МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г.Локтионова
«24» 12 2017 г.

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ СТЕКОВ И ОЧЕРЕДЕЙ

Методические указания по выполнению лабораторной работы по дисциплине "Алгоритмы и структуры данных" для студентов направления подготовки 09.03.04 "Программная инженерия"

Курск 2017

УДК 681.3.06(071.8)

Составители: Т.М. Белова, В.Г. Белов

Рецензент
Кандидат технических наук, доцент кафедры программной инженерии ЮЗГУ И.Н. Ефремова

Программирование с использованием стеков и очередей: методические указания по выполнению лабораторной работы по дисциплине "Алгоритмы и структуры данных" для студентов направления подготовки 09.03.04 "Программная инженерия" / Юго-Зап. гос. ун-т; сост. Т.М. Белова, В.Г. Белов. Курск, 2017. – 13 с.

Содержат основные теоретические положения и приемы разработки программ с использованием стеков и очередей на языке С++, индивидуальные задания и контрольные вопросы к защите лабораторной работы.

Методические указания соответствуют требованиям рабочей программы по дисциплине "Алгоритмы и структуры данных".

Предназначены для студентов направления подготовки 09.03.04 «Программная инженерия» дневной и заочной форм обучения.

Текст печатается в авторской редакции.

Подписано в печать 27.12.17. Формат 60х84 1/16.
Усл. печ. л. 0,7. Уч.-изд. л. 0,6. Тираж 100 экз. Заказ 4398. Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

3

Программирование с использованием стеков и очередей

- **Цель работы** — изучение фундаментальных структур данных и наиболее распространенных алгоритмов их обработки, формирование практических навыков организации и использования при решении задач динамических структур данных стеков и очередей.

Основные понятия

Память, отводимая под данные программы, делится на *статическую* и *динамическую.* Статическая память выделяется до начала работы программы под глобальные переменные и константы и освобождается только при завершении программы. Динамическая память, называемая также *кучей*, выделяется явно по запросу во время выполнения программы из ресурсов операционной системы. Она не инициализируется автоматически и должна быть явно освобождена. В отличие от статической памяти динамическая память ограничена лишь размером оперативной памяти и может динамически меняться в процессе работы программы. Недостатки динамической памяти являются продолжением ее достоинств. Во-первых, поскольку она контролируется указателями, доступ к ней осуществляется несколько дольше, чем для статической памяти. Во-вторых, программист сам должен заботиться о выделении и освобождении памяти, что чревато потенциальными ошибками.

Выделение и освобождение динамической памяти выполняется специальной программой, называемой *менеджером кучи*. Менеджер кучи хранит список всех незанятых блоков в динамической памяти. При вызове функции выделения памяти менеджер кучи ищет незанятый блок подходящего размера, выделяет в нем память и модифицирует список незанятых блоков. При вызове функции освобождения памяти блок вновь помечается как свободный. После завершения программы вся выделенная для нее динамическая память автоматически возвращается назад системе.

Как было отмечено, при работе с динамической памятью можно совершить большое количество ошибок, которые имеют различные последствия и различную степень тяжести. Большинство этих ошибок проявляется не сразу, а через некоторое время в процессе выполнения программы. Следовательно, такие ошибки трудно находимы и потому особенно опасны. Используя принцип «предупрежден – значит,

вооружен», перечислим наиболее часто встречающиеся варианты ошибок при работе с динамической памятью:

- попытка воспользоваться неинициализированным указателем;
- «висячие» указатели: после освобождения динамической памяти указатель продолжает указывать на старое место. Такие указатели называются «висячими». Попытка записи по такому указателю не приводит к немедленной ошибке. Однако память, на которую он указывает, могла быть уже выделена другой динамической переменной, и попытка записи приведет к порче этой переменной;
- «утечка» памяти: данная ошибка возникает, когда память не освобождается, но перестает контролироваться указателем. Подобную ошибку называют «утечкой» памяти, поскольку такую память невозможно освободить. Такая ошибка трудно находима, поскольку практически не сказывается на работе приложения. Однако при систематических утечках программа требует все больше памяти у операционной системы, замедляя работу других приложений.
- попытка освободить динамическую память, не выделенную ранее;
- выход за за границы выделенной памяти.

