

Самостоятельная работа по темам 8–10 (Раздел 4 – Сортировка и поиск)

Цель выполнения самостоятельной работы – закрепление теоретических знаний и получение навыков решения задач по темам Раздела 4 «Сортировка и поиск»:

Тема 8. Задача сортировки и сортировка массивов

Тема 9. Поиск и хеширование

Тема 10. Внешняя сортировка и поиск информации на внешних устройствах

В ходе выполнения *проверяются следующие планируемые результаты обучения*:

- Студент *знает*: основные методы и алгоритмы сортировки массивов; понятие индекса и простейшие способы индексации данных в массивах и файлах; особенности сортировки и поиска данных на внешних запоминающих устройствах; возможности использования динамических структур данных (списков, бинарных деревьев) для решения задач сортировки и поиска.
- Студент *умеет*: разрабатывать программы средней сложности на языках высокого уровня, конструируя типы данных, используя функции управления памятью для создания динамических структур (бинарные деревья, одно- и двунаправленные линейные списки с различными дисциплинами вставки и удаления элементов пр.), с использованием языка C++; разрабатывать и оценивать алгоритмы сортировки и поиска на основе различных методов (с использованием языка C++).
- Студент *имеет навыки* разработки и отладки приложений, использующих различные типы данных и динамические структуры данных (бинарные деревья, одно- и двунаправленные линейные списки с различными дисциплинами вставки и удаления элементов пр.), реализующих итерационные и рекурсивные алгоритмы на языках высокого уровня с использованием современных инструментальных средств (с использованием языка C++); работы с файлами с различной организацией; самостоятельного решения задач с помощью компьютеров, изучения новых возможностей и средств разработки программ (работы с редактором кода, средствами отладки).
- Студент *имеет представление* об организации хранения данных на внешних запоминающих устройствах, файлов с различной организацией; о методах сортировки и поиска данных на внешних запоминающих устройствах.

Задания самостоятельной работы

Разработайте структурный тип данных языке C++ (аналог типа данных «запись» (Record) Языка Pascal) для представления информации о любом объекте любой предметной области (на выбор студента, согласуется с преподавателем в течение одной недели после выдачи самостоятельной работы).

Пример 1:

- Данные о рейтинге студентов (номер, имя, шифр группы, направление подготовки, рейтинг и пр.).
- Данные участников соревнований (номер, имя, возраст, название команды, результат и пр.).
- Телефонный справочник (с указанием имени, категории, даты рождения, номера телефона и пр.).
- Справочник контактов (с указанием имени, e-mail, типа почты, организации и пр.).
- Учёт финансовых операций (расходов на различные нужды и т. п.).
- Планирование мероприятий (с указанием названия, места проведения, даты и времени проведения и т. д.).
- Расписание движения транспорта (маршрут, остановка, дата прибытия транспорта и пр.).

- Расписание занятий (название занятия, дата и время проведения занятия, приоритет участия в занятии и пр.) ...

Рекомендация:

Рассмотрим на примере описание структурного типа данных для представления информации о рейтинге студента. Структура должна включать следующие поля (таблица 1):

Таблица 1 – Информация о студентах

Номер зачётной книжки студента	Фамилия	Имя	Отчество	Год рождения	Рейтинг
--------------------------------	---------	-----	----------	--------------	---------

На языке C++ для описания *записи данных* о студенте необходимо определить структурный тип данных, включающий эти информационные поля:

```
struct Student // менее этот тип данных можно использовать также, как и любой базовый тип
{
    unsigned int StudentNumber; // один из ключей записи
    string LastName;
    string FirstName;           // другой из ключей записи
    string MiddleName;
    unsigned int BirthYear;
    unsigned int Rating;        // еще один из ключей записи
};
```

(1)

Предполагаем, что номер зачётной книжки как ключ является уникальным.

Разработайте свою структуру данных, используя поля (атрибуты) разных типов, представляющие характеристики объектов.

Каждое из информационных полей должно иметь уникальное имя.

Тип данных для этих полей должен быть выбран с учетом возможных значений (**окружающий мир**) и минимально достаточного размера памяти для представления в компьютере (ограничения **аппаратуры и возможности языка программирования**).

Задание 1. Сортировка и поиск данных в массивах с использованием индексов-массивов

Пример 2. Пусть нам известны данные о 10 студентах (таблица 2).

Таблица 2 – Данные о студентах

Номер зачётной книжки студента	Фамилия	Имя	Отчество	Год рождения	Рейтинг
24503	Петров	Александр	Валентинович	2006	76
24465	Сидоров	Леонид	Данилович	2007	12
23377	Кукушкина	Анастасия	Юрьевна	2005	54
24564	Павлов	Анатолий	Борисович	2006	92
24423	Репин	Виктор	Андреевич	2006	4
22398	Рогова	Мария	Владимировна	2004	90
24350	Носкова	Светлана	Афанасьевна	2006	65
24413	Тырцев	Петр	Алексеевич	2007	112
24487	Яшкин	Арсений	Петрович	2007	44
23515	Аспидова	Ева-Валерия	Ивановна	2005	89

С этими данными необходимо выполнять различные **действия**. Рассмотрим некоторые из них.

Пример 3. Сформировать список студентов с учетом возрастания номера зачетной книжки (упорядочить (**отсортировать**) по возрастанию ключа «Номер зачетной книжки студента»). Результат сортировки представлен в таблице 3.

Таблица 3 – Данные о студентах, отсортированные по возрастанию номера зачетной книжки

Номер зачётной книжки студента	Фамилия	Имя	Отчество	Год рождения	Рейтинг
22398	Рогова	Мария	Владимировна	2004	90
23377	Кукушкина	Анастасия	Юрьевна	2005	54
23515	Аспидова	Ева-Валерия	Ивановна	2005	89
24350	Носкова	Светлана	Афанасьевна	2006	65
24413	Тырцев	Петр	Алексеевич	2007	112
24423	Репин	Виктор	Андреевич	2006	4
24465	Сидоров	Леонид	Данилович	2007	12
24487	Яшкин	Арсений	Петрович	2007	44
24503	Петров	Александр	Валентинович	2006	76
24564	Павлов	Анатолий	Борисович	2006	92

Пример 4. Сформировать список студентов с учетом убывания рейтинга (**отсортировать по убыванию** ключа «Рейтинг»). Результат сортировки представлен в таблице 4.

Таблица 4 – Данные о студентах, отсортированные по убыванию рейтинга

Номер зачётной книжки студента	Фамилия	Имя	Отчество	Год рождения	Рейтинг
24413	Тырцев	Петр	Алексеевич	2007	112
24564	Павлов	Анатолий	Борисович	2006	92
22398	Рогова	Мария	Владимировна	2004	90
23515	Аспидова	Ева-Валерия	Ивановна	2005	89
24503	Петров	Александр	Валентинович	2006	76
24350	Носкова	Светлана	Афанасьевна	2006	65
23377	Кукушкина	Анастасия	Юрьевна	2005	54
24487	Яшкин	Арсений	Петрович	2007	44
24465	Сидоров	Леонид	Данилович	2007	12
24423	Репин	Виктор	Андреевич	2006	4

Пример 5. Сформировать список студентов с учетом возрастания значения имён (**отсортировать в лексико-графическом порядке по ключу «Имя»**). Результат сортировки представлен в таблице 5.

В таблице 5 информационное поле «Имя» содержит значение двойного женского имени, в котором используется символ «-».

Таблица 5 – Данные о студентах, отсортированные в лексико-графическом порядке имен

Номер зачётной книжки студента	Фамилия	Имя	Отчество	Год рождения	Рейтинг
24503	Петров	Александр	Валентинович	2006	76
23377	Кукушкина	Анастасия	Юрьевна	2005	54
24564	Павлов	Анатолий	Борисович	2006	92
24487	Яшкин	Арсений	Петрович	2007	44
24423	Репин	Виктор	Андреевич	2006	4
23515	Аспидова	Ева-Валерия	Ивановна	2005	89
24465	Сидоров	Леонид	Данилович	2007	12
22398	Рогова	Мария	Владимировна	2004	90
24413	Тырцев	Петр	Алексеевич	2007	112
24350	Носкова	Светлана	Афанасьевна	2006	65

! При сортировке данных типа «строка» использовать термин «сортировка в алфавитном порядке» некорректно, т.к. в строках могут встречаться символы, отличные от букв. Поэтому профессиональный термин – это «сортировка в лексико-графическом порядке».

Сортировка строковых данных выполняется по числовым кодам символов строк. При этом используются операции посимвольного сравнения строк. Для строк s_1 и s_2 :

– $s_1 = s_2$, если длина $s_1 =$ длина s_2 и все символы s_1 и s_2 попарно равны

Пример 6: "РИС"="РИС", но "RIS-24" != "RIS", "RIS" != "RIO", "RIS" != "HSE"

– $s_1 > s_2$, если длина $s_1 >$ длина s_2 или (длина $s_1 =$ длина s_2 и существует i , для которого код(s_{1i}) > код(s_{2i}))

Пример 7: "РИС-24" > "РИС", "RIS" > "RIO", "RIS" > "HSE"

– $s_1 < s_2$, если длина $s_1 <$ длина s_2 или (длина $s_1 =$ длина s_2 и существует i , для которого код(s_{1i}) < код(s_{2i}))

Пример 8: "RIS" < "RIS-24", "RIS-23" < "RIO-24", "HSE" < "RIS"

Опишите массив записей созданного выше типа данных согласно (1). Организуйте ввод данных с клавиатуры для описанного массива (данные в массив включаются в порядке ввода): напишите функцию ввода данных в массив.

