массива m_elems, а функция main должна выводить сообщение error, при считывании некорректной операции.

Пример протокола работы программы

Ввод	Вывод

```
push 2          ok
back            2
pop             2
size            0
pop             error
push 1          ok
size            1
exit            bye
```

Упражнение C - стек без ограничения на размер

Реализуйте стек динамического размера, то есть ограниченный только объемом свободной оперативной памяти. Для этого используйте указатели и динамически распределяемую память. Если для полностью заполненного стека вызывается метод push размер динамического массива, отведенного для хранения стека, должен увеличиваться.

Очередь

Очередью (англ. *queue*)) называется структура данных, в которой элементы кладутся в конец, а извлекаются из начала. Таким образом, первым из очереди будет извлечен тот элемент, который будет добавлен раньше других.

Элементы очереди будем также хранить в массиве. При этом из очереди удаляется первый элемент, и, чтобы не сдвигать все элементы очереди, будем в отдельном поле m_start хранить индекс элемента массива, с которого начинается очередь. При удалении элементов, очередь будет "ползти" дальше от начала массива. Чтобы при этом не происходил выход за границы массива, замкнем массив в кольцо: будем считать, что за последним элементом массива следует первый.

Описание структуры очередь:

```
const int MAX_SIZE=1000;

struct queue {
    int m_size;           // Количество элементов в очереди
    int m_start;          // Номер элемента, с которого начинается очередь
    int m_elems[MAX_SIZE]; // Массив для хранения элементов

    queue();              // Конструктор
    ~queue();             // Деструктор
    void push(int d);     // Добавить в очередь новый элемент
    int pop();            // Удалить из очереди первый элемент
                          // и вернуть его значение
    int front();          // Вернуть значение первого элемента
    int size();           // Вернуть количество элементов в очереди
    void clear();         // Очистить очередь
};
```

Упражнение D - простая очередь

Реализуйте простейшую очередь, размер которой не превосходит 100 элементов. Очередь поддерживает те же операции, что и стек, за исключением операции back, которая заменена операцией front. Операции front и pop всегда корректны.

Упражнение E - очередь с обработкой ошибок

Аналогично заданию B, но для очереди. Операции front и pop могут быть некорректными, в этом случае необходимо вывести error.

Программа должна содержать "двойную защиту" от некорректных операций: как в функции main, так и в самих методах pop и front.

Упражнение F - очередь без ограничений на размер

Аналогично заданию C, но для очереди. Необходимо реализовать очередь, память для которой динамически выделяется при увеличении количества элементов в ней.

Дек

Деком (англ. *deque* – аббревиатура от *double-ended queue*, двухсторонняя очередь) называется структура данных, в которую можно удалять и добавлять элементы как в начало, так и в конец. Дек хранится в памяти так же, как и очередь. Система команд дека:

push_front
: Добавить (положить) в начало дека новый элемент

push_back
: Добавить (положить) в конец дека новый элемент

pop_front
: Извлечь из дека первый элемент

pop_back
: Извлечь из дека последний элемент

front
: Узнать значение первого элемента (не удаляя его)

back
: Узнать значение последнего элемента (не удаляя его)

size
: Узнать количество элементов в деке

clear
: Очистить дек (удалить из него все элементы)

Упражнение G - простой дек

Аналогично заданиям A и D, но для дека. Количество элементов в деке в любой момент не превосходит 100. Все операции pop_front, pop_back, front, back всегда корректны.

Упражнение H - дек с обработкой ошибок

Аналогично заданиям B и E, но для дека. Количество элементов в деке в любой момент не превосходит 100. При выполнении некорректных операций необходимо вывести error.

Упражнение I - дек неограниченного размера

Аналогично заданиям C и F, но для дека. Необходимо увеличивать размер дека при увеличении числа элементов в нем. При выполнении некорректных операций необходимо вывести error.

Стеки, очереди, деки в STL

Язык программирования C++ содержит библиотеку STL (Standart template library, стандартная библиотека шаблонов). В библиотеку STL входят шаблоны для реализации стеков, деков, очередей для произвольных типов данных. Например, чтобы создать стек целых чисел необходимо использовать тип stack<int>, для создания стека строк – stack<string>, для создания стека объектов типа Person – stack<Person>.

Рассмотрим подробнее стеки, очереди, деки в STL. Описание шаблонов stack, queue, deque содержится в одноименных заголовочных файлах, то есть для использования этих структур необходимо подключить один из трех заголовочных файлов:

```
#include<stack>
#include<queue>
#include<deque>
```

После этого можно создавать объекты на основе данных шаблонов:

```
stack<int> S;      // Стек объектов типа int
queue<double> Q;   // Очередь объектов типа double
deque<string> D;   // Дек объектов типа string
```

После этого с объявленными объектами можно выполнять операции, например:

```
S.push(3);      // Положить в стек S число 3
Q.size();       // Узнать размер очереди Q
D.pop_front();  // Удалить элемент из начала дека D
```

Рассмотрим более подробно список методов, присущих шаблонам stack<type>, queue<type>, deque<type> (typeT – это тип данных, хранящихся в соответствующей структуре).

Обратите внимание, что методы удаления pop, pop_front и pop_back в STL не возвращают значения. Кроме того, реализации стека, очереди, дека в STL не содержат проверок на возникновение ошибок, то есть за некорректную работу программы при вызове операций на пустом стеке отвечает сам пользователь.