Динамические структуры данных организуются в динамической памяти с использованием указателей.

Указатели

Указатель – это переменная, которая в качестве своего значения содержит адрес некоторого байта памяти.

Значением типа указатель является адрес участка памяти, выделенного для объекта конкретного типа (ссылка на объект). Связь указателя P с объектом можно изобразить следующим образом (рисунок 1):

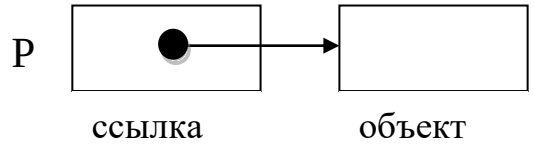

Рисунок 1 – Указатель P содержит адрес объекта

По указателю осуществляется обращение (доступ) к объекту.

Определение переменной типа указатель:

type *имя_указателя ;

где type – обозначение типа, на который будет указывать переменная с именем (идентификатором) имя_указателя. Символ ‘*’ - это унарная операция раскрытия ссылки (операция разыменования, операция обращения по адресу, операция доступа по адресу), но в данном контексте служит признаком типа указатель.

При определении переменной-указателя можно выполнить инициализацию:

type *имя_указателя инициализатор ;

Инициализатор имеет две формы записи, поэтому допустимо:

type *имя_указателя = инициализирующее_ выражение ;

type *имя_указателя (инициализирующее_ выражение);

Инициализирующее_ выражение – это константное выражение, которое может быть задано указателем, уже имеющим значение или выражением, позволяющим получить адрес объекта с помощью операции & - получение адреса операнда.

Пример 1:

```
char cc = ' f ', *p;        //Определение символьной переменной cc  и неинициализи-
                            // рованного указателя на  объект типа char
int *p1, *p2;               //Определение двух неинициализированных
//указателей  p1 и p2 на объекты типа int
char *pc = &cc;             //Инициализированный указатель на объект типа char
```

Переменной типа указатель (в дальнейшем просто указатель) можно задать значение:

- присваивая ей адрес объекта с помощью операции & (тип объекта должен быть тот же, что и тип объекта, на который ссылается указатель);
- присваивая ей значение другой переменной или выражения типа указатель.

В C++ существует механизм выделения и освобождения динамической памяти — это функции *new* и *delete*.

Пример использования new:

int * p = new int [1000]; // выделение памяти под 1000 элементов типа int.

Освобождение выделенной при помощи new памяти осуществляется посредством следующего вызова:

delete [] p;

Если требуется выделить память под один элемент, то можно использовать

int * q = new int;

или

int * q = new int(10); // выделенный int проинциализируется значением 10, в этом случае удаление будет выглядеть следующим образом:

delete q;

Динамические структуры данных стек и очередь

Стек – это абстрактный тип данных, состоящий из последовательности элементов, которые можно добавлять и извлекать из этой последовательности только с одного конца, называемого *вершиной стека*. Кроме того, можно проверять стек на пустоту.

Стек (англ. stack — стопка; читается стэк) организуется по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

Примером стека является колода карт, если для нее разрешено лишь добавлять или снимать карты с вершины колоды; действия с серединой колоды запрещены. Другой пример – магазин автомата, для которого также можно либо добавить патрон первым в магазин, либо убрать первый патрон из магазина.

О́чередь — абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (FIFO, First In — First Out). Добавление элемента (принято обозначать словом enqueue — поставить в очередь) возможно лишь в конец очереди, выборка — только из начала очереди (что принято называть словом dequeue — убрать из очереди), при этом выбранный элемент из очереди удаляется.

Стеки и очереди могут быть организованы различными способами. При этом они могут быть размещены как в статической памяти, так и в

динамической. При динамическом размещении стека и очереди обычно используется односвязный литейный список.

В стеке, представленном линейным односвязным списком, вершина стека – первый элемент списка.