Рекомендация:

Определите размерность массива – количество записей об объектах предметной области. Это может быть фиксированное количество элементов или пользователь может при каждом выполнении программы определять это количество. Об особенностях передачи массива аргументом функции С++ см. в приложении А. Во втором случае нужно использовать динамический массив (код программы с динамическим массивом см. в файле ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ\Динамический_массив.cpp; пример работы с динамическим массивом см. в приложении Б). Опишите массив данных с записями.

Пример 9: Объявление массива записей с фиксированным количеством элементов

```
const unsigned __int8 SdntNum = 10; // Количество записей о студентах
Student SdntArr[SdntNum]; // Массив (статический) записей о студентах
```

Данные можно вводить в массив по полям каждой записи с клавиатуры или считывать из файла (код программы ввода данных из файла см. в файле ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ\списки\LIFO_List-WithFiles.cpp; пример ввода данных из файла см. в приложении В).

Пример 10. Ввод полей одной записи с клавиатуры в первый элемент массива:

```
cout << "Введите поля записи: > ";
cout << " Введите Номер зачётной книжки студента: > ";
cin >> SdntArr[0].StudentNumber;
cout << " Введите Фамилию студента: > ";
cin >> SdntArr[0].LastName;
cout << " Введите Имя студента: > ";
cin >> SdntArr[0].FirstName;
cout << " Введите Отчество студента: > ";
cin >> SdntArr[0].MiddleName;
cout << " Введите Год рождения студента: > ";
cin >> SdntArr[0].BirthYear;
cout << " Введите Рейтинг студента: > ";
cin >> SdntArr[0].Rating;
```

Для ввода всех записей в массив организуется цикл по всем элементам массива записей. Если данные вводятся из файла, то ввод может быть организован, пока не закончатся записи в файле (пока не будет считан конец файла), но при этом число записей файла не должно превышать количество элементов в массиве, описанном в программе (**SdntNum**).

Если данные вводятся из файла как строки, то нужно определить, как поля будут в строках отделяться друг от друга – определить (задать в программе) символ-разделитель. После ввода каждой строки разделить её по разделителям на отдельные подстроки, которые нужно присвоить значениям полей в структуре. При этом строки должны быть преобразованы к нужному типу.

Пример 11: К числовым типам данных необходимо преобразовать такие поля как «Номер зачётной книжки студента», «Рейтинг» и т.п.

Для проверки правильности ввода выведите введённые в массив данные (в цикле).

Пример 12: Вывод данных из одной записи (числовые данные преобразуются в строку):

```
cout << "Введены записи: > " << endl;
cout << " Данные в записи номер " << to_string(SN) << endl;
cout << " Номер зачётной книжки студента: > " << to_string(SdntArr[0].StudentNumber) << endl;
cout << " Фамилия студента: > " << SdntArr[0].LastName << endl;
cout << " Имя студента: > " << SdntArr[0].FirstName << endl;
cout << " Отчество студента: > " << SdntArr[0].MiddleName << endl;
cout << " Год рождения студента: > " << to_string(SdntArr[0].BirthYear) << endl;
cout << " Рейтинг студента: > " << to_string(SdntArr[0].Rating) << endl;
cout << endl;
```

Чтобы не было проблем с вводом и выводом информации (строк) на кириллице, можно попробовать использовать инструкции:

```
SetConsoleOutputCP(1251);
SetConsoleCP(1251);
```

или варианты из таблицы 6.



Зачем нужны сортировка и индексация?

Если набор ключей отсортирован, то при использовании различных способов представления индексов можно оценить и сравнить вычислительную сложность соответствующего алгоритма поиска.

Таблица 6 – Варианты установки кодировки UTF-8

Код	Пример кода из файла
#pragma execution_character_set("utf-8")	Файл ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ\списки\LIFO_List-WithFiles.cpp
SetConsoleOutputCP(65001);	Файлы ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ\списки\LIFO_List-WithFiles.cpp, ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ\списки>ListLIFOwithDELETE.cpp
setlocale(LC_ALL, "Russian");	Файл ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ\Динамический_массив.cpp

Пример 13: Поиск элемент со значением **10** - сравнение операций поиска в отсортированных массиве, бинарном дереве поиска и линейном списке [1]:

<p><i>массив</i></p>	<p>Для поиска в массиве применяется традиционный подход на основе половинного деления — метод дихотомии</p>
<p><i>бинарное дерево поиска</i></p>	<p>Аналогичная операция и на массиве, и на бинарном дереве поиска будет выполнена за три шага</p>
<p><i>список</i></p>	<p>В то же время поиск этого элемента на списке займет десять шагов</p>

Выполните индексацию записей по значениям атрибутов объектов (как минимум, по двум разным атрибутам объектов), построив индексы для каждого выделенного атрибута в виде отсортированных массивов ключей (по возрастанию и убыванию в зависимости от назначения индекса.

Пример 14: Сортировка по возрастанию значений атрибутов для индекса по именам студентов (сортировка в лексико-графическом порядке) и по убыванию – для рейтинга.



Результаты действий с данными получены в таблицах 3-5 для массива записей небольшого размера (**SdntNum** = 10). Заметить выигрыш в памяти и времени работы алгоритмов при таком размере таблицы записей не получится. Однако при увеличении значения **SdntNum** (например, работать с данными студентов 1-го курса РИС Пермского кампуса ВШЭ или с данными всех студентов 1-го курса всех кампусов ВШЭ) индексация записей покажет свои преимущества.

Массив индекс должен содержать значения атрибута, выбранного для индексации, и номера соответствующих элементов в массиве введенных данных для получения полных данных об объектах по индексу (см. файлы Сортировка массивов, индексы.pdf, Бинарные деревья для поиска и сортировки записей.pdf).

Рекомендация:

Пример 15: Для индексации введенных в массив записей по уникальным номерам зачётной книжки для быстрого поиска по ним данных студентов необходимо описать массив из того же числа элементов, что и массив с полными данными студентов. Структурный тип данных для массива индексов и описание массива индексов:

```
struct StudentOnNumber
{
    unsigned int StudentNumber; // поле, по которому выполняется индексация
    unsigned __int8 RecNum; // номер записи в исходном массиве с полными данными
};
```

StudentOnNumber StudIndexOnNumber[**SdntNum**]; // Массив-индекс по номерам зачёток

Массив-индекс может строиться одновременно с вводом данных (в том же цикле) – после ввода данных заполняются поля и в массиве-индексе через присваивание соответствующих введенных значений:

```
StudIndexOnNumber[SN].StudentNumber = SdntArr[SN].StudentNumber;
StudIndexOnNumber[SN].RecNum = SN;
```

Массив-индекс должен быть отсортирован по ключу (по полю **StudentNumber**).

Пример 16: Для сортировки массива-индекса по номеру зачётной книжки можно использовать следующую функцию:

```
void OrderArrayOnNumber(StudentOnNumber* A, unsigned __int8 AN)
// аргументы: адрес начала массива типа StudentOnNumber в указателе A;
// AN – количество элементов в массиве
{
    StudentOnNumber X; // вспомогательная переменная типа StudentOnNumber
    unsigned __int8 i, j;
    for (i = 2; i < AN; i++)
    {
        X = A[i];
        A[0] = X; // Занесли в барьерный элемент перемещаемое значение
        j = i - 1; // Поиск места для вставки начинаем с элемента, предшествующего выбранному
        while (X.StudentNumber < A[j].StudentNumber) // сравниваем ключи
        {
            A[j + 1] = A[j]; // "Выдавливаем" элементы массива на освобождаемое -
            j = j - 1; // место - раздвигаем отсортированную часть
        }
        // Включаем элемент в отсортированную часть на найденную позицию:
        A[j + 1] = X;
    }
}
```

Вызываем её после ввода всех данных и заполнения массива-индекса:

```
OrderArrayOnNumber(StudIndexOnNumber, SdntNum);
```

Примечание: Сортировку массива-индекса вставками можно выполнять одновременно с вводом данных. Тогда необходимо использовать два индекса – разных для массива с полными данными и для массива-индекса, т. к. записи в индексе будут смещаться при размещении нового элемента не в текущую позицию, а в позицию, выбранную для вставки в порядке возрастания ключей.

Для ускорения поиска и по другим полям для них аналогично нужно описать массивы индексы и выполнить сортировку по ним.

Пример 17: Создать массив-индекс для сортировки по рейтингу в порядке невозрастания ключей.

