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
«27» 12 2017 г.

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ОДНОСВЯЗНЫХ СПИСКОВ

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 односвязных списков: 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. – 12 с.

Содержат основные теоретические положения и приемы разработки программ с использованием односвязных списков 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 60x84 1/16.
Усл. печ. л. 0,6. Уч.-изд. л. 0,5. Тираж 100 экз. Заказ 4397. Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

3

Программирование с использованием односвязных списков

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

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):

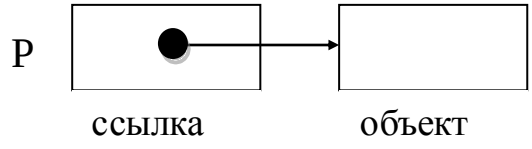

Рисунок 1 – Указатель P содержит адрес объекта

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;

Динамическая структура данных линейный односвязный список

Линейный односвязный список — это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий элемент. Последний элемент списка указывает на NULL. Элемент, на который нет указателя, является первым (головным) элементом списка. В односвязном списке можно передвигаться только в сторону конца списка. Узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно.

Следующие операции можно применять к линейным односвязный спискам:

- получить значение i-го элемента списка или изменить i-й элемент;
- напечатать элементы списка в порядке их расположения;
- найти в списке элемент с заданным значением;
- определить длину списка;
- вставить новый элемент непосредственно за i-м элементом или перед ним, вставить элемент в пустой список;
- удалить i-й элемент;
- соединить два линейных списка в один список;
- разбить список на два списка;
- создать копию списка;
- сделать список пустым.

Достоинства односвязных линейных списков

- Эффективное (за константное время) добавление и удаление элементов;
- размер ограничен только объёмом памяти компьютера и разрядностью указателей;
- динамическое добавление и удаление элементов.

Недостатки односвязных линейных списков

Недостатки связных списков вытекают из их главного свойства — последовательного доступа к данным:

- сложность прямого доступа к элементу, а именно определения физического адреса по его индексу (порядковому номеру) в списке;
- на поля-указатели (указатели на следующий и предыдущий элемент) расходуется дополнительная память (в массивах, например, указатели не нужны);
- некоторые операции со списками медленнее, чем с массивами, так как к произвольному элементу списка можно обратиться, только пройдя все предшествующие ему элементы
- соседние элементы списка могут быть распределены в памяти нелокально, что снизит эффективность кэширования данных в процессоре;
- над связными списками, по сравнению с массивами, гораздо труднее (хоть и возможно) производить параллельные векторные операции, такие, как вычисление суммы: накладные расходы на перебор элементов снижают эффективность распараллеливания.

Кольцевой связный список

Разновидностью связных списков является кольцевой (циклический, замкнутый) список. Последний элемент кольцевого списка содержит указатель на первый элемент списка.

Описание структуры для элемента списка

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 односвязный линейный список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию просмотра элементов списка. Предусмотреть ситуацию, если список пуст.
2. Дан односвязный линейный список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию добавления элемента в начало списка. Предусмотреть ситуацию, если список пуст.
3. Дан односвязный линейный список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию удаления первого элемента списка. Предусмотреть ситуацию, если список пуст.
4. Дан непустой односвязный линейный список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию удаления последнего элемента списка.
5. Даны 2 непустых односвязных линейных списка. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию объединения списков (к 1 списку дописать второй).

6. Даны 2 непустых односвязных линейных списка одинаковой длины. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию объединения списков (элементы второго списка включить в первый список на позиции 2,4,6 и т.д.).
7. Даны 2 непустых односвязных линейных списка одинаковой длины. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию объединения списков (элементы второго списка включить в первый список на позиции 1,3,5 и т.д.).
8. Дан односвязный линейный список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию удаления 1, 3, 5 и т.д. элементов списка. Предусмотреть ситуацию, если список пуст.
9. Дан односвязный линейный список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию поиска элемента по фамилии и удаления этого элемента из списка. Предусмотреть ситуацию, если список пуст или не найден элемент с нужной фамилией.
10. Дан односвязный линейный список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию удаления 2, 4, 6 и т.д. элементов списка. Предусмотреть ситуацию, если список пуст.
11. Дан односвязный циклический список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию удаления первого элемента списка. Предусмотреть ситуацию, если список пуст.
12. Дан односвязный циклический список. Элемент списка содержит следующую информацию: фамилия студента, шифр группы. Требуется создать функцию удаления последнего элемента списка. Предусмотреть ситуацию, если список пуст.
13. Дан односвязный циклический список. Элемент списка содержит координаты (x,y) точек. Требуется создать функцию удаления 1, 3, 5 и т.д. элементов списка. Предусмотреть ситуацию, если список пуст.
14. Дан односвязный циклический список. Элемент списка содержит целое число. Требуется создать функцию удаления

элементов списка, значения которых – четные числа. Предусмотреть ситуацию, если список пуст или четных чисел нет.