Для очереди операции вставки и удаления осуществляются на разных концах очереди, для работы с ней удобно иметь два указателя – на заглавное и на последнее звенья. Если очередь пуста, оба указателя равны NULL (нуль-указатель).

В языке C++ в качестве нулевого указателя можно использовать обычное число 0 или макроопределение NULL.

Чтобы задать динамическую структуру данных надо описать её звено. Так как звено состоит из полей разных типов, то описать его можно неоднородным типом – структурой.

Задание типа элемента списка:

```
struct list
    { list  *next ;
      type  elem ; }
```

Здесь type – тип информационного поля элемента стека или очереди, поле next – ссылка на аналогичную структуру типа list.

Для примера элемент стека или очереди может быть определен:

```
struct list
  {  list   *next  ;
     int  elem ; }
```

Индивидуальные задания

1. Дан указатель head на вершину стека (если стек пуст, то head = NULL). Разработать программу, позволяющую добавлять, извлекать элементы стека, просматривать содержимое стека, подсчитывать количество элементов в стеке.
2. Дан набор из 10 чисел. Создать очередь, содержащую данные числа в указанном порядке (первое число будет размещаться в начале очереди, последнее — в конце), и вывести указатели head и tail на начало и конец очереди.

3. Даны указатели head1 и head2 на вершины стеков, хранящих некоторые отсортированные по убыванию наборы целых чисел. Разработать программу, позволяющую построить стек (head3), содержащий отсортированный по возрастанию набор чисел, извлеченных из исходных стеков.
4. Дан набор из 10 чисел. Создать две очереди: первая должна содержать числа из исходного набора с нечетными номерами (1, 3, …, 9), а вторая — с четными (2, 4, …, 10); порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе. Вывести указатели на начало и конец первой, а затем второй очереди.
5. Даны указатели head1 и head2 на вершины стеков, хранящих некоторые отсортированные по убыванию наборы целых чисел. Разработать программу, позволяющую построить стек (head3), содержащий набор упорядоченных по возрастанию четных чисел, извлеченных из исходных стеков.
6. Дан набор из 10 целых чисел. Создать две очереди: первая должна содержать все нечетные, а вторая — все четные числа из исходного набора (порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе). Вывести указатели на начало и конец первой, а затем второй очереди (одна из очередей может оказаться пустой; в этом случае вывести для нее две константы NULL для head и tail).
7. Даны указатели head1 и head2 на вершины стеков, хранящих некоторые отсортированные по убыванию наборы целых чисел. Разработать программу, позволяющую построить стек (head3), содержащий набор упорядоченных по возрастанию положительных чисел, извлеченных из исходных стеков.
8. Дан набор из 10 чисел. Создать две очереди: первая должна содержать все отрицательные, а вторая — все положительные числа из исходного набора (порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе). Вывести указатели на начало и конец первой, а затем второй очереди (одна из очередей может оказаться пустой; в этом случае вывести для нее две константы NULL для head и tail).

9. Даны указатели head1 и head2 на вершины стеков, хранящих некоторые отсортированные по убыванию наборы целых чисел. Разработать программу, позволяющую построить стек (head3), содержащий набор упорядоченных по возрастанию нечетных чисел, извлеченных из исходных стеков.
10. Дано число D и указатели head и tail на начало и конец очереди, содержащей не менее двух элементов. Добавить элемент со значением D в конец очереди и извлечь из очереди первый (начальный) элемент. Вывести значение извлеченного элемента и новые адреса начала и конца очереди. После извлечения элемента из очереди освободить память, занимаемую этим элементом.
11. Даны указатели head1 и head2 на вершины стеков, хранящих некоторые отсортированные по убыванию наборы символов. Разработать программу, позволяющую построить стек (head3), содержащий набор упорядоченных по возрастанию символов, извлеченных из исходных стеков.
12. Дано число N (> 0) и указатели head и tail на начало и конец непустой очереди. Извлечь из очереди N начальных элементов и вывести их значения (если очередь содержит менее N элементов, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести NULL). После извлечения элементов из очереди освободить память, которую они занимали.
13. Дан указатель head1 на вершину непустого стека. Создать два новых стека, переместив в первый из них все элементы исходного стека с четными значениями, а во второй — с нечетными (элементы в новых стеках будут располагаться в порядке, обратном исходному; один из этих стеков может оказаться пустым). Вывести адреса вершин полученных стеков (для пустого стека вывести NULL).
14. Даны указатели head и tail на начало и конец очереди, содержащей не менее семи элементов. С помощью функции dequeue извлечь из очереди семь элементов и вывести их значения, а также новый адрес

конца очереди. После извлечения элементов из очереди освободить память, занимаемую этими элементами.