Пример 18: Создать массив-индекс для поиска по составному ключу, полученному соединением фамилии, имени и отчества (конкатенацией трёх строк-полей) – ключ для поиска может вычисляться с помощью функции, разработанной программистом по заданным правилам. Прототип (заголовок/сигнатура) такой функции-заглушки *Key()* приведён в файлах Массивы\OrderByInsert&Selection.cpp, Массивы\OrderByInsert.cpp.

Для сортировки можно использовать различные алгоритмы (см. файлы папки Массивы).

В C++ существует встроенная функция qsort в библиотеке `<cstdlib>` [2], которая выполняет сортировку элементов массива, при этом порядок следования элементов в отсортированном массиве задается в вызове этой функции. Вы можете проверить работу алгоритмов, созданных Вами, сравнив их результаты с результатом вызова библиотечной функции.

Усложнение задачи сортировки при создании индексов (работа с разными типами данных, вычисление ключей по значениям полей записи, реализация разных методов сортировки) даёт дополнительный балл к оценке.

После выполнения этого этапа должны быть построены и заполнены основной массив данных с записями об объектах выбранной предметной области и массивы-индексы по двум разным ключам (индексы должны быть отсортированы).

Разрабатывайте функции

- *Вывода данных* (записей из массива) по возрастанию/убыванию значений атрибутов, по которым выполнена индексация. При реализации процедур в цикле последовательно просматриваются элементы массивов-индексов и по указанным в них номерам выбираются записи из основного массива, содержащего введенные данные, которые нужно вывести на экран, отделив записи при выводе, например, пустыми строками.

Рекомендация:

Для вывода данных с сортировкой по заданным ключам разрабатываются отдельные функции для вывода данных, отсортированных по каждому из ключей. В функции передаются массивы-записи с полными данными и массивы индексы, количество элементов в них. В цикле перебираются все элементы массивов-индексов от первого до последнего.

Для каждого элемента массива индекса выбирается поле с номером соответствующей записи в основном массиве данных и выводятся полные записи в порядке, определённом в индексе.

Пример 19: Вывод отдельной записи, соответствующей выбранному элементу индекса (*IndNum*):

```
// Пример вывода записи с данными из массива по номеру строки в массиве-индексе IndNum:  
unsigned __int8 IndNum = 0; // Взяли первую запись в индексе (для этого примера)  
// Выбор номера записи в основном массиве по полю из индекса:  
SN = StudIndexOnNumber[IndNum].RecNum;  
// Выводим данные из найденной записи с номером SN - пример вывода приведён выше...
```

Для вывода всех записей перебираем все записи в индексе по номерам *IndNum* от 0 до *SdntNum-1*.



Если данные отсортированы, то для поиска по ключу можно применять алгоритм бинарного поиска, который в общем случае работает быстрее простого перебора значений.

- Поиска элементов по значениям ключевых атрибутов с использованием бинарного поиска (поиска делением пополам, дихотомии) в построенных индексах. В случае, если элемент не найден, должно быть выведено соответствующее сообщение. Если элемент с заданным для поиска значением ключа найден в индексе, должны быть выведены значения всех атрибутов из соответствующей записи в массиве введенных данных. Разработайте два варианта функций поиска по индексам: по одному из атрибутов – *рекурсивную* функцию поиска, по другому – *итерационную*.

Рекомендация:

При выполнении задания можно выполнить поиск по одному ключу с помощью метода деления массива пополам, реализованного рекурсивной функцией [3], а по другому ключу – итерационной [4-6]. Задача усложняется, если значения ключей не являются уникальными (например, в нашем случае есть полные тёзки – при поиске по составному ключу, собранному по фамилии, имени и отчеству, может быть найдено несколько записей, т. е. в случае успешного поиска нужно вывести не одну запись по найденному номеру, а проверить на соответствие условиям поиска ещё и соседние записи.

Пример вызова функции бинарного поиска:

```
// Поиск элемента в массиве по номеру зачётки:  
// Функция возвращает номер записи массива с номером зачетной книжки StNum  
// или -1, если запись с таким номером не будет найдена  
SN = BinSearch(StudIndexOnNumber, StNum, 0, SdntNum - 1);  
if (SN == -1)  
{ cout << "Запись не найдена!" << endl;  
}  
else  
{ cout << "Найдена запись с номером " << to_string(SN) << endl;  
  // Вывод данных (полей из записи): ...  
};
```

В C++ существует встроенная функция **binary_search** в библиотеке `<algorithm>` [7], которая проверяет, есть ли в отсортированном диапазоне элемент, равный указанному значению. Однако, она возвращает значение `true`, если указанный элемент имеется в диапазоне, и `false`, если такой элемент отсутствует. Вы можете проверить работу алгоритмов, созданных Вами, сравнив их результаты с результатом вызова библиотечной функции.

- Редактирования записей в массиве с возможностью изменения значений атрибутов, по которым выполнена индексация записей массива. При этом соответствующие изменения вносятся и в индексы.

Рекомендация:

Перестройку индекса можно выполнить, как при вводе данных – сформировать массив-индекс по изменённым данным и выполнить его повторную сортировку. Лучший вариант – оптимизировать переиндексацию, не выполняя перестройку индекса полностью.

- Удаления записи с заданным значением ключевого атрибута из массива с соответствующим изменением индекса.

Задание 2. Сортировка и поиск данных в массивах с использованием бинарного дерева

Опишите массив записей созданного выше типа данных согласно (1). Организуйте ввод данных с клавиатуры для описанного массива (данные в массив включаются в порядке ввода): напишите функцию ввода данных в массив.

Выполните индексацию записей по значению выбранного ключевого атрибута объектов, построив индекс в виде бинарного дерева. Индекс (каждый узел бинарного дерева) должен содержать значения ключевого атрибута, выбранного для индексации, и номера соответствующего элемента в массиве введенных данных для получения полных данных об объектах по индексу.

Рекомендация:

Данные в массив вводятся однократно. Индекс (бинарное дерево) строится с помощью функции создания бинарного дерева, где структура элементов включает поля (табл. 6):

Таблица 6 – Поля структурного типа данных для индекса в виде бинарного дерева

Информационные поля		Ссылочные поля	
<i>StudentOnNumber</i>			
StudentNumber	RecNum	Left	Left

// Описание рекурсивного типа данных для узла бинарного дерева:

```
struct TreeNode
```

```
{ StudentOnNumber Inf; // Информационные поля: номер зачётки и номер записи в массиве данных  
  TreeNode* Left = nullptr; // Ссылка на левое поддерево - пустая для новой вершины  
  TreeNode* Right = nullptr; // Ссылка на правое поддерево - пустая для новой вершины  
};
```

TreeNode* RootNode = nullptr; // Ссылка на корень дерева-индекса, пустая, если в дереве нет узлов

То есть данные массива-индекса дополняются двумя полями – ссылками на левое и правое поддерева. Код алгоритма построения дерева и вывода бинарного дерева на языке С приведен в [8]. Алгоритмы построения дерева, поиска и вывода элементов см. файлах Динамические структуры данных, сортировка и поиск данных.pdf; Бинарные деревья для поиска и сортировки записей.pdf; пример кода на языке Pascal в файле ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ\дерева\ProgramTree.pas. Сравнение возможностей ЯП Pascal и C++ использовании указателей (ссылочных переменных) см. в приложении А).

Пример 20: Размещение вершины (узла) в дереве и заполнение полей:

```
// Пример создания одной вершины в дереве-индексе и
```

```
// заполнения полей по данным из записи массива:
```

```
SN = 0; // Номер записи в массиве
```

```
RN = new TreeNode; // Создали новую вершину
```

```
RN->Inf.StudentNumber = SdntArr[SN].StudentNumber; // заполнили поле - значение ключа из записи в массиве
```

```
RN->Inf.RecNum = SN; // заполнили поле - номер записи в массиве с записанным ключом
```

```
// Левых и правых потомков нет - вставляется листовая вершина
```

```
RN->Left = nullptr;
```

```
RN->Right = nullptr;
```

Для построения всего дерева-индекса необходимо в цикле, считывая каждую запись в массиве данных, вызвать функцию вставки вершины в дерево.

Код функции вставки узла в бинарное дерево для ключей, значения которых уникальны, приведён ниже:

```

unsigned __int8 InsertNode(TreeNode*& RootNode, Student* Arr, unsigned __int8 SN)
// Вставка вершины в бинарное дерево-индекс.
// Аргументы:
// RootNode - ссылка на корень дерева (передается по ссылке)
// Arr - массив записей с данными о студентах (передается по ссылке)
// SN - номер записи в массиве данных о студентах
{ unsigned __int8 Res = -1;
  if (RootNode == nullptr) // Нашли место для вставки вершины – вставляем –
  { // создаём новую вершину - включаем по ссылке в дерево:
    RootNode = new TreeNode;
    // Заполняем поле - значение ключа из записи в массиве:
    RootNode->Inf.StudentNumber = Arr[SN].StudentNumber;
    // Заполняем поле - номер записи в массиве с записанным у вершину ключом:
    RootNode->Inf.RecNum = SN;
    // Левых и правых потомков нет - вставляется листовая вершина
    RootNode->Left = nullptr;
    RootNode->Right = nullptr;
    Res = SN; // Вставили успешно - вернули номер вставленной записи
  }
  else
  {
    if (RootNode->Inf.StudentNumber == Arr[SN].StudentNumber)
    { // Не смогли вставить в индекс новую вершину - запись с таким ключом уже есть
      Res = -1;
    }
    else
    { // Ищем место для вставки в левом или правом поддереве:
      if ( Arr[SN].StudentNumber > RootNode->Inf.StudentNumber)
        Res = InsertNode(RootNode->Right, Arr, SN); // Вставка в правое поддерево
      else Res = InsertNode(RootNode->Left, Arr, SN); // Вставка в левое поддерево
    }
  }
}

```

Разработайте функции

- *Вывода данных* (записей из массива) по возрастанию/убыванию значений ключевого атрибута, по которому выполнена индексация. При реализации функций выполняется обход дерева и по указанным в узлах номерам выбираются записи из массива, содержащего введённые данные, которые и нужно вывести на экран, отделив записи при выводе, например, пустыми строками.