Метод очистки clear существует только у дека.

Методы шаблона stack<type>

type top()
Узнать значение верхнего элемента в стеке
void push(type)
Добавить элемент в конец стека
void pop()
Удалить верхний элемент из стека
int size()
Узнать количество элементов в стеке

Методы шаблона queue<type>

type front()
Узнать значение первого элемента в очереди
type back()
Узнать значение последнего элемента в очереди
void push(T)
Добавить элемент в конец очереди
void pop()
Удалить первый элемент из очереди
int size()
Узнать количество элементов в очереди

Методы шаблона deque<type>

T front()
Узнать значение первого элемента в деке
T back()
Узнать значение последнего элемента в деке
void push_front(T)
Добавить элемент в начало дека
void push_back(T)
Добавить элемент в конец дека
void pop_front()
Удалить первый элемент из дека
void pop_back()
Удалить последний элемент из дека
int size()
Узнать количество элементов в деке
void clear()
Удаляет все элементы из дека

Упражнения

Во всех задачах этого листка (кроме "Пробы пера") данные читаются из файла и выводятся в файл.

Задача @ – Проба пера

Решите задачу I предыдущего листка, используя стандартный deque<int>. Решение сдайте как задачу I в предыдущий контест.

Задача A – Списки по классам

Входной файл: ain.txt

Выходной файл: aout.txt

Программа на вход получает список школьников следующего вида:

```
9 Иванов
10 Петров
11 Сидоров
9 Григорьев
```

9 Сергеев
10 Яковлев

В каждой строке сначала записан номер класса (число, равное 9, 10 или 11), затем (через пробел) – фамилия. Необходимо вывести список по классам: сначала всех учащихся 9 класса, затем – 10, затем – 11. Внутри одного класса порядок вывода должен быть таким же, как на входе:

9 Иванов
9 Григорьев
9 Сергеев
10 Петров
10 Яковлев
11 Сидоров

Задача B – Игра в пьяницу

Входной файл: bin.txt

Выходной файл: bout.txt

В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт – проигрывает.

Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту ("шестерка берет туза").

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

Напишите программу, которая моделирует игру в пьяницу и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9. Программа получает на вход две строки: первая строка содержит 5 карт первого игрока, вторая строка содержит 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 10^6 ходов игра не заканчивается, программа должна вывести слово botva.

Пример:

Вход	Выход
1 3 5 7 9 2 4 6 8 0	second 5

Задача C – Правильная скобочная последовательность

Входной файл: cin.txt

Выходной файл: cout.txt

Рассмотрим последовательность, состоящую из круглых, квадратных и фигурных скобок. Программа должна определить, является ли данная скобочная последовательность правильной. Если последовательность правильная, то программа должна вывести строку yes, иначе строку no.

Примеры

Вход	Выход
([{}])	yes
([)]	no
()()	yes
())(	no

Формальное определение. Пустая последовательность является правильной. Если A – правильная, то последовательности (A), [A], {A} – правильные. Если A и B – правильные последовательности, то последовательность AB – правильная.

Указание: заведите стек, в который будут помещаться все открывающиеся скобки.

Задача D – Постфиксная запись

Входной файл: din.txt

Выходной файл: dout.txt

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов. Например, сумма двух чисел A и B записывается как A B +. Запись B C + D * обозначает привычое нам (B + C) * D, а запись A B C + D * + означает A + (B + C) * D. Достоинство постфиксной записи в том, что она не требует скобок и дополнительных соглашений о приоритете операторов для своего чтения.

Дано выражение в постфиксой записи, содержащее однозначные числа, операции +, -, *. Вычислите значение записанного выражения.

Пример

Ввод	Вывод
8 9 + 1 7 - *	-102

Задача E – Контейнеры

Входной файл: ein.txt

Выходной файл: eout.txt

На складе хранятся контейнеры с товарами N различных видов. Все контейнеры составлены в N стопок. В каждой стопке могут находиться контейнеры с товарами любых видов (стопка может быть изначально пустой).

Автопогрузчик может взять верхний контейнер из любой стопки и поставить его сверху в любую стопку. Необходимо расставить все контейнеры с товаром первого вида в первую стопку, второго

вида – во вторую стопку и т.д.

Программа должна вывести последовательность действий автопогрузчика или сообщение о том, что задача решения не имеет.

В первой строке входных данных записано одно натуральное число N, не превосходящее 500. В следующих N строках описаны стопки контейнеров: сначала записано число k_i – количество контейнеров в стопке, а затем k_i чисел виды товара в контейнерах в данной стопке, снизу вверх. В каждой стопке вначале не более 500 контейнеров (в процессе переноса контейнеров это ограничение может быть нарушено).

Программа должна вывести описание действий автопогрузчика: для каждого действия напечатать два числа из какой стопки брать контейнер и в какую стопку класть. (Обратите внимание, что минимизировать количество операций автопогрузчика не требуется.) Если задача не имеет решения, необходимо вывести одно число 0. Если контейнеры изначально правильно размещены по стопкам, то разрешается оставлять выходной файл пустым.

Пример

Ввод	Вывод
3 4 1 2 3 2 0 0	1 2 1 3 1 2

Объяснение примера. Изначально в первой стопке лежат четыре контейнера – снизу контейнер с товаром первого вида, над ним – с товаром второго вида, над ним третьего, и сверху еще один контейнер с товаром второго вида. Вторая и третья стопки – пусты.