15. Дан односвязный циклический список. Элемент списка содержит целое число. Требуется создать функцию удаления элементов списка, значения которых – нечетные числа. Предусмотреть ситуацию, если список пуст или нечетных чисел нет.
16. Дан односвязный циклический список. Элемент списка содержит целое число. Требуется создать функцию удаления элементов списка, значения которых – положительные числа. Предусмотреть ситуацию, если список пуст или положительных чисел нет.
17. Дан односвязный линейный список. Элемент списка содержит целое число. Требуется создать функцию удаления элементов списка, значения которых – отрицательные числа. Предусмотреть ситуацию, если список пуст или отрицательных чисел нет.
18. Дан односвязный линейный список. Элемент списка содержит некоторое слово. Требуется создать функцию удаления элементов списка, слова в которых начинаются на символ 'a'. Предусмотреть ситуацию, если список пуст или таких слов нет.
19. Дан односвязный линейный список. Элемент списка содержит некоторое слово. Требуется создать функцию удаления элементов списка, слова в которых имеют длину больше 5. Предусмотреть ситуацию, если список пуст или таких слов нет.
20. Дан односвязный линейный список. Элемент списка содержит целое число. Требуется создать функцию удаления элементов списка, значения которых меньше 10. Предусмотреть ситуацию, если список пуст или таких чисел нет.
21. Дан односвязный линейный список. Элемент списка содержит целое число. Требуется создать функцию удаления элементов списка, значения равны 0. Предусмотреть ситуацию, если список пуст или таких чисел нет.
22. Дан односвязный линейный список. Элемент списка содержит некоторое слово или пустую строку. Требуется создать функцию удаления элементов списка, слова в которых – пустые строки. Предусмотреть ситуацию, если список пуст или таких слов нет.
23. Дан односвязный линейный список. Элемент списка содержит целое число. Требуется создать функцию проверки: является ли

последовательность чисел в списке упорядоченной по возрастанию. Предусмотреть ситуацию, если список пуст.

24. Дан односвязный линейный список. Элемент списка содержит целое число. Требуется создать функцию проверки: является ли последовательность чисел в списке упорядоченной по убыванию. Предусмотреть ситуацию, если список пуст.
25. Дан односвязный линейный список. Элемент списка содержит целое число. Требуется переставить первый и последний элементы списка. Предусмотреть ситуацию, если список пуст.
26. Дан односвязный линейный список. Элемент списка содержит целое число. Требуется переставить первый и второй элементы списка. Предусмотреть ситуацию, если список пуст.
27. Дан односвязный линейный список. Элемент списка содержит некоторое слово. Требуется создать функцию удаления элементов списка, слова в которых совпадают с введенным словом. Предусмотреть ситуацию, если список пуст или таких слов нет.
28. Дан односвязный линейный список. Элемент списка содержит целое число. Требуется создать функцию удаления элементов списка, значения которых совпадают с введенным числом. Предусмотреть ситуацию, если список пуст или таких чисел нет.
29. Даны 2 односвязных линейных списка. Элементы списка содержат целые числа. Создать третий список, в который включить только одинаковые числа из списков. Предусмотреть ситуацию, если списки пусты или таких чисел нет.
30. Даны 2 односвязных линейных списка. Элементы списка содержат целые числа. Создать третий список, в который включить только разные числа из списков. Предусмотреть ситуацию, если списки пусты.

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 односвязный линейный список.

6 Представление в памяти структуры данных односвязный линейный список.
7 Задание структуры данных односвязный линейный список.
8 Основные операции над структурой данных односвязный линейный список.
9 Достоинства и недостатки различного представления в памяти структуры данных односвязный линейный список.

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.
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.
7. Страуструп, Б. Программирование. Принципы и практика использования C++ [Текст] / Б. Страуструп. – М.: Вильямс, 2011. – 1206 с.