Рекомендация:

Алгоритмы обхода дерева-индекса приведены в файле *Бинарные деревья для поиска и сортировки записей.pdf*. Для вывода данных (полей) из массивов с номерами записей, содержащихся в вершинах дерева, выполняется следующий код:

```

// Выбор записи в массиве по номеру, содержащемуся в узле дерева в соответствующем поле:
SN = RN->Inf.RecNum; // выбрали номер записи в массиве по ссылке RN на узел в дереве
// Далее данные из всех полей записи с номером SN в массиве выводятся, как показано выше

```

- *Поиска элементов по значениям ключевого атрибута* с использованием бинарного дерева (выполняется обход дерева). В случае, если элемент не найден, должно быть выведено соответствующее сообщение. Если элемент с заданным для поиска значением ключа найден в индексе, должны быть выведены значения всех атрибутов из соответствующей записи в массиве введённых данных.

Рекомендация:

При поиске нужного элемента по ключу выполняется обход дерева. При этом выполняется сравнение ключа, записанного в вершине дерева, и в зависимости от результатов сравнения поиск либо продолжается, либо выводится найденное значение или возвращается номер найденной записи в массиве в указанном значении ключа. Если просмотр выполнен до вершины-листа, а значение не найдено, возвращается признак «неудачи» при поиске или выводится соответствующее сообщение.

- Редактирования записей в массиве с возможностью изменения значений атрибутов, по которым выполнена индексация записей массива в бинарном дереве. При этом соответствующие изменения вносятся и в индексы.
- Удаления записи с заданным значением ключевого атрибута из массива с соответствующим изменением индекса (бинарного дерева).

Задание 3. Создание линейных списков с сортировкой элементов

Разработайте структурный тип данных для описания элемента линейного списка с разработанной структурой представления данных об объектах предметной области. Организуйте ввод данных с клавиатуры – напишите функцию ввода и включения данных в линейный список. Элементы в списке должны быть упорядочены, как минимум, по значениям двух атрибутов, по которым выполняется поиск (см. приложение Г). Линейный список не является индексом! Его элементы могут содержать полные данные, которые могут быть перенесены из массива данных, введены с клавиатуры или из файла при создании списка (см. файл Динамические структуры данных, сортировка и поиск данных.pdf).

Разработайте функции

- Ввода новых записей и включения их в линейный список с упорядочением по выбранным атрибутам по возрастанию/убыванию их значений.
- Просмотр (вывод) записей из линейного списка в порядке их ввода.
- Просмотр (вывод) записей из линейного списка в порядке возрастания/убывания значений выбранных для сортировки атрибутов.
- Поиск и вывод записи/записей с заданным значением атрибута, по которому выполнена сортировка (значений всех атрибутов найденной записи, а если запись не найдена, то должно быть выведено соответствующее сообщение).
- Удаление записи с заданным значением атрибута, по которому выполнена сортировка.

Примечание: Коды алгоритмов построения линейного списка с дисциплинами обслуживания LIFO и FIFO на языке C приведен в [9-10]. При выполнении см. примеры в файлах папки ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ\списки.

Отчёт о выполнении самостоятельной работы – файл в формате MS Word – загружается в LMS в установленные сроки. Отчёт включает результаты выполнения всех заданий. Результаты выполнения каждого задания описываются под заголовком первого уровня. За титульным листом вставляется страница с оглавлением (содержанием) отчёта. Файл именуется с использованием шифра подгруппы и фамилии и инициалов автора. Например:

РИС-24-9-Фамилия_ИО (СР).docx
РИС-24-6-Фамилия_ИО (СР).zip) .

Скриншоты в тексте должны быть сжаты для загрузки в LMS.

При нарушении правил оформления отчёта работа не будет проверяться. После дедлайна отчёты не принимаются. Ответы на вопросы в ходе защиты могут повлиять на оценку.

Критерии оценки приведены в файле «Критерии СР ТОИ.xlsx»

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы на деревьях. Что такое бинарные деревья поиска. URL: https://ru.hexlet.io/courses/algorithms-trees/lessons/binary/theory_unit
2. Функция qsort. URL: <http://cppstudio.com/post/891/>
3. Бинарный поиск в массиве через рекурсию на C++. URL: <https://s3.hpc.name/thread/i411/69923/binarnyy-poisk-v-massive-cherez-rekursiyu-na-c.html>
4. Двоичный (бинарный) поиск C++. URL: <https://purecodecpp.com/archives/1977>
5. Бинарный поиск в C++: подробное руководство. URL: <https://codelessons.dev/ru/binarnyj-poisk-po-massivu-c/>
6. Бинарный поиск. URL: <https://ikcprog.github.io/topics/binsearch/>
7. std::binary_search. URL: https://en.cppreference.com/w/cpp/algorithm/binary_search
8. Керниган Б., Ритчи Д. Язык программирования C. 2-е издание: Пер. с англ. "Вильямс", 2009. - 304 с. URL: https://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf
9. Уинер Р. Язык Турбо Си. Перевод с англ. М. П. Матекина; под ред. В. В. Мартынюка. – М.: Мир, 1991. – 380 с.
10. Односвязный список. URL: https://learn.c.info/adts/linked_list.html
11. Справка PascalABC.NET. Указатели. URL: <https://pascalabc.net/downloads/pabcnethelp/index.htm?page=LangGuide/Types/pointers.html>
11. Pascal для студентов. Урок Ч: указатели^. URL: <https://pascalbook.narod.ru/pointer.htm>
12. Добавление в список элемента с помощью функции. URL: <https://ru.stackoverflow.com/questions/518970/%D0%94%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA-%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%B0-%D1%81-%D0%BF%D0%BE%D0%BC%D0%BE%D1%89%D1%8C%D1%8E-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8>

ПРИЛОЖЕНИЕ А

ТИП ДАННЫХ «УКАЗАТЕЛЬ» В ЯЗЫКАХ PASCAL И C++

Оперативная память компьютера состоит из ячеек, каждая из которых имеет уникальный номер (адрес). По адресу можно обратиться к любой ячейке памяти.

При адресации памяти используется шестнадцатеричная система счисления, как показано на рисунке А.1. В персональных компьютерах минимальной адресуемой ячейкой памяти является байт. Получить адрес отдельного бита байта невозможно. Несколько подряд идущих ячеек могут группироваться в более крупные конструкции. Обычно полуслово состоит из двух ячеек, слово – из четырех, двойное слово – из восьми. Адресом полуслова (слова, двойного слова) является адрес первой ячейки из соответствующей последовательности ячеек. Полуслово размещается в памяти таким образом, что его адрес обязательно кратен 2; у слова адрес кратен 4, у двойного слова – 8.

Адреса	...	3088	3089	308A	308B	308C	308D	308E	308F
Двойное слово									
Адрес двойного слова: 3088									
Слова									
Адреса слов: 3088, 308C									
Полуслова									
Адреса полуслов: 3088, 308A, 308C, 308E									
Ячейки									
Адреса ячеек: 3088, 3089, 308A, 308B, 308C, 308C, 308E, 308F									

Рисунок А.1 – Организация оперативной памяти компьютера и адресация ячеек

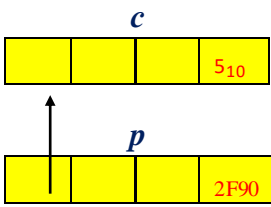
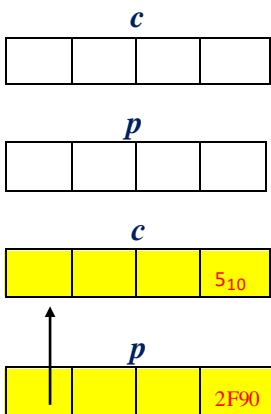
В ЯП существует тип данных «указатель» (*ссылочный тип данных*). Значением переменной типа «указатель» является *адрес (ссылка)*.

Сравнение типа данных «указатель» в языках Pascal и C++ приведено в таблице А.1.