15. Дан указатель head1 на вершину стека (если стек пуст, то P_1 = NULL). Также дано число N (> 0) и набор из N чисел. С помощью функции Push, которая включает одно число в стек, добавить в исходный стек данный набор чисел (последнее число будет вершиной стека) и вывести адрес новой вершины стека.
16. Даны указатели head и tail на начало и конец непустой очереди. Извлекать из очереди элементы, пока значение конечного элемента очереди не станет четным, и выводить значения извлеченных элементов (если очередь не содержит элементов с четными значениями, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести NULL). После извлечения элементов из очереди освобождать память, которую они занимали.
17. Дан указатель head1 на вершину стека, содержащего не менее пяти элементов. С помощью функции Pop извлечь из исходного стека пять элементов и вывести их значения. Вывести также указатель на новую вершину стека (если результирующий стек окажется пустым, то этот указатель должен быть равен NULL).
18. Даны указатели head и tail на начало и конец непустой очереди. Извлекать из очереди элементы, пока значение конечного элемента очереди не станет отрицательным, и выводить значения извлеченных элементов (если очередь не содержит элементов с отрицательными значениями, то извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести NULL). После извлечения элементов из очереди освободить память, которую они занимали.
19. Даны указатели head1 и head2 на вершины стеков, хранящих некоторые отсортированные по убыванию наборы целых чисел. Разработать программу, позволяющую построить стек (head3), содержащий набор упорядоченных по возрастанию нечетных чисел, извлеченных из исходных стеков.
20. Даны указатели head и tail на начало и конец непустой очереди. Извлекать из очереди элементы, пока значение конечного элемента очереди не станет четным, и выводить значения извлеченных элементов (если очередь не содержит элементов с четными значениями, то

извлечь все ее элементы). Вывести также новые адреса начала и конца очереди (для пустой очереди дважды вывести NULL). После извлечения элементов из очереди освободить память, которую они занимали.

21. Даны указатели head1 и head2 на вершины стеков, хранящих некоторые отсортированные по возрастанию наборы целых чисел. Разработать программу, позволяющую построить стек (head3), содержащий набор упорядоченных по убыванию положительных чисел, извлеченных из исходных стеков.
22. Даны две очереди, адреса начала и конца первой равны head1 и tail1, а второй — head2 и tail2 (если очередь является пустой, то соответствующие адреса равны NULL). Переместить все элементы первой очереди (в порядке от начала к концу) в конец второй очереди и вывести новые адреса начала и конца второй очереди. Операции выделения и освобождения памяти не использовать.
23. Дан указатель head1 на вершину стека, содержащего не менее пяти элементов. С помощью функции Pop извлекать из исходного стека элементы, пока значение начального элемента стека не станет четным, и выводить значения извлеченных элементов (если стек не содержит элементов с четными значениями, то извлечь все его элементы). Вывести также новые адрес начала стека и (для пустого стека вывести NULL). После извлечения элементов из стека освободить память, которую они занимали.
24. Дано число N (> 0) и две непустые очереди; адреса начала и конца первой равны head1 и tail1, а второй — head2 и tail2. Переместить N начальных элементов первой очереди в конец второй очереди. Если первая очередь содержит менее N элементов, то переместить из первой очереди во вторую все элементы. Вывести новые адреса начала и конца первой, а затем второй очереди (для пустой очереди дважды вывести NULL). Операции выделения и освобождения памяти не использовать.
25. . Даны две непустые очереди; адреса начала и конца первой равны head1 и tail1, а второй — head2 и tail2. Перемещать элементы из начала первой очереди в конец второй, пока значение начального элемента первой очереди не станет четным (если первая очередь не содержит четных элементов, то переместить из первой очереди во вторую все элементы). Вывести новые адреса начала и конца первой, а затем

второй очереди (для пустой очереди дважды вывести NULL). Операции выделения и освобождения памяти не использовать.