Таблица А.1 – Тип данных «указатель» в языках Pascal и C++

	C++		Pascal	
	Синтаксис	Пример	Синтаксис	Пример
1	2	3	4	5
Символ обозначения ссылочного типа данных	*		^	
Указатель может указывать на данные того типа, который указан при его объявлении:				
Объявление указателя	<тип> * <имя_указателя>; или typedef <имя_типа_указателя> *<тип>;	int * p; float * pf;	var <имя_указателя> ^ <тип>; или type <имя_типа_указателя> = ^<тип>;	var p: ^Integer; pr: ^Real;
		typedef TPF *float; TPF pf;		type tpr = ^ Real; var pr: tpr ;
В PascalABC.NET указатели делятся на типизированные (содержат адрес ячейки памяти данного типа) и бестиповые (содержат адрес оперативной памяти, не связанный с данными какого-либо определенного типа) [11]				
Имя указателя (ссылочной переменной) рекомендуется начинать с префикса p или ptr (от слова «pointer»)				
		int * ptrArray;		var pArray: ^Integer;
Оператор адреса (взять адрес, получить адрес) Унарный префиксный	&	float x; &x // получить адрес x	@	var x: real; @ x { получить адрес x }
Присвоение значения указателю: В ЯП C++ начала должна быть объявлена и инициализирована переменная, затем ее адрес можно записать в указатель Читается: «указатель p ссылается (указывает) на переменную c »	<имя_указателя> = & <имя_переменной>;	int c = 5; int * p = &c ;		var p: ^Integer; c: Integer; begin c := 5; p := @ c; end;
		int a {10}; int *pa { &a }; // указатель pa хранит адрес переменной a		
Еще один вариант присвоения значения указателю в ЯП Pascal:				var p : ^Integer; c: integer; begin new (p); p^ := 5; end;

Продолжение таблицы А.1

1	2		3	4		5
Представление в памяти	<i>int c = 5;</i> <i>int* p = &c;</i>	Адрес 2F90 3DA4		<i>var</i> <i>c: Integer;</i> <i>p: ^Integer;</i> <i>begin</i> <i>c := 5;</i> <i>p := @c;</i> <i>end;</i>	Адрес 2F90 3DA4 2F90 3DA4	
Нулевой/пустой указатель Это специальное значение, которое означает, что указатель ни на что не указывает	<i>null</i> <i>nullpt</i>		<i>int * p = null;</i> // Нулевой указатель	<i>NIL</i> <i>nil</i>		<i>var p: ^integer;</i> <i>begin</i> <i>p := NIL;</i> { пустой указатель } <i>end;</i>
ЯП Pascal: Процедура <i>New(параметр)</i> Создает новую динамическую переменную и устанавливает на нее указатель Процедура <i>Dispose(параметр)</i> Удаляет переменную, созданную с помощью <i>New</i> . После обращения к <i>Dispose</i> , значение указателя-аргумента становится неопределенным и ссылаться на него является ошибкой				Параметр, передаваемый в New – это <i>mun</i> указателя на объект, а не сам указатель		<i>type</i> Str18 = String[18]; <i>var</i> P : ^Str18; <i>begin</i> <i>New(P);</i> P^ := 'Переменная есть...'; <i>Dispose(P);</i> {теперь нет... } <i>P := Nil;</i> {обязательно!} <i>end.</i>

Продолжение таблицы А.1

1	2	3	4	5
Разыменование указателя Для доступа к ячейке памяти, адрес которой хранит указатель (в ЯП Pascal только типизированный), используется операция разыменования	$\ast\langle\text{имя_указателя}\rangle$ $\text{int } c = 5;$ $\text{int}^\ast p = \&c;$ $\text{int } y = \ast p;$ $// y = 5$		$\wedge\langle\text{имя_указателя}\rangle$ var $c: \text{Integer};$ $p: \wedge\text{Integer};$ $y: \text{Integer};$ begin $c := 5;$ $p := @c;$ $y := p^\wedge;$ end.	
Изменение значения переменной через указатель	$\text{int } c = 5;$ $\text{int}^\ast p = \&c;$ $// c = 15$ $// 1. \text{Перейти по адресу, хранящемуся в } p$ $// 2. \text{Изменить значение по этому адресу}$ $\ast p = 15;$		var $c: \text{Integer};$ $p: \wedge\text{Integer};$ begin $\{ \text{указателю присвоили переменной } c \}$ $p := @c;$ $\{ \text{переменной присвоили 15} \}$ $p^\wedge := 15;$ end.	

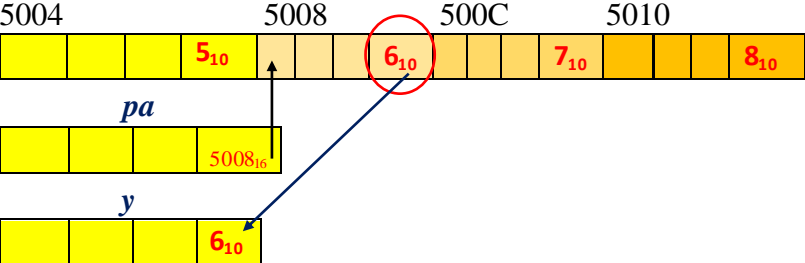
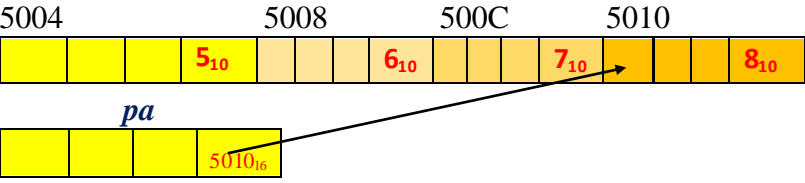
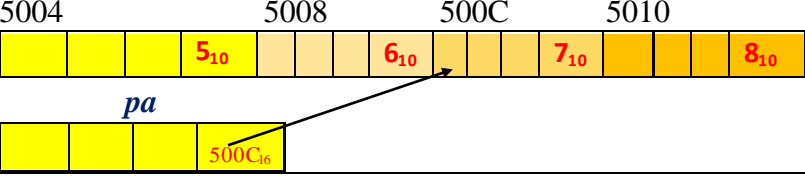
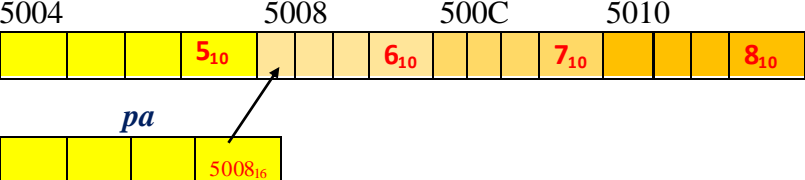
Продолжение таблицы А.1

1	2	3	4	5
<p>Указатель – это переменная. Можно присваивать значения указателей друг другу. Так можно получить несколько ссылок на одну и ту же переменную и ее значение</p>	<pre> int a = 5, b = -5, ya, yb; int *pa = &a, // pa ссылается на a: *pb = &b; // pb ссылается на b ya = *pa; // ya = 5, значению a yb = *pb; // yb = -5, значению b pa = pb; // pa тоже ссылается на b ya = *pa; // ya = -5 yb = *pb; // yb = -5 </pre>	<pre> var a, b, ya, yb: Integer; pa, pb: ^Integer; begin a := 5; b := -5; pa := @a; {pa ссылается на a} pb := @b; {pb ссылается на b} ya := ^pa; {ya = 5} yb := ^pb; {yb = -5} pa := pb; {pa тоже ссылается на b} ya := *pa; {ya = -5} yb := *pb; {yb = -5} end. </pre>		
<p>Указатель – это переменная. Можно сравнивать значения указателей (хранимых в них адресов памяти) на равенство и неравенство</p>	<pre> int a = 5, b = -5, res1, res2, res3; int *pa = &a, // pa ссылается на a: *pb = &b; // pb ссылается на b res1 = pa == pb; // 1) res1 есть "ложь" pa = pb; // pa тоже ссылается на b res2 = pa == pb; // 2) res2 есть "истина" res3 = pa != null; // 3) res3 есть "истина" // в 1) - 3) круглые скобки справа от знака // оператора присваивания не нужны, // т.к. у присваивания самый низкий // приоритет среди операций </pre>	<pre> var a, b, res1, res2, res3: Integer; pa, pb: ^Integer; begin a := 5; b := -5; pa := @a; {pa ссылается на a} pb := @b; {pb ссылается на b} res1 := (pa = pb); // 1) res1 есть "ложь" pa := pb; {pa тоже ссылается на b} res2 := (pa = pb); // 2) res2 есть "истина" res3 := (pa <> NIL); // 3) res3 есть "истина" end. </pre>		


Продолжение таблицы А.1

1		2	3	4	5
ЯП C++ Связь указателей и массивов Имя массива есть адрес его первого элемента (с номером 0). Операция вырезки <code>[]</code> эквивалента операции разыменования указателя <code>*</code> , если указатель связан с массивом		<pre>int y; int a[4]{10, 20, 30, 40}; int *pa = a; // pa = &a[0]</pre> <p>равны $a[0]$ и $*pa$ равны $a[1]$ и $*(pa+1)$ равны $a[i]$ и $*(pa+i)$</p>			
! Теперь вы знаете, почему в C(C++/C#/Java...) элементы массивов по любому измерению нумеруются от 0 !					
Адресная арифметика К указателям <i>можно</i> применять некоторые арифметические операции (сложение, вычитание, инкремент, декремент)		<pre>int y; int a[4] {5, 6, 7, 8}; int *pa = a; // pa = &a[0] y=*a; // y = 5 (y равен содержимому a[0])</pre>			
Инструкция	Адрес	Состояние памяти			
<pre>int y;</pre>	4060 ₁₆				
<pre>int a[4] {5, 6, 7, 8};</pre>	5004 ₁₆				
<pre>int *pa = a</pre>	405C ₁₆				
<pre>y=*a;</pre>	4060 ₁₆				

Продолжение таблицы А.1

1		2	3	4	5
		$y = *(pa++);$ // pa увеличивается на размер данного типа int (на 4), т.е. ссылается на следующий элемент массива a (переходит к адресу элемента a с номером $[0+1]$, это $a[1]$) // $y = 6$ (y равен содержимому $a[1]$)			
... $pa++$...	5004 ₁₆				
		$pa = pa + 2;$ // pa увеличивается на размер данного типа int <u>ДВА</u> раза (на 8), т.е. ссылается на 4-й элемент массива a (переходит к адресу элемента a с номером $[1+2]$, это $a[3]$)			
$pa = pa + 2;$	5004 ₁₆				
		$pa = \&a[2];$ // pa ссылается на 3-й элемент массива a			
$pa = \&a[2];$	5004 ₁₆				
		$pa --;$ // pa уменьшается на размер данного типа int (на 4), т.е. ссылается на 2-й элемент массива a : (переходит к адресу элемента a с номером $[2-1]$, это $a[1]$)			
$pa --;$	5004 ₁₆				

Продолжение таблицы А.1

1		2	3	4	5
	При использовании адресной арифметики следует помнить о том, что приоритет любого унарного оператора выше, чем любого бинарного. Поэтому для <i>int a[4]={10, 20, 30, 40};</i> <i>int *pa = a;</i>				
	<i>*pa + 1;</i>	// 10 + 1= 11	выдает значение, на которое ссылается <i>pa</i> (<i>a[0]</i>), увеличенное на 1		
	<i>*(p1+1)</i>	// 20	выдает значение, хранящееся по адресу <i>pa + 1</i> (значение <i>a[1]</i>)		
	Указатель может ссылаться на переменные структурного типа данных. Оператор доступа к полю (члену) переменной структурного типа (структуры в C++ или записи Pascal)				
			<имя_структуры> -> <имя_члена>	<имя_структуры> ^. <имя_члена>	
			Структурный тип данных Point создан для описания точек на плоскости, использующихся в компьютерной растровой графике (значения координат только целые)		
			// описание структурного типа <i>struct Point</i> <i>{int x;</i> <i>int y;</i> <i>};</i> <i>Point b = {15, 25};</i> // точка b(15; 5) <i>Point * p = &b;</i> <i>int xb = p -> x;</i> // xb=15 <i>int yb = p -> y;</i> // yb=25 <i>*(p -> x)++;</i> // xb++ (xb=16)	{описание структурного типа} <i>Type Point = record</i> <i>x: Integer;</i> <i>y: Integer</i> <i>end;</i> {описание типа указатель на структурный тип} <i>RefPoint = ^Point;</i> <i>var</i> <i>b: Point := (x: 15; y: 25);</i> <i>p: RefPoint;</i> <i>xb: Integer;</i> <i>yb: Integer;</i> <i>begin</i> <i>p := @b;</i> <i>xb := p^.x;</i> {xb=15} <i>yb := p^.y;</i> {yb=25} <i>Inc(p^.x)</i> { xb=xb+1 (xb=16)} <i>end;</i>	

Продолжение таблицы А.1

1	2	3	4	5
Тип данных указатель может часть описания более сложного типа	Структурный рекурсивный тип данных <i>Tree</i> создан для описания узла бинарного дерева с одним информационным полем <i>inf</i> и двумя ссылочными полями <i>Left</i> и <i>Right</i>			
	<pre> struct Tree {float <i>info</i>; // информационное поле <i>Tree</i> *<i>left</i>; // указатель на левое поддерево <i>Tree</i> *<i>right</i>; // указатель на правое поддерево }; </pre>	<pre> Type <i>RefTree</i> = ^<i>Tree</i>; <i>Tree</i> = record {структура узла} <i>Info</i>: <i>Real</i>; {информационное поле} {ссылки на левое и правое поддерева} <i>Left</i>, <i>Right</i>: <i>RefTree</i> end; </pre>		
Ссылочный тип данных должен быть использован при описании формального аргумента функции (параметра процедуры) при его передаче <u>по ссылке</u>	<pre> // обмен местами значений двух // переменных x и y void swap (<i>int</i> * <i>x</i>, <i>int</i> *<i>y</i>) { <i>int</i> <i>r</i> = *<i>x</i>; *<i>x</i> = *<i>y</i>; *<i>y</i> = <i>r</i>; } <i>int</i> <i>a</i> = -2, <i>b</i> = 2; swap(&<i>a</i>, &<i>b</i>); // <i>a</i> = 2, <i>b</i> = -2 </pre>			
Фактическим аргументом должен быть передан адрес				
Для передача аргументом массива можно использовать один из двух эквивалентных способов (фактически в функцию передается указатель на первый элемент массива). Обязательным является также передача аргументом размера этого массива.	<pre> 1) void f1(<i>int</i> <i>numbers</i>[], <i>int</i> <i>n</i>); 2) void f2(<i>int</i> *<i>numbers</i>, <i>int</i> <i>n</i>); <i>int</i> <i>a</i>[4]{10, 20, 30, 40}; <i>int</i> <i>numA</i> = 5; f1(<i>a</i>, <i>numA</i>); f2(<i>a</i>, 5); </pre>			
Фактическим аргументом передается имя массива и его размер .				

Продолжение таблицы А.1

1	2	3	4	5
<p>Односвязный и двусвязный список</p> <p>Односвязный (линейный) список содержит только один указатель на следующий элемент. Этот список имеет один указатель – на начало списка.</p> <p>Связный список, содержащий два поля указателя – на следующий элемент и на предыдущий, называется двусвязным. Этот список имеет два указателя – на начало и на конец списка.</p>	<pre>// Линейный список типа List, у которого // информационное поле есть структура: // имя и возраст студента [12]: struct Student { char* name; int age; }; struct List { Student student; List* next; };</pre>			
<p>Для передачи аргументом начала и/или конца списка следует использовать передачу по ссылке с помощью оператора *, если начало и/или конец не изменяются в функции.</p> <p>Фактическим аргументом должно быть передано имя соответствующего указателя.</p>	<pre>// вывод элементов списка void Print(List *begin) { List* print = begin; while(print) { cout << print->student.name << "->" << print->student.age << endl; print = print->next; } cout<<"NULL"<<endl; } int main() { List *begin = NULL; // ...создание списка ... Print(begin); // вывод списка return 0; }</pre>			

Продолжение таблицы А.1




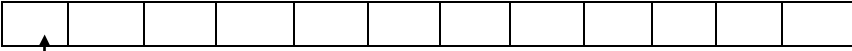

1	2	3	4	5
<p>Для передачи аргументом начала и/или конца списка следует использовать передачу по ссылке с помощью **, если начало и/или конец изменяются в функции.</p> <p>В этом случае правый оператор * сообщает о том, что аргументом является указатель, а левый – о том, что значение этого указателя в функции изменяется, и это изменение должно быть доступно в вызывающей функции</p> <p>Фактическим аргументом должен быть передан адрес соответствующего указателя.</p>	<pre>// инициализация списка List * Init (List **begin) { *begin = new List; (*begin) -> student.name = "Andrew"; (*begin) -> student.age = 20; (*begin) -> next = NULL; return *begin; } int main() { List *begin = NULL; List *end = Init (&begin); return 0; }</pre>			

ПРИЛОЖЕНИЕ Б


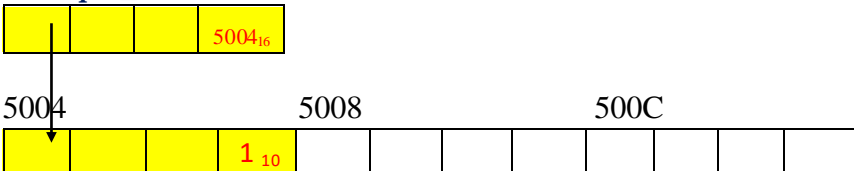

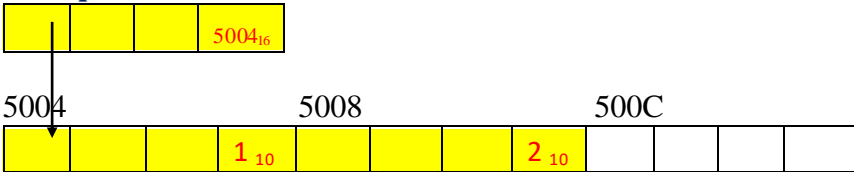

ПРИМЕР РАБОТЫ С ДИНАМИЧЕСКИМ МАССИВОМ В C++

Таблица Б.1 иллюстрирует выполнение инструкций ЯП C++ при создании массива, память для элементов которого выделяется динамически, и работе с ним.