26. Даны две непустые очереди; адреса начала и конца первой равны head1 и tail1, а второй — head2 и tail2. Элементы каждой из очередей упорядочены по возрастанию (в направлении от начала очереди к концу). Объединить очереди в одну с сохранением упорядоченности элементов. Вывести указатели на начало и конец полученной очереди. Операции выделения и освобождения памяти не использовать, поля elem не изменять.
27. Даны две непустые очереди; адреса начала и конца первой равны head1 и tail1, а второй — head2 и tail2. Очереди содержат одинаковое количество элементов. Объединить очереди в одну, в которой элементы исходных очередей чередуются (начиная с первого элемента первой очереди). Вывести указатели на начало и конец полученной очереди. Операции выделения и освобождения памяти не использовать.
28. Даны указатель head1 на вершину стека, хранящего некоторые слова. Разработать программу, позволяющую построить стек (head2), содержащий набор из 5 слов, извлеченных из исходного стека. Операции выделения и освобождения памяти не использовать
29. Даны указатели head и tail на начало и конец очереди, содержащей не менее пяти элементов. С помощью функции Dequeue извлечь из исходной очереди пять начальных элементов и вывести их значения. Вывести также адреса начала и конца результирующей очереди (если очередь окажется пустой, то эти адреса должны быть равны NULL).
30. Даны указатели head1 и head2 на вершины стеков, хранящих некоторые слова и имеющих одинаковое количество элементов. Разработать программу, позволяющую построить стек (head3), содержащий набор слов, извлеченных из исходных стеков и в котором элементы исходных стеков чередуются (начиная с первого элемента первого стека). Операции выделения и освобождения памяти не использовать

Контрольные вопросы к защите лабораторной работы

1 Понятие типа указатель.
2 Задание переменных типа указатель. Операции над указателями.
3 Понятие статического и динамического объекта.

4 Создание и уничтожение динамического объекта. Операции над динамическим объектом.
5 Понятие структуры данных стек, очередь.
6 Представление в памяти структур данных стек, очередь.
7 Задание структур данных стек, очередь.
8 Основные операции над структурами данных стек, очередь.
9 Достоинства и недостатки различного представления в памяти структур данных стек, очередь.
10 Использование структур данных стек и очередь для решения задач.

Содержание отчета

Отчет по лабораторной работе включает:
- титульный лист;
- условие задания;
- алгоритм решения задачи;
- текст программы;
- результаты тестирования программы.

Список используемых источников

1. Белов В.Г. Основы программирования на языке C++ Builder [Текст]: учеб. пособие / В.Г. Белов, Т.М. Белова; Юго-Зап. гос. ун-т. – Курск, 2015. – 160 с.
2. Белов В.Г. Основы программирования на языке C++ Builder [Электронный ресурс]: учеб. пособие / В.Г. Белов, Т.М. Белова; Юго-Зап. гос. ун-т. – Курск, 2015. – 160 с.
3. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона: учебник [Текст]. Приложение: 1 электрон. опт. Диск (CD-ROM). — М.: ДМК Пресс, 2012. — с. 272.: ил.
4. Лафоре Р. Объектно-ориентированное программирование в C++ [Текст]/ Р. Лафоре. – СПб.: ПИТЕР, 2013. – 924 с.: ил.
5. Прата, С. Язык программирования C++. Лекции и упражне-ния [Текст] / С. Прата. – М.: Вильямс, 2012. – 1244 с.
6. Липпман Стенли Б. Язык программирования C++. Базовый курс [Текст] / Стенли Б. Липпман, Жози Лажойе, Барбара Э. Му. – М.: Вильямс, 2014. – 1120 с.