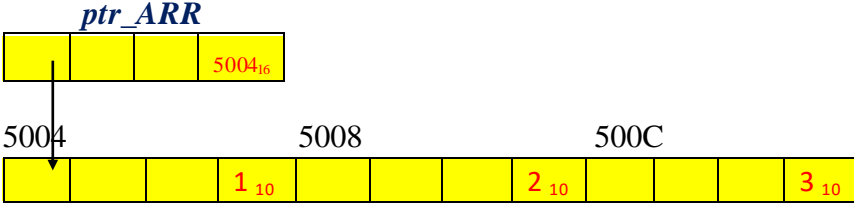
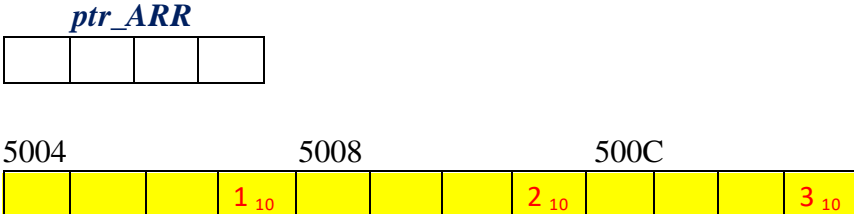
Таблица Б.1– Работа с динамическим массивом в C++

Инструкция C++	Значения переменных	Состояние оперативной памяти	
		Адрес *	Значение в памяти
1	2	3	5
<code>int* ptr_ARR</code>	Не определено	405C ₁₆	<p><i>ptr_ARR</i></p> 
<code>int size_ARR;</code>	Не определено	4060 ₁₆	<p><i>size_ARR</i></p> 
<code>cin >> size_ARR;</code>	3	4060 ₁₆	<p><i>size_ARR</i></p> 
<code>ptr_ARR = (int *) malloc (size_ARR * sizeof(int))</code>			<p><code>sizeof(int) → 4</code> // оператор определения размера типа данных (здесь для int) <code>size_ARR * sizeof(int) = 3 * 4 = 12 →</code> необходимая память для массива – 12 байтов <code>malloc(size_ARR * sizeof(int)) →</code> выделить <u>динамически</u> память 12 байтов</p>
	Не определено Куча	5004 ₁₆	<p><code>(int*)malloc(size_ARR * sizeof(int)) →</code> адрес первого байта этой памяти должен быть кратен размеру типа данных int, т.е. четырём</p> <p>5004 5008 500C</p>  <p><code>ptr_ARR = (int*)malloc(size_ARR * sizeof(int))</code></p> <p><i>ptr_ARR</i></p> 

Продолжение таблицы Б.1

1	2	3	5
<pre>for (int j = 0; j < size_ARR; ...) {</pre>		4064 ₁₆	j 
<pre>ptr_ARR[j] = j + 1</pre>	Куча	405C ₁₆ 5004 ₁₆	$ptr_ARR[j] \rightarrow ptr_ARR + j * sizeof(int) = 5004_{16} + 0*4 = 5004_{16}$ ptr_ARR 
<pre>... ++j) ...</pre>		4064 ₁₆	j 
<pre>ptr_ARR[j] = j + 1</pre>	Куча	405C ₁₆ 5004 ₁₆	$ptr_ARR[j] \rightarrow ptr_ARR + j * sizeof(int) = 5004_{16} + 1*4 = 5008_{16}$ ptr_ARR 
<pre>... ++j) ...</pre>		4064 ₁₆	j 

Продолжение таблицы Б.1

1	2	3	5
$ptr_ARR[j] = j + 1$	Куча	405C ₁₆	$ptr_ARR[j] \rightarrow ptr_ARR + j * \text{sizeof}(\text{int}) = 5004_{16} + 2 * 4 = 500C_{16}$ 
...		5004 ₁₆	
$free(ptr_ARR);$	Не определено	405C ₁₆	
	Куча	5004 ₁₆	

* Значения, использованные в столбце «Адрес» условные, они определяются в IDE компилятора C++.

ПРИЛОЖЕНИЕ В ПРИМЕР ВВОДА ДАННЫХ ИЗ ФАЙЛА-ПОТОКА

Таблица 3.1 иллюстрирует выполнение инструкций ЯП С++ при вводе данных из файла, представленного потоком.

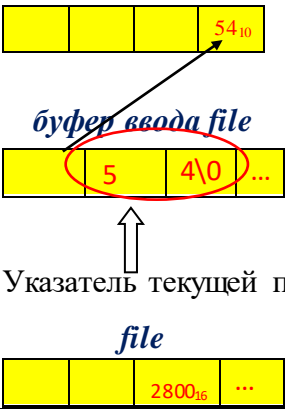
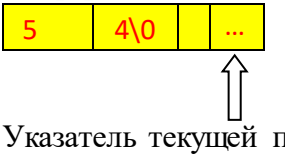

Таблица В.1– Ввод данных из потока в С++

Инструкция С++	Значения переменных	Адрес	Состояние внешней памяти (файла)	Состояние оперативной памяти	
			Значение в файле	Адрес *	Значение в памяти
1	2	3	4	5	8
<code>filesystem::path currentPath = filesystem::current_path();</code>	SR**			4170 ₁₆ 3247 ₁₆	<p><i>currentPath</i></p>
<code>string fileName;</code>	Не определено			4174 ₁₆	<p><i>fileName</i></p>
<code>int value;</code>	Не определено			4178 ₁₆	<p><i>value</i></p>
<code>cin >> fileName;</code> <code>// cin</code>	vxod			5480 ₁₆	<p><i>cin</i></p>
<code>// fileName</code>	vxod			3247 ₁₆	<p><i>fileName</i></p>
<code>ifstream file;</code>	2800 ₁₆			5498 ₁₆ 2800 ₁₆	<p><i>file</i></p>

Продолжение таблицы В.1

1	2	3	4	5	8																	
<code>file.open(currentPath.string() + "\\\" + fileName, ios::in);</code>			<div>Открытие файла с именем "SR\vxod" (значение константы "\\\" есть один обратный слэш)</div> <div>Установка текущей позиции в этом файле на начало (смещение от начала равно 0 байтов)</div> <div>SR\vxod</div> <div><table><tr><td>12</td><td></td><td>54</td><td></td><td>-2</td><td>EOF</td><td></td><td></td></tr></table></div> <div><div>↑ текущая позиция</div><div><table><tr><td>0</td></tr></table></div></div>	12		54		-2	EOF			0	<div>5498₁₆</div> <div>2800₁₆</div>	<div>file</div> <div><table><tr><td>...</td><td>...</td><td>2800₁₆</td><td>...</td></tr></table></div> <div>буфер ввода file</div> <div><table><tr><td>1</td><td>2\0</td><td></td><td>...</td></tr></table></div> <div><div>↑</div><div>Указатель текущей позиции буфера</div></div>	2800 ₁₆	...	1	2\0		...
12		54		-2	EOF																	
0																						
...	...	2800 ₁₆	...																			
1	2\0		...																			
<code>while (file >> value) {</code>	<div>Не конец файла file (считан не EOF)</div> <div>12₁₀</div>	<div>0₁₆</div>		<div>4178₁₆</div> <div>2800₁₆</div> <div>5498₁₆</div>	<div>value</div> <div><table><tr><td></td><td></td><td></td><td>12₁₀</td></tr></table></div> <div>буфер ввода file</div> <div><table><tr><td>1</td><td>2\0</td><td></td><td>...</td></tr></table></div> <div><div>↑</div><div>Указатель текущей позиции буфера</div></div> <div>file</div> <div><table><tr><td></td><td></td><td>2800₁₆</td><td>...</td></tr></table></div>				12 ₁₀	1	2\0		...			2800 ₁₆	...					
			12 ₁₀																			
1	2\0		...																			
		2800 ₁₆	...																			
			<div>SR\vxod</div> <div><table><tr><td>12</td><td></td><td>54</td><td></td><td>-2</td><td>EOF</td><td></td><td></td></tr></table></div> <div><div>↑ Текущая позиция</div><div><table><tr><td>3</td></tr></table></div></div>	12		54		-2	EOF			3	2800 ₁₆	<div>буфер ввода file</div> <div><table><tr><td>1</td><td>2\0</td><td></td><td>...</td></tr></table></div> <div><div>↑</div><div>Указатель текущей позиции буфера</div></div>	1	2\0		...				
12		54		-2	EOF																	
3																						
1	2\0		...																			

Продолжение таблицы В.1

1	2	3	4	5	8						
... file >> value ... // value	Не конец файла file (считан не EOF) 54 ₁₀			4178 ₁₆ 2800 ₁₆ 5498 ₁₆	<p><i>value</i></p>  <p><i>буфер ввода file</i></p> <p>Указатель текущей позиции буфера</p> <p><i>file</i></p>						
			<p><i>SR\vxod</i></p> <table border="1"> <tr> <td>12</td><td>54</td><td>-2</td><td>EOF</td><td></td><td></td></tr> </table> <p>Текущая позиция</p> <div style="border: 1px solid black; padding: 2px; display: inline-block;">6</div>	12	54	-2	EOF			2800 ₁₆	<p><i>буфер ввода file</i></p>  <p>Указатель текущей позиции буфера</p>
12	54	-2	EOF								
... file >> value ... // value	Не конец файла file (считан не EOF) -2 ₁₀			4178 ₁₆ 2800 ₁₆ 5498 ₁₆	<p><i>value</i></p>  <p><i>буфер ввода file</i></p> <p>Указатель текущей позиции буфера</p> <p><i>file</i></p>						

Продолжение таблицы В.1

1	2	3	4	5	8																
			<div><div>SR\vxod</div><table><tr><td>12</td><td></td><td>54</td><td></td><td>-2</td><td>EOF</td><td></td><td></td></tr></table><div>Текущая позиция</div><div>6</div></div>	12		54		-2	EOF			2800 ₁₆	<div><div>буфер ввода file</div><table><tr><td>-</td><td>2\0</td><td>...</td><td>...</td></tr></table><div>↑</div><div>Указатель текущей позиции буфера</div></div>	-	2\0				
12		54		-2	EOF																
-	2\0																		
... file >> value ... }	Конец файла file (считан EOF)																				
...																					
file.close();			<div><div>SR\vxod</div><table><tr><td>12</td><td></td><td>54</td><td></td><td>-2</td><td>EOF</td><td></td><td></td></tr></table></div>	12		54		-2	EOF			2800 ₁₆	<div><div>буфер ввода file</div><table><tr><td></td><td>-</td><td>2\0</td><td>...</td></tr></table><div><div>file</div><table><tr><td></td><td></td><td></td><td>...</td></tr></table></div></div>		-	2\0
12		54		-2	EOF																
	-	2\0	...																		
			...																		
				5498 ₁₆																	

* Значения, использованные в столбце «Адрес» условные, они определяются в IDE компилятора C++.

** Используется условное имя текущей директории

ПРИЛОЖЕНИЕ Г РАБОТА СО СПИСКАМИ

Таблица Г.1 иллюстрирует последовательность состояний односвязного списка при его создании и добавлении в него элементов. Дисциплина обслуживания **FIFO** (**F**irst **I**ntput – **F**irst **O**utput). В этом приложении пустой указатель обозначается символом Θ .

Таблица Г.1 – Формирование односвязного списка целочисленных значений

Операция	Состояние списка до выполнения операции	Состояние списка после выполнения операции
Создание списка	<p>head</p> <div> <div> <div></div> <div>Θ</div> </div> </div>	
Добавить 2	<p>head</p> <div> <div> <div></div> <div>Θ</div> </div> <div> <div>value</div> <div>next</div> <div></div> <div></div> </div> </div>	<p>head</p> <div> <div> <div></div> <div>Θ</div> </div> <div> <div>value</div> <div>next</div> <div>2</div> <div>Θ</div> </div> </div> <p>head</p> <div> <div></div> <div> <div>value</div> <div>next</div> <div>2</div> <div>Θ</div> </div> </div>
Добавить 3	<p>head</p> <div> <div></div> <div> <div>value</div> <div>next</div> <div>2</div> <div>Θ</div> </div> <div> <div>value</div> <div>next</div> <div></div> <div></div> </div> </div>	<p>head</p> <div> <div></div> <div> <div>value</div> <div>next</div> <div>2</div> <div>Θ</div> </div> <div> <div>value</div> <div>next</div> <div>3</div> <div>Θ</div> </div> </div> <p>head</p> <div> <div></div> <div> <div>value</div> <div>next</div> <div>2</div> <div></div> </div> <div> <div>value</div> <div>next</div> <div>3</div> <div>Θ</div> </div> </div>

Чтобы проиллюстрировать процесс создания индексов для односвязного списка, используются данные о студентах, которые поступают в порядке, представленном в таблице Г.2.

Таблица Г.2 – Данные о студентах для их представления односвязным списком

Фамилия	Год рождения	Рейтинг
Рогова	2004	90
Носкова	2006	65
Тырцев	2007	112

Таблица Г.3 иллюстрирует последовательность состояний односвязного списка при создании его индекса с сортировкой рейтинга студентов по возрастанию. В ней используется единственное информационное поле таблицы Г.2 – рейтинг студента. Имя указателя головы списка – hR .

Таблица Г.4 иллюстрирует последовательность состояний односвязного списка при создании его индекса с сортировкой рейтинга студентов по убыванию и сортировки в лексико-графическом порядке по фамилии. В ней используются два информационных поля таблицы Г.2 – рейтинг и фамилия студента, причем последнее представляется первой буквой значений. Имя указателя головы списка по рейтингу – hR , головы по «списку» фамилий – hF .

В таблицах Г.3-Г.4 информационные поля, предназначенные для хранения значений индексов, начинаются с префикса “N”, ссылочные поля – с префикса “next”.

Продолжение таблицы Г.3

1	2	3
Добавить рейтинг 3-го студента	<p>hR</p> <pre> graph TD hR[hR] --> Node1 subgraph Node1 [] direction LR R1[R 90] --- nR1[nR 1] --- nextR1[nextR Θ] end subgraph Node2 [] direction LR R2[R 65] --- nR2[nR 2] --- nextR2[nextR Θ] end subgraph Node3 [] direction LR R3[R] --- nR3[nR] --- nextR3[nextR] end Node1 --> Node2 Node2 --> Node3 </pre>	<p>hR</p> <pre> graph TD hR[hR] --> Node1 subgraph Node1 [] direction LR R1[R 90] --- nR1[nR 1] --- nextR1[nextR Θ] end subgraph Node2 [] direction LR R2[R 65] --- nR2[nR 2] --- nextR2[nextR Θ] end subgraph Node3 [] direction LR R3[R 112] --- nR3[nR 3] --- nextR3[nextR Θ] end Node1 --> Node2 Node2 --> Node3 </pre> <p>hR</p> <pre> graph TD hR[hR] --> Node1 subgraph Node1 [] direction LR R1[R 90] --- nR1[nR 1] --- nextR1[nextR Θ] end subgraph Node2 [] direction LR R2[R 65] --- nR2[nR 2] --- nextR2[nextR Θ] end subgraph Node3 [] direction LR R3[R 112] --- nR3[nR 3] --- nextR3[nextR Θ] end Node1 --> Node2 Node2 --> Node3 </pre>

Таблица Г.4 – Формирование индекса односвязного списка при его создании (сортировка данных студентов по убыванию рейтинга и сортировки в лексико-графическом порядке по фамилии)

Операция	Состояние списка до выполнения операции	Состояние списка после выполнения операции
1	2	3
Создание списка	<div>hR</div> <div>⊖</div>	
Добавить данные 1-го студента	<div>hR</div> <div>⊖</div> <div>hF</div> <div>⊖</div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> <div></div> <div></div> <div></div> <div></div> <div></div>	<div>hR</div> <div>⊖</div> <div>hF</div> <div>⊖</div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> <div>90</div> <div>1</div> <div>⊖</div> <div>P</div> <div>⊖</div>
	<div>hR</div> <div></div> <div>hF</div> <div></div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> <div>90</div> <div>1</div> <div>⊖</div> <div>P</div> <div>⊖</div>	

Продолжение таблицы Г.3

1	2	3
Добавить данные 2-го студента	<div> <div>hR</div> <div>hF</div> <div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> </div> <div> <div>90</div> <div>1</div> <div>⊖</div> <div>P</div> <div>⊖</div> </div> <div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> </div> <div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> </div>	<div> <div>hR</div> <div>hF</div> <div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> </div> <div> <div>90</div> <div>1</div> <div>⊖</div> <div>P</div> <div>⊖</div> </div> <div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> </div> <div> <div>65</div> <div>2</div> <div>⊖</div> <div>H</div> <div>⊖</div> </div> </div> <div> <div>hR</div> <div>hF</div> <div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> </div> <div> <div>90</div> <div>1</div> <div></div> <div>P</div> <div>⊖</div> </div> <div> <div>R</div> <div>nR</div> <div>nextR</div> <div>F</div> <div>nextF</div> </div> <div> <div>65</div> <div>2</div> <div>⊖</div> <div>H</div> <div></div> </div> </div>

Продолжение таблицы Г.3

1	2	3
Добавить данные 3-го студента	<p>Diagram showing a linked list structure with two nodes. The first node contains R=90, nR=1, nextR=, F=P, nextF=Θ. The second node contains R=65, nR=2, nextR=Θ, F=H, nextF= (points to the first node). Arrows from hR and hF point to the first and second nodes respectively.</p>	<p>Diagram showing a linked list structure with three nodes. The first node contains R=90, nR=1, nextR=, F=P, nextF=Θ. The second node contains R=65, nR=2, nextR=Θ, F=H, nextF= (points to the first node). The third node contains R=112, nR=3, nextR=Θ, F=T, nextF=Θ. Arrows from hR and hF point to the first and second nodes respectively.</p>

